MTH6101 Introduction to Machine learning- 2023/24

HMA

April 9, 2024

Printed notes

This booklet contains notes and problem questions for the Module "MTH6101 Introduction to Machine learning" The booklet complements lectures and thus it is **not** a substitute for attending lectures. The printed notes and the scanned notes are also **not** a substitute for your own handwritten notes. In this Module, laboratory work is essential and not an optional activity.

Note that everything in these notes and what is covered in lectures, coursework and labs is examinable.

A note on the Calendar: Computer labs will start in week 2.

Update: In week 12, there will be no Friday lab but we'll have the midterm test.

Week one

This Module is concerned with four topics

1. Principal component analysis

2. Clustering data

3. Classification

4. Lasso and regularization

Formulæ and review of concepts from linear algebra

Before the first Module topic, we review eigenvalues and eigenvectors of a square matrix. Then we look at **two** important matrix factorizations. The first is the Karhunen-Loeve decomposition for square matrices, and the second is the singular value decomposition of a matrix.

Eigenvalues and eigenvectors of a matrix

Consider a real valued **square** matrix $\mathbf{A}$ of size p (i.e. $p \times p$) which has full rank, that is, an **invertible** matrix. This matrix $\mathbf{A}$ has associated **eigenvectors** $\mathbf{v}_1, \ldots, \mathbf{v}_p$ and **eigenvalues** $\lambda_1, \ldots, \lambda_p$. The eigenvalues are (complex) numbers λ_i which are the roots of the **characteristic polynomial** $\det(\mathbf{A} - \lambda \mathbf{I})$ with $\mathbf{I}$ and identity matrix of the appropriate size. The reproducing property that **eigenvector** $\mathbf{v}_i$ and **eigenvalue** λ_i of $\mathbf{A}$ satisfy is

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i.$$

Because the matrix $\mathbf{A}$ is invertible, it has no zero eigenvalues. We further assume that $\mathbf{A}$ has no repeated eigenvalues and that the associated eigenvectors are linearly independent.

Let $\mathbf{Q}$ be defined as $\mathbf{Q} := (\mathbf{v}_1 \; \mathbf{v}_2 \; \cdots \; \mathbf{v}_p)$, that is a matrix built by collecting the eigenvectors of $\mathbf{A}$ as columns of it and let $\boldsymbol{\Lambda}$ be a diagonal matrix with the corresponding eigenvalues of $\mathbf{A}$ in its diagonal $\boldsymbol{\Lambda} := \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p)$. The following decomposition holds

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

Two matrix decompositions

Other two matrix decompositions are given next. The first is the **spectral decomposition**, also known as the **Karhunen-Loeve decomposition** (**KL**). This decomposition is a specialized result which follows from the fact that if $\mathbf{A}$ is symmetric then all its eigenvalues are real and that eigenvectors corresponding to different eigenvalues are orthogonal.

The Karhunen-Loeve (spectral) decomposition of a square **symmetric** matrix $\mathbf{A}$ of size p is the factorization

$$\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T,$$

where $\mathbf{Q} = (\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_p)$ is the same matrix of columns with eigenvectors as above, but now the columns (eigenvectors) of $\mathbf{A}$ are an orthonormal set, that is $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$. It is standard practice that the diagonal matrix $\boldsymbol{\Lambda}$ contains the eigenvalues of $\mathbf{A}$ written in decreasing order. The KL (spectral) decomposition of $\mathbf{A}$ can also be written as

$$\mathbf{A} = \sum_{i=1}^{p} \lambda_i \mathbf{v}_i \mathbf{v}_i^T,$$

that is, a sum of rank one matrices.

Consider a matrix $\mathbf{X}$ of size $n \times p$. The **singular value decomposition** (**SVD**) factorizes $\mathbf{X}$ as

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

where $\mathbf{U}$ is a matrix of size $n \times \min(n,p)$ that satisfies $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and whose columns are eigenvectors of $\mathbf{X}\mathbf{X}^T$; and $\mathbf{V}$ is a matrix of size $p \times \min(n,p)$ that satisfies $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and whose columns are eigenvectors of $\mathbf{X}^T\mathbf{X}$. The matrix $\mathbf{D}$ is a rectangular diagonal matrix of size $\min(n,p) \times \min(n,p)$ that in its diagonal contains the square roots of eigenvalues of $\mathbf{X}\mathbf{X}^T$ in decreasing order.

While the computation of eigenvalues described at the beginning of this revision can only be done for square matrices, the SVD decomposition can be done for any matrix, not necessarily square. An important result is that if $\mathbf{X}$ is square and symmetric, the singular value decomposition coincides with the Karhunen-Loeve (spectral) decomposition.

In a similar manner as with the spectral decomposition, the matrix $\mathbf{X}$ can be written as a sum of rank one matrices, that is

$$\mathbf{X} = \sum_{i=1}^{\min(n,p)} d_i \mathbf{u}_i \mathbf{v}_i^T,$$

where the sum is over the relevant non-zero diagonal elements d_i of $\mathbf{D}$, and $\mathbf{u}_i$ is a (eigenvector) column of $\mathbf{U}$ with n elements and $\mathbf{v}_i$ as a (eigenvector) columns of $\mathbf{V}$ with p elements.

`R` and `RStudio`

You will be using `R` for data analysis and I recommend that you do a review of past `R` material that you may have done. If you have a computer you may want to install the software which can be downloaded for free from `https://www.r-project.org/`. We will use the friendly environment `RStudio` so once you have `R`, `RStudio` can be downloaded from `https://rstudio.com/products/rstudio/download/` We will use the `markdown` capability of `RStudio` so check that it works. If it does not work, you may need to also install the `pandoc` functionality `https://pandoc.org/`

Simple operations

```
2+2 ## Basic arithmetic operations

## [1] 4

2*2; 2-2; pi; 2/0

## [1] 4
## [1] 0
## [1] 3.141593
## [1] Inf
```

Matrix decompositions

```
A<-matrix(ncol=2,byrow=TRUE,c(4,1,-1,0)); A

##      [,1] [,2]
## [1,]    4    1
## [2,]   -1    0

eigen(A) ## eigenvector and eigenvalue calculation

## eigen() decomposition
## $values
## [1] 3.7320508 0.2679492
##
## $vectors
##            [,1]       [,2]
## [1,]  0.9659258 -0.2588190
## [2,] -0.2588190  0.9659258
```

```
svd(A)  ## singular value decomposition
```

```
## $d
## [1] 4.236068 0.236068
##
## $u
##            [,1]      [,2]
## [1,] -0.9732490 0.2297529
## [2,]  0.2297529 0.9732490
##
## $v
##            [,1]       [,2]
## [1,] -0.9732490 -0.2297529
## [2,] -0.2297529  0.9732490
```

Other resources and help

A starting point in `R` is the page `https://www.r-project.org/` and also the page `https://stat.ethz.ch/R-manual/R-devel/library/` For matrix material, a vast manual from `https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf` is “The Matrix Cookbook”.

Exercises matrices

Exercise 1 Eigenvalues, eigenvectors. For each of the following matrices:

$$1:\begin{pmatrix} -1 & -2 & 2 \\ -3 & 5 & -3 \\ -3 & 8 & -6 \end{pmatrix},\ 2:\begin{pmatrix} -6 & 4 & 4 \\ -1 & 2 & 1 \\ -7 & 4 & 5 \end{pmatrix},\ 3:\begin{pmatrix} 1 & 1 & 1 \\ -6 & 3 & 6 \\ 4 & 1 & -2 \end{pmatrix},$$

$$4:\begin{pmatrix} -1 & 3 & -3 & 3 \\ 0 & 2 & -6 & 6 \\ -5 & 3 & 1 & 3 \\ -5 & 3 & 5 & -1 \end{pmatrix},\ 5:\begin{pmatrix} -9 & 1 & 12 & -6 \\ -6 & 4 & 6 & -6 \\ -6 & 1 & 9 & -6 \\ 1 & 0 & -1 & -2 \end{pmatrix},$$

$$6:\begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix},\ 7:\begin{pmatrix} -1 & -2 & 2 \\ 1 & 2 & -4 \\ 1 & 1 & -3 \end{pmatrix},\ 8:\begin{pmatrix} -2 & 4 & -4 \\ -1 & 0 & -1 \\ -1 & -2 & 1 \end{pmatrix}.$$

1. Compute the eigenvector-eigenvalue decomposition of the matrices. Remember to define the matrix using the `R` command `matrix` and then use the function `eigen`.

2. For every pair eigenvector-eigenvalue, verify numerically that they satisfy the reproducing property $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$. Recall that `%*%` is the syntax for matrix product in `R`.

3. Compute and write the characteristic polynomial. Also find numerically its zeroes. For this, use functions `charpoly`, `poly2str` and `polyroots` from library `pracma`.

4. Plot the characteristic polynomial and show graphically that it has zeroes at the eigenvalues you just found. For this, write a `function` using `polyval` and the characteristic polynomial you just computed, and then plot it using `curve`.

Exercise 2 Eigenvalues, eigenvectors (challenging). Doing computations **by hand**, for each of the following matrices write the characteristic polynomial and then determine eigenvalues and eigenvectors:

$$\begin{pmatrix} 1 & -2 \\ 1 & 1 \end{pmatrix},\ \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix},\ \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Comment on your results. Are you surprised?

Exercise 3 Spectral decomposition For each of the following matrices.

$$1:\begin{pmatrix} 0.51111 & 0.17778 & -0.22222 \\ 0.17778 & 0.64444 & 0.04444 \\ -0.22222 & 0.04444 & 0.44444 \end{pmatrix},\ 2:\begin{pmatrix} 1 & 0.13333 & 0.26667 \\ 0.13333 & 0.66667 & -0.4 \\ 0.26667 & -0.4 & 0.73333 \end{pmatrix},$$

$$3:\begin{pmatrix} 0.64444 & -0.22222 & 0.04444 \\ -0.22222 & 0.71111 & 0.17778 \\ 0.04444 & 0.17778 & 0.84444 \end{pmatrix},$$

$$4:\begin{pmatrix} 1.1476 & -0.26529 & 0.22452 & -0.17005 \\ -0.26529 & 1.20698 & 0.01164 & -0.33588 \\ 0.22452 & 0.01164 & 1.28606 & 0.10687 \\ -0.17005 & -0.33588 & 0.10687 & 1.15936 \end{pmatrix},$$

$$5:\begin{pmatrix} 0.63111 & -0.22133 & 0.27727 & 0.15962 \\ -0.22133 & 0.7872 & -0.31364 & -0.38423 \\ 0.27727 & -0.31364 & 0.58496 & 0.06731 \\ 0.15962 & -0.38423 & 0.06731 & 0.59672 \end{pmatrix}$$

1. Compute the Karhunen-Loeve decomposition (spectral decomposition). This is a computation of eigenvalues λ_i and eigenvectors $\mathbf{a}_i$ similar to that of Exercise 1 and for this use the `R` function `eigen`.

2. Using the results you just obtained and with your eigenvalues $\lambda_1, \lambda_2, \ldots$ in decreasing order, write a series of matrices $\sum_{i=1}^{1} \lambda_i \mathbf{a}_i \mathbf{a}_i^T = \lambda_1 \mathbf{a}_1 \mathbf{a}_1^T$, $\sum_{i=1}^{2} \lambda_i \mathbf{a}_i \mathbf{a}_i^T = \lambda_1 \mathbf{a}_1 \mathbf{a}_1^T + \lambda_2 \mathbf{a}_2 \mathbf{a}_2^T$, …, $\sum_{i=1}^{p} \lambda_i \mathbf{a}_i \mathbf{a}_i^T$ which are approximations to the matrix you just decomposed. Comment on the approximation. For this, simply use the eigenvalues and eigenvectors read as `$values` and `$vectors` from the result of the command `eigen`.

3. In a similar form as the previous step and using the same decreasing order for the eigenvectors, compute (with `R`) and write a series of matrices $\sum_{i=1}^{1} \mathbf{a}_i \mathbf{a}_i^T = \mathbf{a}_1 \mathbf{a}_1^T$, $\sum_{i=1}^{2} \mathbf{a}_i \mathbf{a}_i^T = \mathbf{a}_1 \mathbf{a}_1^T + \mathbf{a}_2 \mathbf{a}_2^T$, …, $\sum_{i=1}^{p} \mathbf{a}_i \mathbf{a}_i^T$. Does your result coincide with what you would expect?

Exercise 4 Singular value decomposition For each of the following matrices:

$$1:\begin{pmatrix} -1 & 1 & -1 \\ 1 & 0 & 0 \end{pmatrix},\ 2:\begin{pmatrix} 0 & -1 & -1 \\ 1 & -1 & -1 \end{pmatrix},$$

$$3: \begin{pmatrix} -1 & 0 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ -1 & 1 & 1 & 1 \end{pmatrix},\ 4: \begin{pmatrix} -1 & -1 \\ -1 & 0 \\ -1 & 1 \end{pmatrix}$$

$$5: \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix},\ 6: \begin{pmatrix} -1 & 1 \\ -1 & 0 \\ 1 & 0 \end{pmatrix}$$

1. Compute the singular value decomposition (`R`) and report the results.

2. Using the results you just obtained with eigenvalues $d_1, d_2, \ldots$ sorted out in decreasing order and corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots$ and $\mathbf{v}_1, \mathbf{v}_2, \ldots$, compute a series of approximations of rank one using `R`. These are $d_1\mathbf{u}_1\mathbf{v}_1^T$, $d_1\mathbf{u}_1\mathbf{v}_1^T + d_2\mathbf{u}_2\mathbf{v}_2^T, \ldots$ as appropriate according to the matrix dimensions. Comment on your results. Recall that you will be using the columns of the matrices from the `svd` decomposition together with the eigenvalues.

Exercise 5 Singular value decomposition The singular value decomposition is not restricted to square matrices yet it can be done for square matrices. Do the numerical singular value decomposition for each of the square matrices in Exercises 1 and 3. Comment on the results with special emphasis in the simmilarities and differences between what you obtained with commands `eigen` and `svd`.

Important points and concepts of week one:

1. Eigenvalues and eigenvectors of a **square** matrix; the reproducing property $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$ and the decomposition $\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}$.

2. The Karhunen-Loeve (spectral) decomposition of a square **symmetric** matrix $\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T = \sum_{i=1}^{p} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$.

3. The singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T = \sum_{i=1} d_i \mathbf{u}_i \mathbf{v}_i^T$ of **any** matrix $\mathbf{X}$.

4. Distinguish between the cases above: When you can do each factorization? When do they coincide?

5. You are expected to become proficient with the **numerical manipulation** of the matrices and decompositions using `R` and `RStudio`. The main functions for practice are `eigen()` and `svd()`.

Week two

Preliminaries from Calculus: derivatives and Lagrange multipliers

Consider the **linear form $\mathbf{As}$** and the **quadratic form $\mathbf{s}^T\mathbf{Bs}$**, where $\mathbf{B}$ is a symmetric matrix and $\mathbf{s}$ is a column vector of indeterminates $s_1, s_2, \dots$ and the operations with matrices are well defined. The derivatives of the linear form and of the quadratic form are:

$$\frac{\partial}{\partial \mathbf{s}}(\mathbf{As}) = \mathbf{A} \text{ and } \frac{\partial}{\partial \mathbf{s}}(\mathbf{s}^T\mathbf{Bs}) = 2\mathbf{Bs}.$$

Lagrange multipliers is the name of a procedure to find the extreme points (maximum-minumum) of a function $f(\mathbf{x})$ subject to a series of conditions (constraints) $h_1(\mathbf{x}) = 0, h_2(\mathbf{x}) = 0, \dots, h_m(\mathbf{x}) = 0$. Note that $f(x)$ and the constraints $h_j(\mathbf{x})$ may be functions of more than one input so x is in general a vector $\mathbf{x} = (x_1, \dots, x_l)$. The Lagrange multipliers method consists of forming the **Lagrangian** defined as

$$L = f(\mathbf{x}) - \sum_{j=1}^{m} \lambda_j h_j(\mathbf{x}).$$

The quantities λ_j are known as **Lagrange multipliers**. The collection of extreme points is obtained by solving a **system of simultaneous equations** obtained by computing the derivative of L with respect to all the x_i inputs and also with respect to the Lagrange multipliers λ_j. This system of equations has $l + m$ equations. If the solution exists, it satisfies all the conditions.

Preliminaries from Statistics: moments and variance matrices

Moments for populations

Consider a multivariate random vector $\mathbf{X} = (X_1, X_2, \dots, X_p)^T \in \mathbb{R}^p$, here p is the number of variables. The **vector of expectations** (population means) is

$$E(\mathbf{X}) = (E(X_1), E(X_2), \dots, E(X_p))^T.$$

The covariance between variables X_1 and X_2 is the usual $Cov(X_1, X_2) = E(X_1X_2) - E(X_1)E(X_2)$ so that the **covariance matrix** of $\mathbf{X}$ (variance-covariance matrix) is the symmetric matrix of size $p \times p$ whose diagonal contains variances of the variables and off diagonal entries are pairwise covariances. It is defined as

$$Cov(\mathbf{X}) = E(\mathbf{X}\mathbf{X}^T) - E(\mathbf{X})E(\mathbf{X})^T.$$

The **total variance** of $\mathbf{X}$ is $\sum_{i=1}^{p} Var(X_i) = tr(Cov(\mathbf{X}))$.

Moments of a linear transformation. If $\mathbf{C}$ is a matrix so that the product $\mathbf{CX}$ is well defined, then

$$E(\mathbf{CX}) = \mathbf{C}E(\mathbf{X}) \text{ and } Cov(\mathbf{CX}) = \mathbf{C}Cov(\mathbf{X})\mathbf{C}^T.$$

Sample moments

The **data** is a realization of a multivariate random vector. It is collected in a matrix $\mathbf{X}$ of size $n \times p$. Each of the n rows of $\mathbf{X}$ is one multivariate observation, carried out in each of p variables. The **vector of means** (sample means) is the vector of sample means per variable, that is the row vector of means per column

$$\overline{\mathbf{X}} = \left(\frac{1}{n}\sum_{i=1}^{n} x_{i1}, \frac{1}{n}\sum_{i=1}^{n} x_{i2}, \ldots, \frac{1}{n}\sum_{i=1}^{n} x_{ip} \right) = (\bar{x}_1, \bar{x}_2, \ldots .\bar{x}_p).$$

This vector can be simply written as $\overline{\mathbf{X}} = \mathbf{1}^T\mathbf{X}/n$ where $\mathbf{1}$ is a matrix of ones of size $n \times 1$.

The **sample covariance matrix** (sample variance-covariance matrix) is the symmetric matrix $\mathbf{\Sigma}$ of size $p \times p$ that has the sample variances of each column in its diagonal and entries outside the diagonal are sample covariances between columns. That is, the entry (j, j) of this matrix is the sample variance of the j-th column

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2 = \frac{1}{n-1}\sum_{i=1}^{n} x_{ij}^2 - \frac{n}{n-1}\bar{x}_j^2;$$

and the (j, k) entry is the sample covariance between j- and k-th columns

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) = \frac{1}{n-1}\sum_{i=1}^{n} x_{ij}x_{ik} - \frac{n}{n-1}\bar{x}_j\bar{x}_k.$$

Using matrix operations, this symmetric matrix is

$$\mathbf{\Sigma} = \frac{1}{n-1}\left(\mathbf{X}^T\mathbf{X} - n\overline{\mathbf{X}}^T\overline{\mathbf{X}}\right).$$

If the columns of $\mathbf{X}$ have been centered around their means, then the sample covariance is simply $\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X}/(n-1)$.

Data manipulation: means, centering, scaling

```
X ## running times of 10 persons in three different types of ground

##    Hill Flat Tarmac
## 1    19   17     19
## 2    29   21     25
## 3    26   18     24
## 4    21   29     17
## 5    29   26     26
## 6    26   23     23
## 7    29   22     31
## 8    22   27     22
## 9    17   21     16
## 10   31   20     33

colMeans(x=X) ## means per variable (column)

##   Hill   Flat Tarmac
##   24.9   22.4   23.6

apply(X=X,MARGIN=2,FUN=mean) ## same result for means per variable

##   Hill   Flat Tarmac
##   24.9   22.4   23.6

apply(X=X,MARGIN=2,FUN=sd) ## standard deviation per (variable) column

##     Hill     Flat   Tarmac
## 4.840799 3.893014 5.541761

for(i in 1:3){ print(colnames(X)[i])
  print(X[,i]-mean(X[,i]))} ## center each column

## [1] "Hill"
##    1    2    3    4    5    6    7    8    9   10
## -5.9  4.1  1.1 -3.9  4.1  1.1  4.1 -2.9 -7.9  6.1
## [1] "Flat"
##    1    2    3    4    5    6    7    8    9   10
## -5.4 -1.4 -4.4  6.6  3.6  0.6 -0.4  4.6 -1.4 -2.4
## [1] "Tarmac"
##    1    2    3    4    5    6    7    8    9   10
## -4.6  1.4  0.4 -6.6  2.4 -0.6  7.4 -1.6 -7.6  9.4
```

```r
## single function scale, here X is the data matrix
scale(x=X,center=TRUE,scale=FALSE) ## center not scaling

##    Hill Flat Tarmac
## 1  -5.9 -5.4   -4.6
## 2   4.1 -1.4    1.4
## 3   1.1 -4.4    0.4
## 4  -3.9  6.6   -6.6
## 5   4.1  3.6    2.4
## 6   1.1  0.6   -0.6
## 7   4.1 -0.4    7.4
## 8  -2.9  4.6   -1.6
## 9  -7.9 -1.4   -7.6
## 10  6.1 -2.4    9.4
## attr(,"scaled:center")
##   Hill   Flat Tarmac
##   24.9   22.4   23.6

scale(x=X,center=TRUE,scale=TRUE) ## center and scaling

##          Hill       Flat      Tarmac
## 1  -1.2188071 -1.3871002 -0.83006111
## 2   0.8469676 -0.3596186  0.25262729
## 3   0.2272352 -1.1302298  0.07217923
## 4  -0.8056522  1.6953447 -1.19095725
## 5   0.8469676  0.9247335  0.43307536
## 6   0.2272352  0.1541222 -0.10826884
## 7   0.8469676 -0.1027482  1.33531570
## 8  -0.5990747  1.1816039 -0.28871691
## 9  -1.6319621 -0.3596186 -1.37140531
## 10  1.2601226 -0.6164890  1.69621183
## attr(,"scaled:center")
##   Hill   Flat Tarmac
##   24.9   22.4   23.6
## attr(,"scaled:scale")
##     Hill     Flat   Tarmac
## 4.840799 3.893014 5.541761
```

Data manipulation: variance-covariance matrices

```
## here X is the data matrix
var(x=X) ## raw data

##              Hill       Flat    Tarmac
## Hill   23.4333333 -0.8444444 24.511111
## Flat   -0.8444444 15.1555556 -4.044444
## Tarmac 24.5111111 -4.0444444 30.711111

var(x=scale(x=X,center=TRUE,scale=FALSE)) ## centered data

##              Hill       Flat    Tarmac
## Hill   23.4333333 -0.8444444 24.511111
## Flat   -0.8444444 15.1555556 -4.044444
## Tarmac 24.5111111 -4.0444444 30.711111

var(x=scale(x=X,center=TRUE,scale=TRUE)) ## centered and scaled data

##              Hill       Flat     Tarmac
## Hill    1.0000000 -0.0448093  0.9136886
## Flat   -0.0448093  1.0000000 -0.1874672
## Tarmac  0.9136886 -0.1874672  1.0000000
```

The last result (variance-covariance matrix of centered and scaled data) is now obtained in two different forms.

```
## First form, correlation
cor(x=X) ## correlation matrix of raw data

##              Hill       Flat     Tarmac
## Hill    1.0000000 -0.0448093  0.9136886
## Flat   -0.0448093  1.0000000 -0.1874672
## Tarmac  0.9136886 -0.1874672  1.0000000
## The second form, using standard deviations
VX<-solve(diag(diag(var(x=X)))); VX ## reciprocal of variances per column

##            [,1]      [,2]       [,3]
## [1,] 0.04267425 0.0000000 0.00000000
## [2,] 0.00000000 0.0659824 0.00000000
## [3,] 0.00000000 0.0000000 0.03256151

sqrt(VX) ## reciprocal of standard deviations

##           [,1]      [,2]      [,3]
## [1,] 0.2065775 0.0000000 0.0000000
## [2,] 0.0000000 0.2568704 0.0000000
## [3,] 0.0000000 0.0000000 0.1804481

sqrt(VX)%*%var(x=X)%*%sqrt(VX) ## equal to the correlation

##            [,1]       [,2]       [,3]
## [1,]  1.0000000 -0.0448093  0.9136886
## [2,] -0.0448093  1.0000000 -0.1874672
## [3,]  0.9136886 -0.1874672  1.0000000
```

1 Introduction: What is machine learning

Machine learning is an umbrella term for a collection of **automated** methods for data analysis. The intention is to **detect patterns** in data and then use these patterns to **describe future** behavior (predict) or to perform some **decision making**.

There are several types of algorithms in machine learning. The **two main** types are **unsupervised learning** and **supervised learning**. The difference between the two types is whether there is an **output** available or not.

Unsupervised learning The outputs are not specified in advance. The main aim is to identify distinct groups. Some problems are:

1. Dimensionality reduction. The data has a high number of dimensions and data needs to be mapped into low dimensions while still preserving relevant information. Examples of techniques: **Principal Component Analysis**, Factor analysis, Multidimensional scaling.
2. Clustering. Data needs to be organised in groups "clusters" of similar points. Examples of techniques: **agglomerative clustering**, **k-means clustering**. Gaussian mixtures, Dirichlet process mixtures.

Supervised learning The outputs are known in advance. The aim is to predict targets correctly. Some problems are

1. Classification. Data is assigned to one of several categories, and the task is to predict labels from input data after the model has been trained on labelled data. Examples of techniques are **logistic regression**, **decision trees**, support vector machines, neural networks, random forests.
2. Regression. The analyses predict continuous quantities from input data. Typical methods include **linear regression**, neural networks and Gaussian processes.

Semi-supervised learning This is a mixed approach, such as unsupervised analysis followed by supervised analysis. Examples of techniques are probabilistic models, graph-based semi-supervised learning.

Reinforcement learning Techniques use feedback in the form of rewards to make improvements. Examples of techniques are Q-learning, direct-policy methods.

Some simple questions may help elucidate which problem and type of machine learning technique is most adequate:

Unsupervised learning

1. Dimensionality reduction: What are the most significant features of this data and how can they be summarised?

2. Clustering: Which data points are similar to each other?

Supervised learning

1. Classification: To which category does this data point belong?

2. Regression: Given this input from a data set, what is the likely value of a particular quantity?

Semi-supervised learning

How can labelled and unlabelled data be combined?

Reinforcement learning

What actions will most effectively achieve a desired endpoint?

Exercises concepts

Exercise 6 Here is a list of questions/situations. In each case, identify which type of learning can be applied, and which machine learning problem is the relevant one.

Email scanning Is this email spam or just a regular message?

Insurance What will be the price an insurance policy for new drivers?

Transport Which customers exhibit similar travel behavior to each other?

Banking Is this transaction fraudulent?

Image processing Is this photo in the social media of yourself?

Medical diagnosis Does this X-ray show sign of disease?

Voice processing An app that adapts over time to the users voice.

Housing What would the price of this house be if it were sold today?

Image processing In this photo, is this a car or a bycicle?

Biology How can scientists summarise the behaviour of all 20,000 genes in a particular diseased tissue?

Food quality How many days before this strawberry is ripe?

Image processing How old is this person in this photo?

Exercise 7 Consider the vector $\mathbf{z} = (z_1, z_2, z_3)^T$ and the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 4 & 0 \\ -1 & 1 & 2 \end{pmatrix}.$$

1. Build explicitly $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ and then $\mathbf{Az}$ and $\mathbf{z}^T\mathbf{Bz}$.
2. Compute by hand the derivatives $\frac{\partial}{\partial \mathbf{z}}(\mathbf{Az})$ and $\frac{\partial}{\partial \mathbf{z}}(\mathbf{z}^T\mathbf{Bz})$.
3. Verify that your results coincide with the result given in lectures.

Exercise 8 Do all the steps of Exercise 7 for the following cases.

1. For $\mathbf{A} = \begin{pmatrix} -1 & 1 \end{pmatrix}$ and $\mathbf{z} = (z_1, z_2)^T$.

2. For $\mathbf{A} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$ and $\mathbf{z} = (z_1, z_2)^T$.

3. For $\mathbf{A} = \begin{pmatrix} -2 & 1 & 1 & 2 \\ 1 & 2 & -1 & 2 \end{pmatrix}$ and $\mathbf{z} = (z_1, z_2, z_3, z_4)^T$.

Exercise 9 Do a simple descriptive analysis of data using scatterplots and boxplots and describe any interesting patterns. Use functions such as `pairs`, `boxplot`, `summary` and computing means and variances for

1. the running times data of Page 12,

2. the four quantitative variables of the iris data (loaded with the command `data(iris)`),

3. airquality data. Use the command `data(airquality)` and exclude the variable `month`.

Important points and concepts of week two:

1. At a concept level and operative level, understand and be able to calculate derivatives of linear and quadratic forms.

2. The concept of Lagrange multipliers.

3. The **taxonomy** of Machine Learning. Within it, different **types** of learning and **problems** within types of learning.

4. You are expected to become proficient with basic **numerical manipulation** of data using `R` and `RStudio`. You need to be able to load data from file or type it in, and also compute means per column and sample variance-covariance matrix. The main functions for practice are `var()` and `scale()`.

Week three

2 Principal Component Analysis

The idea behind principal components analysis (PCA) is to study projections of the data that best reproduce the variability in the data. One important objective of this analysis is to reduce dimensionality of the data, that is, with only a few features of the data, to still reproduce the variability present in the whole data set.

Here we present two possible approaches to this: linear projections of the data as well as low rank approximation of the data itself.

2.1 Principal component analysis

Consider the data set collected in the matrix $\mathbf{X}$ of size $n \times p$. Each of the n rows corresponds to an individual and the columns are p measurements taken on an individual. Without lack of generality, assume that every column of the matrix $\mathbf{X}$ has been centered around its mean to have zero mean. This centering of the columns of $\mathbf{X}$ **does not** alter variance-covariance calculations. We want to a create a **linear projection of the data X that has maximal variance**.

2.1.1 First component

Define this projection as $\mathbf{z}_1 = \mathbf{X}\mathbf{a}_1$. Clearly, $\mathbf{z}_1$ has zero mean and its sample variance is just

$$\frac{1}{n-1}\mathbf{z}_1^T\mathbf{z}_1 = \frac{1}{n-1}\mathbf{a}_1^T\mathbf{X}^T\mathbf{X}\mathbf{a}_1 = \mathbf{a}_1^T \underbrace{\left(\frac{1}{n-1}\mathbf{X}^T\mathbf{X}\right)}_{\Sigma} \mathbf{a}_1 = \mathbf{a}_1^T\boldsymbol{\Sigma}\mathbf{a}_1.$$

That is, the sample variance of $\mathbf{z}_1$ is the quadratic form $\mathbf{a}_1^T\boldsymbol{\Sigma}\mathbf{a}_1$, with $\boldsymbol{\Sigma}$ as defined earlier. Note that $\boldsymbol{\Sigma}$ is a matrix of constants.

We want to pick $\mathbf{a}_1$ to maximize the sample variance of $\mathbf{z}_1$. Note that this sample variance can be made arbitrarily large so we constrain the search to vectors $\mathbf{a}_1$ that have unit norm, that is $\mathbf{a}_1^T\mathbf{a}_1 = 1$. So our problem is:

$$\text{Pick } \mathbf{a}_1 \text{ to maximize } \mathbf{a}_1^T\boldsymbol{\Sigma}\mathbf{a}_1 \text{ subject to } \mathbf{a}_1^T\mathbf{a}_1 = 1.$$

This problem can be simply solved by Lagrange multipliers. Here is the Lagrangian:

$$L = \underbrace{\mathbf{a}_1^T\boldsymbol{\Sigma}\mathbf{a}_1}_{\text{variance}} - \lambda(\underbrace{\mathbf{a}_1^T\mathbf{a}_1 - 1}_{\text{unit vector}}).$$

The derivatives of the Lagrangian are simply computed by formulæ and a simultaneous system of equations is formed:

$$\left.\begin{aligned} \frac{\partial}{\partial \mathbf{a}_1} L &= 2\Sigma\mathbf{a}_1 - 2\lambda\mathbf{a}_1 \\ \frac{\partial}{\partial \lambda} L &= \mathbf{a}_1^T\mathbf{a}_1 - 1 \end{aligned}\right\}\text{so that} \quad \begin{gathered} \Sigma\mathbf{a}_1 = \lambda\mathbf{a}_1 \\ \text{and} \\ \mathbf{a}_1^T\mathbf{a}_1 = 1. \end{gathered}$$

On reading the equations obtained above, it is clear that this is an eigenvalue scenario, i.e. the object $\mathbf{a}_1$ is an eigenvector of $\mathbf{\Sigma}$ with unit norm for which the eigenvalue is λ. We want to maximize the variance of $\mathbf{z}_1$ so when using the above result, the variance of $\mathbf{z}_1$ is now

$$\mathbf{a}_1^T \underbrace{\mathbf{\Sigma}\mathbf{a}_1}_{\lambda\mathbf{a}_1} = \lambda\mathbf{a}_1^T\mathbf{a}_1 = \lambda.$$

In order to maximize the variance of $\mathbf{z}_1$, take λ to be the largest eigenvalue of $\mathbf{\Sigma}$ with eigenvector $\mathbf{a}_1$. In summary, the eigenvector $\mathbf{a}_1$ gives the **coefficients** to build the linear combination of variables $\mathbf{z}_1$, while the corresponding eigenvalue gives the sample **variance** of $\mathbf{z}_1$, the first principal component.

2.1.2 Second and following components

After we have picked the first component, we now define $\mathbf{z}_2 = \mathbf{X}\mathbf{a}_2$. We require that the variance of $\mathbf{z}_2$ is maximized as a function of $\mathbf{a}_2$, subject to the vector $\mathbf{a}_2$ having unit norm and being orthogonal to the vector $\mathbf{a}_1$ of the first step.

Exercise 10 *Show that the Lagrangian for the derivation of the second principal component is $L = a_2^T\mathbf{\Sigma}\mathbf{a}_2 - \mu(\mathbf{a}_2^T\mathbf{a}_2 - 1) - \nu(\mathbf{a}_1^T\mathbf{a}_2)$. Complete the derivation by doing derivatives and show the result.*

It turns out that the vector $\mathbf{a}_2$ that maximizes the variance is when $\mathbf{a}_2$ is an eigenvector of $\mathbf{\Sigma}$ with eigenvalue μ. Moreover, the vector $\mathbf{a}_2$ has unit norm and is orthogonal to $\mathbf{a}_1$. To maximize the variance of $\mathbf{z}_2$, we cannot take the largest eigenvalue of $\mathbf{\Sigma}$ as that was done in the first step. We thus take take μ to be the second largest eigenvalue of $\mathbf{\Sigma}$ with eigenvector $\mathbf{a}_2$.

Exercise 11 *Build the Lagrangian and complete the derivation of the third principal component.*

At this moment the pattern is clear: for maximizing the variance of a projection in a sequential manner, the projection vectors and variance of projections are the pairs eigenvector-eigenvalue of the (sample) variance-covariance matrix $\mathbf{\Sigma}$. In short, we are using and following the spectral decomposition of $\mathbf{\Sigma}$. The following paragraph summarizes PCA for samples.

Summary 1 (Principal Component Analysis PCA) Let $\mathbf{\Sigma}$ be the (sample) variance-covariance matrix of centered data $\mathbf{X}$, and let $\mathbf{A}$ and $\mathbf{\Lambda}$ be the matrices of the Karhunen-Loeve (spectral) decomposition of $\mathbf{\Sigma}$ as

$$\mathbf{\Sigma} = \mathbf{A}\mathbf{\Lambda}\mathbf{A}^T.$$

Here $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p)$ is the matrix with eigenvalues of $\mathbf{\Sigma}$ in decreasing order $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_p \geqslant 0$, and these eigenvalues have corresponding eigenvectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p$, collected as columns of matrix $\mathbf{A}$.

The eigenvectors $\mathbf{a}_i$ are known as **loadings** (PC loadings) and the transformed variables $\mathbf{z}_1 = \mathbf{X}\mathbf{a}_1, \mathbf{z}_2 = \mathbf{X}\mathbf{a}_2, \ldots, \mathbf{z}_p = \mathbf{X}\mathbf{a}_p$ are the **scores** (PC scores).

If we collect all the PC scores as columns of a matrix $\mathbf{Z} = (\mathbf{z}_1\ \mathbf{z}_2\ \cdots\ \mathbf{z}_p)$, we have $\mathbf{Z} = \mathbf{X}\mathbf{A}$, and the (sample) variance-covariance matrix of the PC scores is

$$\frac{1}{n-1}\mathbf{Z}^T\mathbf{Z} = \mathbf{A}^T\mathbf{\Sigma}\mathbf{A} = \mathbf{\Lambda},$$

where the last result used the fact that the matrix $\mathbf{A}$ is an orthogonal matrix. In particular, the (sample) variance of the score $\mathbf{z}_i$ is the eigenvalue λ_i, and the **scores are uncorrelated**. Moreover, the **total variance** of $\mathbf{X}$ equals that of $\mathbf{Z}$.

Standard Principal Component Analysis uses the Karhunen-Loeve decomposition $\mathbf{\Sigma} = \mathbf{A}\mathbf{\Lambda}\mathbf{A}^T$. Importantly, note that all the results from PCA can be recovered from the singular value decomposition of the data matrix $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. We substitute this singular value decomposition of $\mathbf{X}$ in the definition of $\mathbf{\Sigma}$:

$$\underbrace{\mathbf{A}\mathbf{\Lambda}\mathbf{A}^T = \mathbf{\Sigma}}_{\text{this is KL}} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X} = \frac{1}{n-1}\underbrace{\mathbf{V}\mathbf{D}\mathbf{U}^T}_{\mathbf{X}^T}\underbrace{\mathbf{U}\mathbf{D}\mathbf{V}^T}_{\mathbf{X}} = \mathbf{V}\left(\frac{1}{n-1}\mathbf{D}^2\right)\mathbf{V}^T.$$

Comparing with the Karhunen-Loeve decomposition, it follows that the matrix of **PC loadings** (eigenvectors of $\mathbf{\Sigma}$) equals the matrix $\mathbf{V}$ of the singular value decomposition, that is $\mathbf{A} = \mathbf{V}$. Also, the **eigenvalue** matrices of both decompositions have the following relation $\mathbf{\Lambda} = \frac{1}{n-1}\mathbf{D}^2$; in other words, for corresponding eigenvalues λ_i of $\mathbf{\Sigma}$ and d_i of $\mathbf{X}$ we have $\lambda_i = \frac{1}{n-1}d_i^2$. Finally, the **PC scores** can be written using the singular value decomposition:

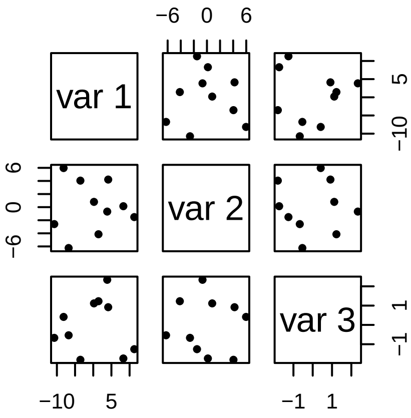
$$\mathbf{Z} = \mathbf{X}\mathbf{A} = \mathbf{X}\mathbf{V} = \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V} = \mathbf{U}\mathbf{D}.$$

Example, runners data

Recall the dataset of running times in different types of ground shown in Page 12. Here we do **three equivalent** versions of PCA. As the units for each variable are the same (minutes), the analysis is done without transforming data.

First version: Karhunen-Loeve decomposition with function `eigen`:

```
VX<-var(X) ## variance-covariance matrix of data (matrix) X
eigen(VX)->E;  ## KL of the variance-covariance matrix
pairs(X%*%E$vectors,pch=16,cex=0.85) ## Plot of PC scores
```



Here are the eigenvalues of KL. When rescaled, they are percentages of variability accounted by each component: 75.33 %, 21.81 %, 2.86 %.

```
E$values

## [1] 52.205300 15.112674  1.982026

E$values/sum(E$values)

## [1] 0.75332323 0.21807610 0.02860067
```

Here are the coefficients of the rotation:

```
E$vectors
```

```
##             [,1]        [,2]       [,3]
## [1,]  0.64706823  0.17947577  0.7410069
## [2,] -0.09729664  0.98339053 -0.1532202
## [3,]  0.75619844 -0.02704643 -0.6537832
```

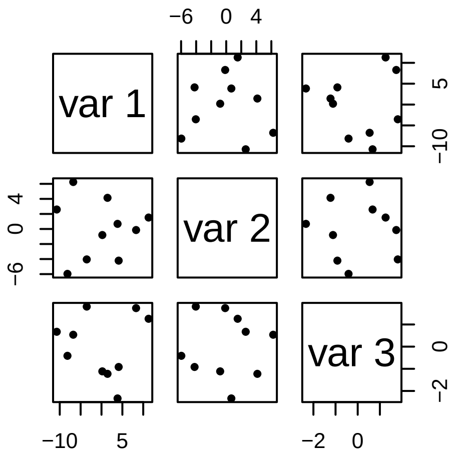
These coefficients are read from the columns above. That is, if the data values (by row) were stored in variables "Hill, Flat, Tarmac", the first principal component would be

0.6471*Hill + -0.0973*Flat + 0.7562*Tarmac.

The eigenvectors (coefficients, PC loadings) are unique up to sign change, so this first component could be computed by reversing all the signs in the first column so coefficients for "Hill, Flat, Tarmac" would be $-0.6471, 0.0973, -0.7562$, respectively. The individual PC1 scores would have sign change, but the variability accounted by it is exactly the same as in the first case above.

Second version: singular value decomposition with function `svd`:

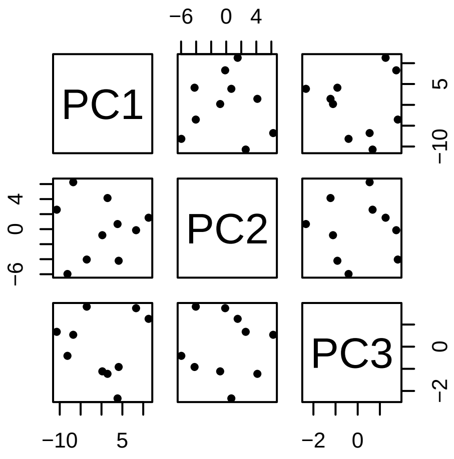
```
S<-svd(X) ## Singular value decomposition of data matrix X
S$d^2/(10-1) ## eigenvalues of KL

## [1] 52.205300 15.112674  1.982026

S$v # PC loadings

##              [,1]        [,2]       [,3]
## [1,]  0.64706823 -0.17947577 -0.7410069
## [2,] -0.09729664 -0.98339053  0.1532202
## [3,]  0.75619844  0.02704643  0.6537832

pairs(S$u%*%diag(S$d),pch=16,cex=0.85) ## PC scores
```



Note that all quantities are equal up to sign (see comment in Page 25) thus diagrams may be reflected.

Third version: function `prcomp`:

```
PCA<-prcomp(x=X); summary(PCA); pairs(PCA$x,pch=16,cex=0.85) ## PC scores

## Importance of components:
##                           PC1    PC2    PC3
## Standard deviation     7.2253 3.8875 1.4078
## Proportion of Variance 0.7533 0.2181 0.0286
## Cumulative Proportion  0.7533 0.9714 1.0000
```



```
PCA$rotation

##                PC1         PC2        PC3
## Hill    0.64706823 -0.17947577 -0.7410069
## Flat   -0.09729664 -0.98339053  0.1532202
## Tarmac  0.75619844  0.02704643  0.6537832
```

Exercise 12 Perform PCA analysis for the `iris` data set of Exercise 9. Do the analysis for both cases: centered only and centered and scaled.

Exercise 13 Consider a data matrix loaded in `R` as `X` (if you want a specific data instance, use the matrix for the runners' data in Page 12). Consider each of the following cases of parameters to be used in the function `scale` before analysing the data in each instance. Comment on the PCA results obtained in each case.

`center`	`scale`	Comments
`FALSE`	`FALSE`	
`FALSE`	`TRUE`	
`TRUE`	`FALSE`	
`TRUE`	`TRUE`	

Exercise 14 For the data sets of Exercise 9, compare and comment the differences between PCA when data has **not** been centered and when data has been centered.

Exercise 15 Do PCA for a data set of your own interest. Interpret your results.

Exercise 16 (Extra) Consider the results from standard linear regression for coefficient estimates $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$ and predicted values $\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$. Using the singular value decomposition of $\mathbf{X}$, develop versions of both formulæ.

Important points and concepts of week three:

1. Understand the (sample) variance-covariance matrix of a centered data set $\mathbf{\Sigma} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$ and be able to compute it either by hand in small example or numerically using software.

2. Understand and compute numerically PCA in its **two** versions: as Karhunen-Loeve decomposition of $\mathbf{\Sigma}$ and using the Singular value decomposition of $\mathbf{X}$. That is, become familiar with the following two factorisations

$$\frac{1}{n-1}\mathbf{X}^T\mathbf{X} = \underbrace{\mathbf{A}\mathbf{\Lambda}\mathbf{A}^T}_{\text{KL-based PCA}} = \underbrace{\mathbf{V}\left(\frac{1}{n-1}\mathbf{D}^2\right)\mathbf{V}^T}_{\text{SVD-based PCA}}.$$

3. Compute and interpret PC **loadings** $\mathbf{A} = \mathbf{V}$ and PC **scores** $\mathbf{XA} = \mathbf{UD}$.

4. Understand the difference in PCA when data has/has not been centered and when data has/has not been scaled.

Week four

2.2 Interpreting Principal Component Analysis (PCA) output

Principal Component Analysis of data produces three output objects: the eigenvalues, which are the variances of principal components; the eigenvectors, also known as PC loadings; and the rotated data, known as PC scores. Using the notation of Karhunen-Loeve decomposition, the eigenvalues are collected in the diagonal of matrix $\mathbf{\Lambda}$, the eigenvectors are columns of $\mathbf{A}$ and the rotated data which are rows of the matrix $\mathbf{XA}$. Each of this objects can be computed and written equivalently using the singular value decomposition, as discussed previously.

2.2.1 How many components to select?

The main objective of Principal Component Analysis is to do a reduction in the dimensionality of the data and selecting a few components that represent the variability of the whole data set is one important aim of this analysis.

A simple graphical tool is to plot the eigenvalues in decreasing order, and if there is a point after which the eigenvalues clearly flatten, this suggests the number of components to take. This is known as the **scree plot**, which is produced in `R` by plotting the output of the command `prcomp`.

A different take is to consider the matrix $\mathbf{Z}_k$ of size $n \times p$ with the first k PC scores $\mathbf{Z}_k = (\mathbf{z}_1 \ \mathbf{z}_2 \ \cdots \ \mathbf{z}_k \ \mathbf{0} \ \cdots \ \mathbf{0})$. The PC scores $\mathbf{z}_1, \mathbf{z}_2, \ldots$ are columns taken from $\mathbf{XA}$ and correspond to the first PC, second PC etc. The **total variance** of $\mathbf{Z}_k$ is $\lambda_1 + \lambda_2 + \ldots + \lambda_k$ so that the percentage of total variability explained by the first k components is

$$\frac{\lambda_1 + \lambda_2 + \ldots + \lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_k + \ldots + \lambda_p}.$$

A table or a plot of this percentage may suggest a suitable number of components.

The **Pareto principle** "few important, most trivial" might be a useful tool to help decide on number of components. In PCA, the Pareto principle states that around 80% of the variability is explained by the 20% of the components. However in a given analysis the results are data dependent and there is **no** guarantee that for a given data set, the 80-20 Pareto principle will hold.

By the **close link** we have seen and studied between the Karhunen-Loeve decomposition of $\mathbf{\Sigma} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$ and the Singular value decomposition of $\mathbf{X}$, the percentage of variability explained by the first k principal components can be equivalently written using eigenvalues d_i of $\mathbf{X}$ as

$$\frac{d_1^2 + d_2^2 + \ldots + d_k^2}{d_1^2 + d_2^2 + \ldots + d_k^2 + \ldots + d_p^2}.$$

2.2.2 Interpretation, scaling data

The PC scores are **linear** combinations of variables and interpreting them is also needed as part of PCA. To interpret the component, look at each eigenvector. The signs and the magnitudes of the entries of each eigenvalue are used to interpret that principal component as **weighted averages** or as **contrasts/comparisons of weighted averages** of variables.

As a general rule, the results of PCA should not depend on the **scale** of units of measurements per variable. There may be exceptions to this, as for example when the variables are measured over the same units. It is however generally recommended that the analysis is carried out on standardized variables.

When on standardized variables, the PCA computations are performed over the sample correlation matrix. In this case, the total variance is equal to p and the proportion of variability explained by the first k components is $\frac{1}{p}\sum_{i=1}^{k}\lambda_i$.

In general, Principal component analyses and conclusions when using scaled (standardized) data do **not** coincide with those obtained with non-standardized data is used. To see this, call $\mathbf{S}$ to the diagonal matrix whose diagonal is that of $\mathbf{\Sigma}$ so that we retrieve the sample correlation matrix $\mathbf{R} := \mathbf{S}^{-1/2}\mathbf{\Sigma}\mathbf{S}^{-1/2}$, which is the variance-covariance matrix for scaled data. We now do this operation on both sides of the Karhunen-Loeve expansion of unscaled data $\mathbf{\Sigma} = \mathbf{A}\mathbf{\Lambda}\mathbf{A}^T$ to have

$$\mathbf{R} = \mathbf{S}^{-1/2}\mathbf{A}\mathbf{\Lambda}\mathbf{A}^T\mathbf{S}^{-1/2}.$$

If this was a Karhunen-Loeve expansion with eigenvalues $\mathbf{\Lambda}$, this should satisfy the reproducing property which for the unscaled data is $\mathbf{\Sigma}\mathbf{A} = \mathbf{A}\mathbf{\Lambda}$. We easily see that this fails, either when right-multiplying by the obvious candidate to eigenvector $\mathbf{S}^{-1/2}\mathbf{A}$ or the different attempt multiplying by $\mathbf{S}^{1/2}\mathbf{A}$. The reproducibility property is not retrieved and thus the eigenvalues -and eigenvectors- of $\mathbf{R}$ (scaled data) differ from those of $\mathbf{\Sigma}$ (unscaled data).

2.2.3 Plots

A pairwise **plot of PC scores**, that is, the rotated variables is a scatterplot of variables that are uncorrelated. This plot can be used to examine the data for patterns such as groups of observations or detection of potential outliers. Recall that the PC scores are the columns of $\mathbf{XA}$ (using KL) or of $\mathbf{UD}$ (using SVD).

A special plot called **biplot** is produced by **overlaying PC loadings** per variable on top of a **scatterplot of PC scores**. The PC loadings are represented as arrows and are the coefficients of the linear transformation, i.e. eigenvalues contained in matrix $\mathbf{A}$ of the K-L decomposition of $\mathbf{\Sigma}$. This plot is useful to further describe data by looking at the contributions of individual observations with respect to variables in the Principal Component Analysis.

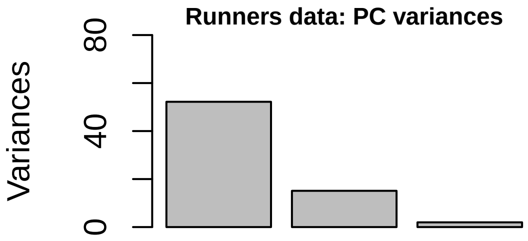
Example, runners data

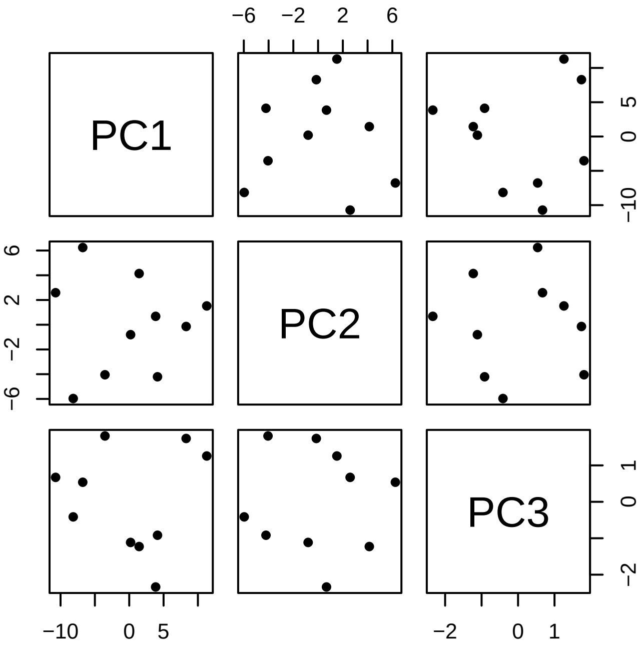
Continuing with the running times data shown in Page 12, and the codes already shown earlier. Here I only comment on results.

Unscaled data: One component is enough to recover over 80% of the total variability. This first component is a weighted average of all the variables, with `Hill` and `Tarmac` having similar magnitude than `Flat`.

```
## Importance of components:
##                           PC1    PC2    PC3
## Standard deviation     7.2253 3.8875 1.4078
## Proportion of Variance 0.7533 0.2181 0.0286
## Cumulative Proportion  0.7533 0.9714 1.0000
##                PC1         PC2        PC3
## Hill    0.64706823 -0.17947577 -0.7410069
## Flat   -0.09729664 -0.98339053  0.1532202
## Tarmac  0.75619844  0.02704643  0.6537832
```

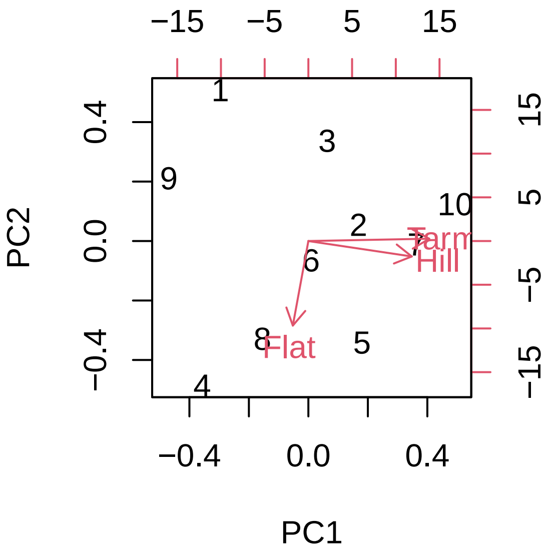
The selection of only one principal component is supported by the scree plot.





The pairwise scatterplot of PC projections (PC scores) may help in determining if there are groups in data or potential outliers. In this case there are too few data points and it is not evident some unusual pattern.

In the biplot, the position of arrows indicates how much each variable contributes to the component. Variable `Hill`, for instance, while not being the main contributor by magnitude to the first component, contributes little to the second component.



In the biplot we can also see that e.g. observations 9 and 10 have opposing values relative to the variable `Tarmac`. We look at the raw observations.

```
##    Hill Flat Tarmac
## 9    17   21     16
## 10   31   20     33
```

The opposing values mean that, for two runners, the times on `Tarmac` were very different. This becomes clearer when we look at the centered data.

```
##    Hill Flat Tarmac
## 9  -7.9 -1.4   -7.6
## 10  6.1 -2.4    9.4
```

For variable `Flat` a similar opposing pattern is observed for observations 1 and 4. Here are the entries of the raw data

```
##   Hill Flat Tarmac
## 1   19   17     19
## 4   21   29     17
```

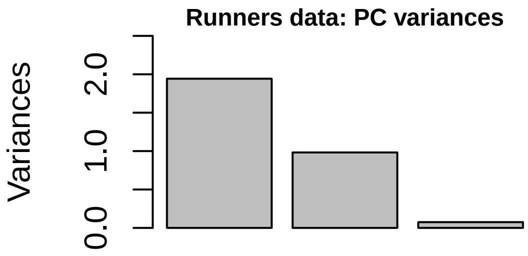
and of the centered data.

```
##   Hill Flat Tarmac
## 1 -5.9 -5.4   -4.6
## 4 -3.9  6.6   -6.6
```

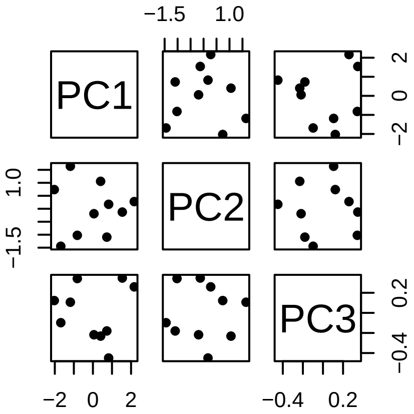
Scaled data: Here two components are required to recover at least 80% of total variability. With relatively few columns, the advantage of PCA is less evident as dropping only one dimension is not a huge dimensionality reduction.

```
## Importance of components:
##                           PC1    PC2     PC3
## Standard deviation     1.3937 0.9912 0.27410
## Proportion of Variance 0.6475 0.3275 0.02504
## Cumulative Proportion  0.6475 0.9750 1.00000
##               PC1         PC2        PC3
## Hill    0.6898224 -0.19997668 -0.6958120
## Flat   -0.1726480 -0.97880552  0.1101473
## Tarmac  0.7030915 -0.04414842  0.7097276
```

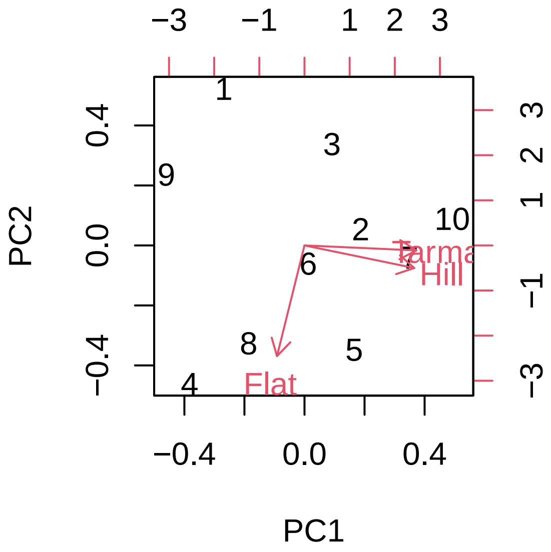
The first component is an average of the three variables, with smaller weight in variable `Flat`. The second component is mostly variable `Flat` compared against an average of the other two variables.



The selection of only two principal components is seen in the scree plot, and the scatterplot of PC scores does not indicate something extraordinary.



As in every biplot, the position of arrows indicates how much each variable contributes to the component. Variables `Tarmac` and `Hill`, contribute little to the second component, while the three variables contribute to the first. The fact that arrows point in the same direction is an obvious consequence of the PC loadings having the same sign.



We can establish similar descriptions of individuals with respect to variables as in the cases described in the **unscaled** case.

Exercise 17 Complete PC analysis for the datasets described in Exercises 9 and 14. Your complete analysis includes: determining number of PCs, interpretation of PC loadings and of descriptive plots of raw data, of PC scores and biplot.

Exercise 18 Ditto as per Exercise 17 for other datasets in `R` such as

the data from black cherry trees (command `data(trees)`);

Intercountry Life-Cycle Savings Data (use `data(LifeCycleSavings)`);

data for Motor Trend Car Road Tests (`data(mtcars)`) and

data with measurements on petroleum rock samples (`data(rock)`).

Also perform PCA for a couple of relatively large data sets:

the dataset on Daily Closing Prices of Major European Stock Indices, 1991-1998 (`data(EuStockMarkets)`), and

the data with prices of 50,000 round cut diamonds (`data(diamonds)`).

In these latter cases, as well as standard PCA, comment upon close inspection of resulting plots.

Exercise 19 Using the theoretical results of the Karhunen-Loeve expansion, show that the total variance of $\mathbf{Z}_k$ is $\lambda_1 + \lambda_2 + \ldots + \lambda_k$. Repeat your derivation of the total variance of $\mathbf{Z}_k$ using results from SVD.

Important points and concepts of week four:

1. The output of Principal components has three elements: the PC variances, PC loadings and PC scores. Become familiar with them, identify them and understand how to compute them. In particular:

2. determine number of components to select. If no other threshold has been established, use 80% of the total variability;

3. interpret PC loadings, that is contributions of variables in each column (eigenvector) of $\mathbf{A}$ and

4. to further study and describe the data, use scatterplots of PC scores (projections) and the biplot which doubles with PC loadings (coefficients).

Week five

3 Clustering methods

Clustering methods aim to detect groups in the individuals that compose the data. We discuss two possible approaches to this: the **agglomerative clustering** approach and the **K-means** method. We first describe how to create distances between individuals.

3.1 Distances and distance matrix

To perform clustering, **distances** between individuals must be computed. Recall that each individual refers to a row of the data matrix $\mathbf{X}$, so that the distance between individuals i and j is the quantity d_{ij} that reflects how close are the i-th row $(x_{i1}, x_{i2}, \ldots, x_{ip})$ and the j-th row $(x_{j1}, x_{j2}, \ldots, x_{jp})$ of the data matrix.

Assuming that the entries of the data matrix are real values, some commonly known distances are the '**Manhattan**', '**Euclidean**' and the '**Minkowski**' distances, see Table 1. Other distances that depend on variances and correlations are the Mahalanobis distance and the absolute correlation. In the above table, to keep with literature, the quantity $\mathbf{x}_{i\cdot}$ in the Mahalanobis distance is the i-th row of the matrix $\mathbf{X}$ but written as column, and the quantity ρ_{ij} is the (sample) correlation between rows i and j.

Once a distance has been selected, the information is collected in the distance matrix (d_{ij}). This is a symmetric matrix of size $n \times n$ with value d_{ij} in the entry (i, j). Note that the diagonal of this matrix is exactly equal to zero.

The Manhattan distance and the Euclidean distance are particular cases of the Minkowski distance and note that these satisfy the requirements of a **metric** for general real points. In the case of the data points and the distance matrix, the properties of a metric are: the distance is non-negative $d_{ij} \geqslant 0$; the distance is symmetric $d_{ij} = d_{ji}$; it satisfies the triangle inequality $d_{ij} \leqslant d_{ik} + d_{kj}$ and finally, $d_{ij} = 0$ if and only if $i = j$.

3.2 Agglomerative clustering

Agglomerative clustering is a sequential procedure to cluster n individuals. **Initially each individual is its own cluster**. At every step, an **updated table of distances between clusters** (dissimilarities) is considered and minimized to merge another individual into a cluster. At the end of the procedure, **a single cluster** is achieved. The process is depicted in a special plot termed a **dendrogram**.

Name	Distance d_{ij}
Manhattan	$\sum_{l=1}^{p} \left\lvert x_{il} - x_{jl} \right\rvert$
Euclidean	$\sqrt{\sum_{l=1}^{p} \left(x_{il} - x_{jl} \right)^2}$
Minkowski	$\left(\sum_{l=1}^{p} \left\lvert x_{il} - x_{jl} \right\rvert^m \right)^{1/m}$
Mahalanobis	$\sqrt{(\mathbf{x}_{i\cdot} - \mathbf{x}_{j\cdot})^T \, \mathbf{\Sigma}^{-1} \, (\mathbf{x}_{i\cdot} - \mathbf{x}_{j\cdot})}$
Absolute correlation	$\sqrt{1 - \lvert\rho_{ij}\rvert}$

Table 1: Common distances between individuals.

In the initial step of agglomerative clustering each point is its own cluster and distance between clusters i and j (at this initial step, the clusters are individuals i and j) is computed by the entry d_{ij} of the distance matrix (d_{ij}). However, apart from this step, we need to compute the distance between clusters that have in general, **more than one point**. This distance is also referred to as **dissimilarity**. There are several proposals to construct dissimilarities ("distances") between clusters of points:

• **single linkage** The dissimilarity between two clusters is that between the closest two points, hence this is known as "nearest neighbor".

• **average linkage** The dissimilarity between two clusters is the average of distances between them, hence the name "average neighbor"

• **complete linkage** The dissimilarity between clusters is that between the two points that are furthest apart, hence "farthest neighbor"

• **Ward linkage**

After the initial step, clustering proceeds by an updated dissimilarity matrix. The minimum non diagonal entry of this matrix is found and this determines the next cluster to be formed. A new updated matrix is produced and the method continues until there is only a single cluster.

3.2.1 Example of clustering

To motivate with an example, consider the following a data set with $n = 5$ points with integer coordinates. The data is given below.

x_1	x_2
0	4
3	6
6	2
0	5
1	1

In Figure 1 we plot the data, and in the plot we put the row numbers in the corresponding coordinates. For two dimensions it is possible to observe and conjecture from such a plot the existence of clusters, but when data lies in higher dimension it is more difficult to conclude clusters from scatterplots, although the `R` command `pairs` can always be used to construct such scatterplots.

To begin the agglomerative cluster process, we need to compute the dissimilarity matrix. Recall that in the initial step, the dissimilarities are distances. Assume we use 'Manhattan' distance so the matrix of distances d_{ij} is:

	1	2	3	4	5
1	0	5	8	1	4
2	5	0	7	4	7
3	8	7	0	9	6
4	1	4	9	0	5
5	4	7	6	5	0

As expected, the diagonal of this matrix consists of zeroes, and the analysis ignores it. The matrix is symmetric and the rest of the procedure can be achieved by using indistinctly either the upper half or the lower half of this matrix of distances.

At this initial stage, we have five clusters and each point is its own individual cluster. At this point, the smallest distance between clusters in the table is 1 which is that between clusters '1' and '4' and thus **these two clusters are joined** at that distance.

3.2.2 Example of clustering: linkage options for the next step

For the next step in the procedure, we require an updated table of dissimilarities between the current clusters. We have several choices. The simplest is the **single linkage** which produces the following updated table of dissimilarities.

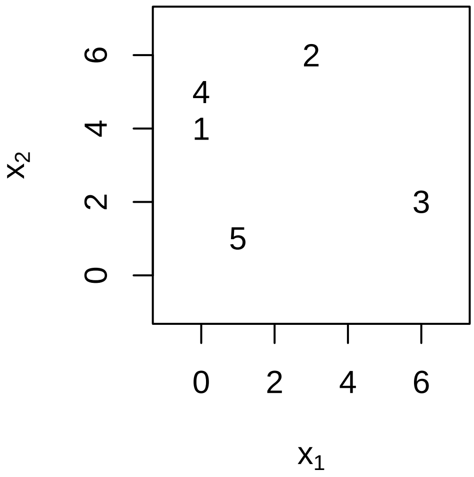


Figure 1: Data of synthetic example of Page 42.

	14	2	3	5
14	0	4	8	4
2	4	0	7	7
3	8	7	0	6
5	4	7	6	0

The updated tables of dissimilarities for the **average linkage** and for the **complete linkage** are:

	14	2	3	5
14	0	4.5	8.5	4.5
2	4.5	0	7	7
3	8.5	7	0	6
5	4.5	7	6	0

and

	14	2	3	5
14	0	5	9	5
2	5	0	7	7
3	9	7	0	6
5	5	7	6	0

3.2.3 Example of clustering: rest of the procedure with single linkage

For the remaining of this example we use the **single linkage** to determine dissimilarities ("distances") between clusters. The next cluster is determined by finding the minimum dissimilarity between clusters in the updated dissimilarity table. This minimum "distance" value is 4 at which point the clusters '14' and '2' are merged.

The updated table of dissimilarities at this stage is the following.

	124	3	5
124	0	7	4
3	7	0	6
5	4	6	0

The procedure continues by finding the minimum dissimilarity between clusters in the newly updated distance table. This minimum value is 4 at which point the clusters '124' and '5' are merged.

This is the updated table of dissimilarities at this stage.

	1245	3
1245	0	6
3	6	0

The last step of this agglomerative clustering process is the trivial merging of the clusters '1245' and '3' at the distance 6.

3.2.4 The dendrogram

The process of agglomerative clustering is summarized in a simple tree-like plot called dendrogram (from the Greek '*dendro*' that means tree). In this plot, the initial individual clusters are represented as leaves (or roots, depending on the orientation of the diagram). As we move along the diagram, the clusters gradually join the trunk at heights determined by the distance metric used and the linkage method used. In one end of the diagram we have n individual clusters and at the other end, all individuals have merged in a single cluster. Note that ordering of the leaves in the dendrogram does not generally correspond to the original ordering of the data.

The height of jumps in the dendrogram suggest the number of clusters that may describe adequately the data. Big jumps in the dendrogram indicate clusters of individuals that are separate and should not be merged. It is difficult in general to suggest a single hard and fast rule for the **metric** and the type of **linkage used**, and the choice may depend on the ultimate use of cluster results.

The following lines reproduce our running example, we use function `agnes` from the library `cluster`.

```
library(cluster); X; A<-agnes(x=X,method = "single",metric="manhattan")

##      [,1] [,2]
## [1,]    0    4
## [2,]    3    6
## [3,]    6    2
## [4,]    0    5
## [5,]    1    1
```

The clustering object in the analysis above is stored in a variable termed `A`. Among other results, this object contains the ordering of points and heights (distances) at which they merge:

```
A$order; A$height

## [1] 1 4 5 2 3
## [1] 1 4 4 6
```

We may want to look at the dendrogram of it, which can be plotted with the simple command `plot(A,which.plot=2)`. This dendrogram for the running example that started in of Page 42 is shown in Figure 2. The clusters are formed at the heights specified by the algorithm output. The same figure also shows the results for other linkages.

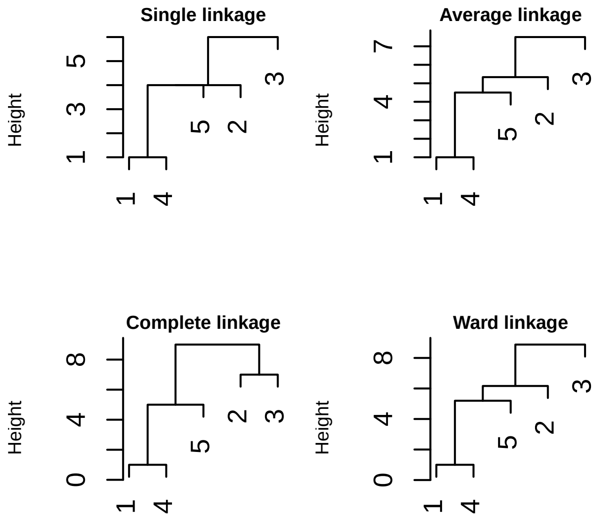
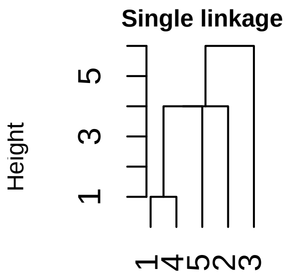


Figure 2: Dendrograms for the example of Page 42 and different linkage methods.

3.2.5 Further exploration with the dendrogram

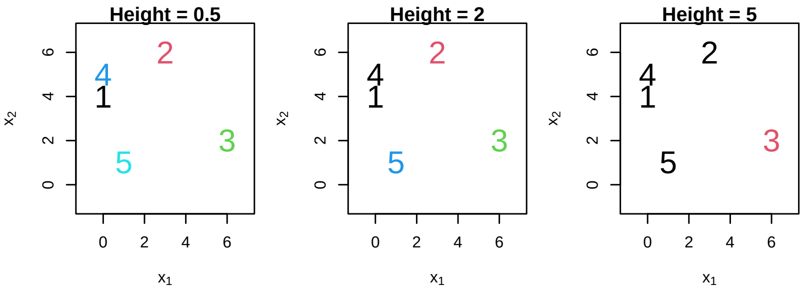
The function `cutree` from library `dendextend` allows manipulation of the dendrogram, to either identify clusters formed at certain heights or a specific number of clusters. In below `A` is the output of the command `agnes` for the first example.

```
plot(A,which.plot=2,main="Single",xlab="",cex.main=0.75,cex.lab=0.75,hang=-1)
```



```
## First cut A at height = 2 then cut A to have 2 clusters
cutree(tree=A,h=2); cutree(tree=A,k=2)

## [1] 1 2 3 1 4
## [1] 1 1 2 1 1
```

The colors of labels correspond to clusters formed at specified heights.



Exercise 20 Perform agglomerative cluster analysis exploring different options for distances (option `metric`) and linkage methods (option `method`). Pick one of the datasets from the package `cluster`, such as `pluton`, `xclara` or `ruspini` (done in lab for one linkage). You may also try the `iris` data as well. In your chosen case, examine your dendrogram and suggest a number of clusters.

Exercise 21 The data set `agriculture` is a small data set with GDP and percentage of population working in agriculture for EU countries in 1993. Perform agglomerative cluster analysis; from the dendrogram suggest a number of clusters and interpret your results.

Exercise 22 Ditto with the `USArrests` data. In this case compare the clusters obtained without scaling the data against scaled data.

Important points and concepts of week five:

1. Become familiar with the procedure of agglomerative clustering.
2. The process of agglomerative clustering does **not** change despite using **different** distances and linkage options. In other words, you only need to become familiar with one general method and there is not need to memorize e.g. nine apparently 'different' methods (three distances and three linkages).
3. Use the function `agnes` from library `cluster` to do agglomerative clustering. Plot and interpret results.

Week six

3.3 Method of K-means

The method known as K-means is a different approach to clustering: the user supplies a desired number of clusters K, and an algorithm attempts to cluster observations in the best way while minimizing a measure of cluster discrepancy. This measure of cluster discrepancy is usually the Error Sum of Squares (ESS), defined as

$$\text{ESS} = \sum_{k=1}^{K} \sum_{i} \left(\mathbf{x}_i - \bar{\mathbf{x}}_k\right)\left(\mathbf{x}_i - \bar{\mathbf{x}}_k\right)^T .$$

In the formula above, the inner sum is performed over i such that $c(i) = k$, that is, all such vectors $\mathbf{x}_i$ belonging to cluster k. The vector $\bar{\mathbf{x}}_k$ is a centroid for cluster k. The choice of $\bar{\mathbf{x}}_k$ leads to **two big variations** of the procedure. One is to have $\bar{\mathbf{x}}_k$ to be the **sample mean** of cluster k. The other case is to use the "**medoid**", where $\bar{\mathbf{x}}_k$ is one of the observations in cluster k such that the inner sum in ESS is minimized.

3.3.1 Simple example

Assume we are interested in building the K-means method with $K = 2$, i.e. two clusters. The data we want to cluster is the set of $n = 5$ points of Page 42:

x_1	x_2
0	4
3	6
6	2
0	5
1	1

From the point of view of software, all we needed is to run a command like `kmeans` for clustering around centroid mean, or `pam` from the library `cluster` for the medoid method. Another medoid function from the same library is `clara`, recommended for larger data sizes. Note that these functions perform random searches hence it is recommended to run them a few times to make sure some stable cluster partition has been achieved.

We first run the K-means method.

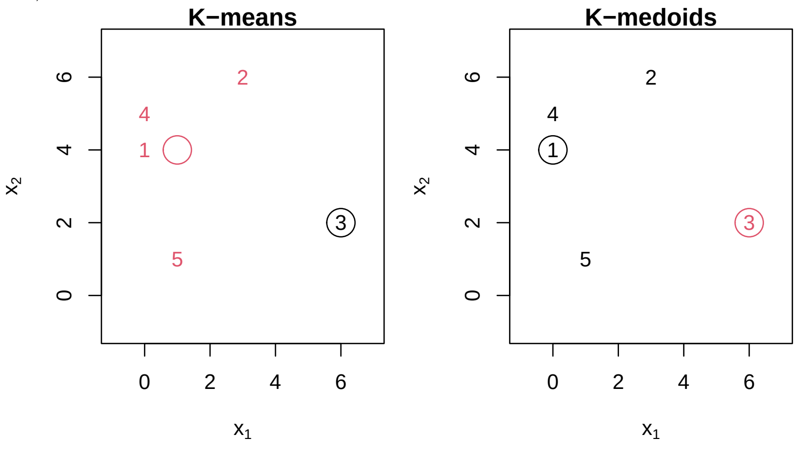
```
K<-2; kmeans(x=X,centers=K)->KM; ## clustering around the mean
KM$cluster ## the clusters obtained

## [1] 2 2 1 2 2
```

The given output is the set of labels for the clusters, that is, observations 1,2,4,5 form one cluster, and observation 3 forms another. Labels in cluster numbering are not important per se, i.e. output labels `2 2 1 2 2` are equivalent to `1 1 2 1 1`.

The following is the K-medoids clustering for the same data. The results coincide with those of the K-means.

```
library(cluster)
K<-2; pam(x=X,k=K)->PM; ## clustering around the medoid
PM$clustering ## the clusters obtained

## [1] 1 1 2 1 1
```

We plot the clusters next; numbers with the same color belong to the same cluster, and the circle indicates the centroid used in each case.



3.3.2 An explanation using exhaustive approach

To give some intuition on how cluster would be done exhaustively, partitioning $n = 5$ observations in $K = 2$ clusters would lead to the following computations. There are $\binom{5}{2} = \binom{5}{3} = 10$ different ways of rearranging labels `1 1 1 2 2`; and there are $\binom{5}{1} = \binom{5}{4} = 5$ ways of rearranging labels `1 1 1 1 2` so in total there can be 15 potential clusters.

The following table summarizes the whole search. From the table, we select for each method the best clustering as that scheme that minimizes ESS. This coincides with the software results.

Clusters	ESS	
	K-medoid	K-mean
123, 45	55	34.5
124, 35	37	21
134, 25	70	43.167
15, 234	45	31.667
14, 235	52	27.167
135, 24	46	30.333
125, 34	68	39.833
145, 23	36	21.833
13, 245	67	38.667
12, 345	56	35.833
1, 2345	64	38
1345, 2	51	34.75
1245, 3	**24**	**20**
1235, 4	63	35.75
1234, 5	48	33.5

Exhaustive search is very expensive indeed. Even in this simple scheme with $K = 2$ clusters, the number of possible clusters grows exponentially as function of n. For this reason, this exhaustive development is intended as a guide to explain, but not as a tool used in practice or in our Module. The relevant `R` functions such as `kmeans`, `pam` or `clara` do a numerical search in an otherwise very expensive problem.

3.3.3 Plotting ESS and the silhouette to suggest K

In practice, different values of K are tried to then select a potential number of clusters. If we consider the measure ESS as a measure of cluster effectiveness, it is useful to remember that as the number of clusters grows, the quantity ESS diminishes so that when every unit is its own cluster, ESS is effectively zero. Hence minimizing ESS cannot be taken as an objective as trivially the minimum value ESS=0 is achieved regardless of data when $K = n$. A plot of ESS can be used to select potential number of clusters by looking at a decrease in ESS while keeping K at the smallest possible.

The **silhouette** $s(i)$ is a numerical summary that suggests how good clustering has been done for individual i. Values of $s(i)$ close to **one** indicate that the individual i has been correctly clustered, while a **zero** value of $s(i)$ suggests that the individual lies between two clusters. A **negative** value of the silhouette suggests that the individual is in the wrong cluster. Usually, the average silhouette value per cluster is reported, or the average silhouette over all clusters.

In a practical application, the clusters obtained have to be interpreted.

3.3.4 Example of K-means clustering

Consider the dataset `USArrests`, previously seen. Before clustering, we compute and analyze Principal Components.

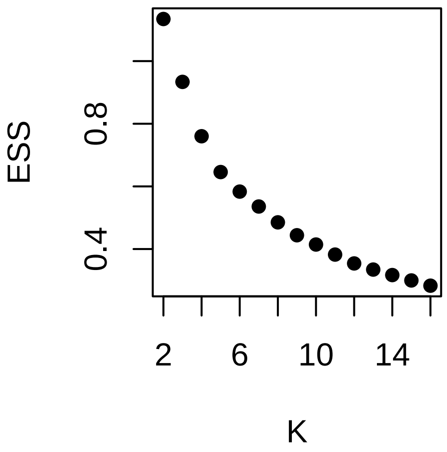
```
X<-scale(x=USArrests,center=TRUE,scale=TRUE);
PC<-prcomp(x=X,scale=FALSE,center=FALSE)
summary(PC)

## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.5749 0.9949 0.59713 0.41645
## Proportion of Variance 0.6201 0.2474 0.08914 0.04336
## Cumulative Proportion  0.6201 0.8675 0.95664 1.00000
```

We need only two components to recover more than 80% of the total variability. The PC scores are the values that will be clustered. We redefine the data to be these.

```
X<-PC$x[,1:2]
```

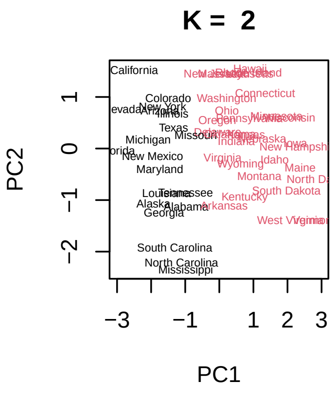
We compute ESS values for K between 1 and 15 and plot them. The ESS plot suggests no more than perhaps $K = 4$ or $K = 5$ clusters.

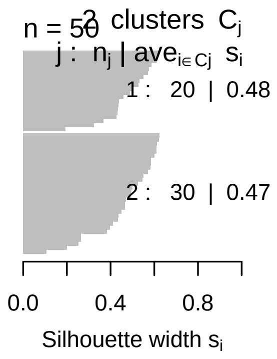
```r
for(K in 1:15) ess[K]<-pam(x=X,k=K+1)$objective[2]
plot(x=(1:15)+1,y=ess,xlab="K",ylab="ESS",pch=16,log="")
```

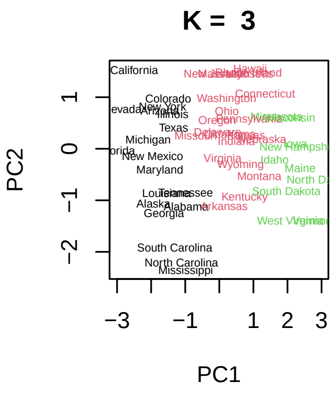


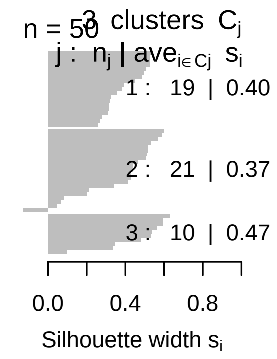
We now explore potential values $K = 2, 3, 4, 5$ for clustering. We plot the data with colors indicating clusters and in each case we do the silhouette plots.

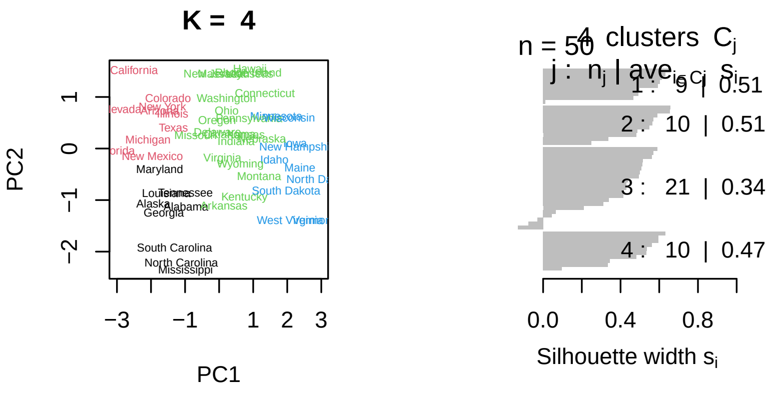
```r
par(mar=c(4,4,3,3),mfrow=c(2,2))
for(K in 2:5){
  clus<-pam(x=X,k=K); si <- silhouette(clus)
  plot(X[,1],X[,2],type="n",xlab="PC1",ylab="PC2",main=paste("K = ",K))
  text(x=X[,1],y=X[,2],labels=rownames(X),cex=0.5,col=clus$clustering)
  plot(si,main="")
}
```

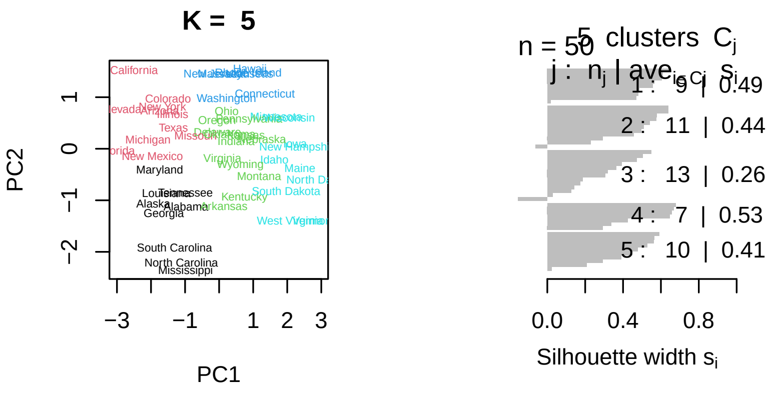

K = 2
PC2
PC1
−3
−1
1
2
3
1
0
−1
−2
California
Colorado
Washington
Connecticut
Ohio
Texas
Michigan
New Mexico
Maryland
Virginia
Wyoming
Idaho
Maine
Montana
Kentucky
South Dakota
Alaska
Alabama
Georgia
Arkansas
West Virginia
South Carolina
North Carolina
Mississippi


n = 50
2 clusters C_j
j : n_j | $ave_{i \in Cj}$ s_i
1 : 20 | 0.48
2 : 30 | 0.47
0.0
0.4
0.8
Silhouette width s_i


K = 3
PC2
PC1
−3
−1
1
2
3
1
0
−1
−2
California
Colorado
Washington
Connecticut
Ohio
Texas
Michigan
New Mexico
Maryland
Virginia
Wyoming
Idaho
Maine
Montana
Kentucky
South Dakota
Alaska
Alabama
Georgia
Arkansas
West Virginia
South Carolina
North Carolina
Mississippi

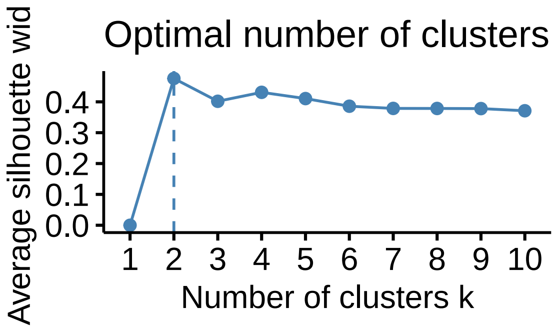

n = 50
3 clusters C_j
j : n_j | $ave_{i \in Cj}$ s_i
1 : 19 | 0.40
2 : 21 | 0.37
3 : 10 | 0.47
0.0
0.4
0.8
Silhouette width s_i


K = 4
PC1
PC2
California
Colorado
New York
Illinois
Texas
Michigan
New Mexico
Maryland
Washington
Connecticut
Ohio
Oregon
Pennsylvania
Minnesota
Wisconsin
Delaware
Missouri
Indiana
Nebraska
Iowa
New Hampshire
Virginia
Wyoming
Idaho
Maine
Montana
North Dakota
Tennessee
Louisiana
Alaska
Alabama
Georgia
Kentucky
Arkansas
South Dakota
West Virginia
Hawaii
South Carolina
North Carolina
Mississippi
n = 50
4 clusters C_j
$j : n_j | \text{ave}_{i \in C_j} s_i$
1 : 9 | 0.51
2 : 10 | 0.51
3 : 21 | 0.34
4 : 10 | 0.47
Silhouette width s_i

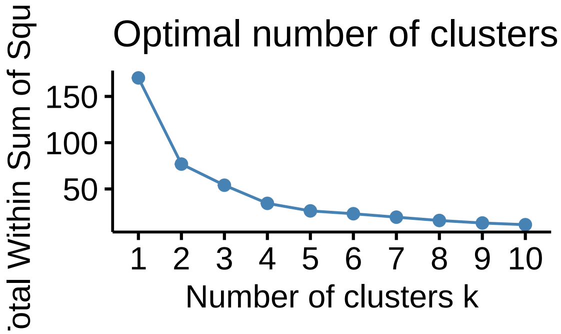

K = 5
PC1
PC2
California
Colorado
New York
Illinois
Texas
Michigan
New Mexico
Maryland
Washington
Connecticut
Ohio
Oregon
Pennsylvania
Minnesota
Wisconsin
Delaware
Missouri
Indiana
Nebraska
Iowa
New Hampshire
Virginia
Wyoming
Idaho
Maine
Montana
North Dakota
Tennessee
Louisiana
Alaska
Alabama
Georgia
Kentucky
Arkansas
South Dakota
West Virginia
Hawaii
South Carolina
North Carolina
Mississippi
n = 50
5 clusters C_j
$j : n_j | \text{ave}_{i \in C_j} s_i$
1 : 9 | 0.49
2 : 11 | 0.44
3 : 13 | 0.26
4 : 7 | 0.53
5 : 10 | 0.41
Silhouette width s_i

A simple summary is the average silhouette width. A direct computation gives $(20 * 0.48 + 30 * 0.47)/50 = 0.474$ for the first clustering with $K = 2$. The following is function from library `factoextra` does that computation for several values of K and produces a plot to help picking a value of K. Note the low average $s(i)$ even for the better case.

```r
library(factoextra); par(mar=c(4,4,1,1))
fviz_nbclust(x=X,FUNcluster=pam, method = "silhouette")
```



```r
fviz_nbclust(x=X,FUNcluster=pam, method = "wss")
```



We also plotted above the sum of squares with the same function. This latter plot offers the same insight as the previous plot of ESS, but the specific values differ due to scaling done with `pam`.

Exercise 23 Use the same data of Example 20 and following to practice with the functions `kmeans` and `pam`. In your selected case, explore different values of K and conclude suggesting a number of clusters.

Exercise 24 Explore the function `pamk` from the library `fpc`. This function creates a list of values of K together with values of a selected criterion. This function should be compared against the function `fviz_nbclust` from the library `factoextra` used above.

Exercise 25 Consider again the `USArrests` data seen in notes. What happens to clustering if PCA analysis is done without centering and scaling? Redo analysis, compare and explain.

Exercise 26 This question is about the Error Sum of Squares (ESS). On a first glance, it looks like the Sum of Squares of the Error (SSE) of ANOVA and/or regression that you have seen before. Are they the same? Write a small paragraph describing the similarities and differences between ESS and SSE.

Important points and concepts of week six:

1. Become familiar with the K-means algorithm in its variants: K-means and K-medoids.

2. Use the functions `pam` from library `cluster` and `kmeans` to do K-means clustering of data. Suggest a number of clusters and interpret them.

3. In order to suggest a number of clusters, become familiar with scatterplots and auxiliary plots of ESS and the silhouette.

Week seven

Supervised learning

In supervised learning, we have a **response** (**output** or **target**), together with information on p **variables** (**predictors** or **features**). The aim is to predict correctly these outputs or targets.

The data available looks like this $(\mathbf{X}|\mathbf{Y})$, where $\mathbf{X}$ is a matrix of variables with the usual size $n \times p$. These values could be discrete or continuous. The matrix $\mathbf{Y}$ is a column vector of size $n \times 1$ with response values.

In the following development, note the important distinction of techniques depending on the nature of the levels of the response vector $\mathbf{Y}$:

In classification problems, the response $\mathbf{Y}$ takes values from a discrete set. The simplest case has only 0/1 values. We will only see the 0/1 case in this Module although be aware that by no means it is the only case to appear in practice.

In regression, the response $\mathbf{Y}$ takes values from a potentially infinite set, commonly real values.

For most possible domains of interest, the number of potentially useful data cases that we cannot observe is very large and thus it is not possible to establish predictive accuracy beyond dispute. Common practice is to estimate predictive accuracy of an algorithm by measuring its accuracy for a data set that has not been used. Hence, we commonly

1. **train** the algorithm with part of the data to then

2. **validate** it using a separate instance of the data

Figure 3 illustrates the approach to splitting the data. For the process of data splitting, sometimes the training and testing data sets are partitioned to be in separate files. More commonly, a partition is needed before doing the analysis. Usually this is a **random** partition and several options for the proportions split are available: 1:1, 2:1, 70:30, 60:40 or other proportions.

There are three main strategies for the data split:

1. divide the data into a single training and a single test sets. This process is shown in Figure 3. Usually the split is done at random.

2. k-fold cross-validation. The data is split into k folds and use all the folds but one to train and the remaining fold to test. This is done k times until each folds has been used to test. The folds are created usually at random, and

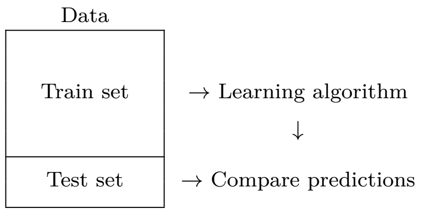


Figure 3: Partition of the data for training and testing.

the sizes of the folds are as even as possible. A popular choice is the 10-fold cross-validation.

3. n-fold cross-validation. This is also known as leave-one-out cross-validation. In each of n instances, one observation is left out of the training data, to be subsequently used for testing.

An important point of supervised learning is to make results reproducible and thus **setting a random seed** is a standard part of the analysis. Reproducibility is achieved using a command such as the `R` command `set.seed`, before randomly splitting the data.

In cross-validation, after fitting each of the k models and obtaining predictions, the usual Machine Learning approach is to compute the prediction error. That is, use the trained model over the test data in each of the instances to compute the error. These errors would be averaged over the different instances tried. The aim is to compare different modeling methods using with the same dataset and partitions instances of it.

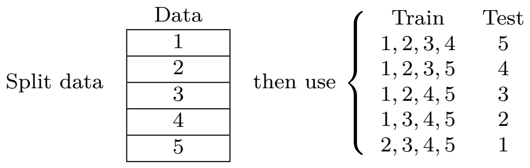


Figure 4: Example of 5-fold partition of the data for training and testing.

As an example of the k-fold approach, consider the 5-fold cross-validation. Data would be partitioned into $k = 5$ parts and then different instances of this partition be used to train the model and to test. There would be $k = 5$ errors, one for each instance which would be averaged. Figure 4 illustrates this.

Machine Learning and Statistics

We have presented the general approach to supervised learning. Here we briefly comment on both Machine Learning and Statistics approaches to analysing data.

In Machine Learning there is emphasis in comparing models using prediction capabilities. The standard approach is to first fit the model using part of the data and then use fresh data to compute the prediction error for comparing models.

The Machine Learning approach is in contrast with the standard modelling approach in statistics. In statistics, models are usually fitted and then terms selected using significance tests. Usually, models are not compared on the basis of their predictive capabilities.

4 Classification methods

The problem of classification arises when the output takes values from a discrete set. These values are commonly referred to as labels and the interest lies in correctly predicting these labels. The case we will consider in our Module is the simplest when there are only two labels, although methodology has been developed for other numbers of labels.

The two labels we will use will be referred to as 0/1. These 0/1 levels are coded values and should be understood as a renaming of what the true values would be in a study, that is the two levels could also be 'No'/'Yes'; 'negative'/'positive' or $-/+$. For example, in recognition of lung disease from scanned x-ray images, a 'positive' (coded as 1) would be a detection of a diseased lung, and a 'negative' (coded as 0) would be the non-detection of irregularity in the lung.

4.1 Performance of methods - confusion matrix

Using available data and the partition strategy defined earlier, the machine learning task of classification involves comparison of classifiers. In what follows, we will describe the standard framework for comparing classifiers of 0/1 data.

To motivate our development, consider a case where we have split the data into training and testing sets. We have available the data with $n = 8$ observations. We have true and predicted data, and the predicted data comes from **two** models Y1 and Y2. Using data, each of the models Y1 and Y2 is going to be **compared** (benchmarked) against the true values available.

Ytrue	Y1	Y2
1	1	1
0	1	1
1	1	0
0	0	1
0	0	1
1	0	0
0	0	1
0	1	1

Consider the above data and the **first** classifier Y1. The first row is an example of true positive as it has been classified as positive and (Y1=1) indeed it was positive (Ytrue=1). The last row is a case of a false positive as it has been classified as a positive (Y1=1) yet it was a negative (Ytrue=0).

For the **second** classifier Y2, the same conclusions apply to the first and last observations. In addition, for this classifier Y2, the third row is an example of a false negative as it has been classified as a negative (Y2=0) yet it was in reality a positive (Ytrue=1).

Define the following cases based upon values of the true response (true classification) and the predicted response (predicted classification):

Definition		Y (true)	Y (predicted)
True positive	is a positive classified as a positive	1	1
False positive	is a negative classified as a positive	0	1
False negative	is a positive classified as a negative	1	0
True negative	is a negative classified as a negative	0	0

For a given data set and a classifier, we count instances of each case and summarize the information in order to be able to compute performance measures of a classifier. The **confusion matrix** summarizes the totals of each case observed in the data in a 2×2 array.

		Predicted class		
		0	1	
True class	0	TN	FP	N
	1	FN	TP	P

The quantities in the cells of the confusion matrix are identified by acronyms which are self explanatory: TP is the number of true positives; FP is the number of false positives; FN is the number of false negatives and TN is the number of true negatives; also P is the total number of positives and N is the number of negatives. We have that N=TN+FP and P=TP+FN.

The confusion matrix can be written with rows and columns permuted as follows. We can work with **either version** as they are equivalent, however we have a slight preference for the **first version above**. Here is the second version.

		Predicted class		
		1	0	
True class	1	TP	FN	P
	0	FP	TN	N

Continuing with the data given earlier, we compute the confusion matrices. The matrices are generated by counting occurences (frequencies) of the different "true-predicted" cases "00", "01", "10", "11" for every prediction given. The values of N and P are common in each table, taken directly from true data N= 5 and P= 3.

		Predicted Y1	
		0	1
True	0	3	2
	1	1	2

		Predicted Y2	
		0	1
True	0	0	5
	1	2	1

4.2 Performance of methods - measures

The next step is to compare methods. Following the example with data and tables given above, the classifier Y1 appears to be best as it has correctly classified 5 observations, while Y2 has only classified correctly 1 observation. To properly quantify this, there are a number of proposals from the literature to measure the performance of classifiers. The next table contains some of the most common performance measures.

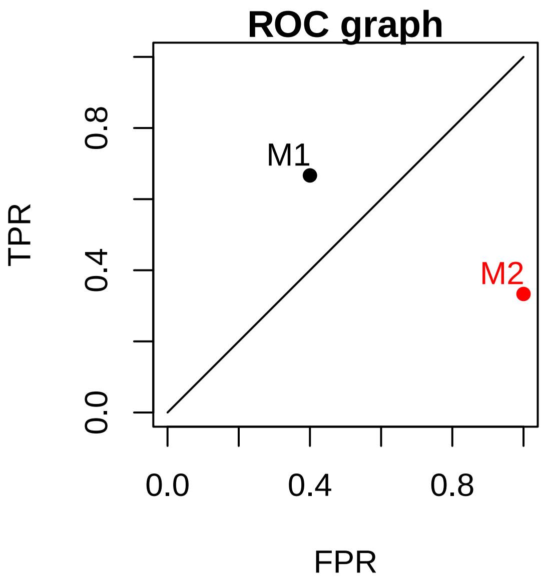
Name	Formula	Description
True Positive Rate	$\mathbf{TPR} = \frac{\mathrm{TP}}{\mathrm{P}}$	Proportion of positives classified as positive (aka Hit rate, recall or **Sensitivity**)
False Positive Rate	$\mathbf{FPR} = \frac{\mathrm{FP}}{\mathrm{N}}$	Proportion of negative instances wrongly classified as positive
False Negative Rate	$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}}$	Proportion of positive instances wrongly classified as negative: $1 - \mathrm{TP}/\mathrm{P}$
True Negative Rate	$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}}$	Proportion of true negatives correctly classified as negative (also known as **Specificity**): $1 - \frac{\mathrm{FP}}{\mathrm{N}}$
Precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$	Proportion of cases classified as positive that are really positive (aka positive predictive value)
Negative predictive value	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}}$	Proportion of cases classified as negative that are really negative
F1 score	$2\frac{\mathrm{Precision}\cdot\mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}}$	Combination of precision and recall
Accuracy	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}}$	Proportion of instances classified correctly
Error rate	$\frac{\mathrm{FP}+\mathrm{FN}}{\mathrm{P}+\mathrm{N}}$	Proportion of instances which are incorrectly classified

The first two measures **TPR** and **FPR** are used to compare classifiers by plotting TPR vs FPR in a plot known as the **Receiver Operating Characteristic graph (ROC graph)**.

For the data of our running example we have with two classifiers, the following table gives the measures TPR and FPR in each case.

Classifier	TPR	FPR
M1 (Predicted Y1)	0.666667	0.4
M2 (Predicted Y2)	0.333333	1

The result measures are shown in the ROC graph below.



Clearly M1 is a much better classifier than M2 having simultaneously better (higher) True Positive Rate TPR and also better (lower) False Positive Rate FPR.

There are **five** regions in the ROC graph which correspond to different patterns of classifiers:

1. the top left corner corresponds to **good** classifiers as they have low values of FPR and high values of TPR.

2. on the opposite corner, the bottom right is that of **badly performing** classifiers with high error rates of FPR and low success rates of TPR.

3. The top right corner is that of **liberal** classifiers which tend to give high rates of true positives (TPR) but in doing so, the rate of false positives is also high (FPR).

4. The bottom left corner is that of **conservative** classifiers which mostly give negative classification and as such it has low success rate for positives (TPR) but also has low error rate FPR.

5. Finally, the central area corresponds to classifiers whose pattern of classification is like a **random** classifier.

The line FPR=TPR is added to the ROC graph to help separate those classifiers with good classification rates from those that classify otherwise.

Exercise 27 Repeat the classification exercise for each of the following result sets. In every case this implies to build the confusion matrix, compute TPR and FPR, plot results in the ROC graph and interpret results, comparing classifiers.

1.

Ytrue	0	1	0	0	1	0	1	1	1	0
Y1	1	1	0	1	0	0	1	0	1	1
Y2	1	1	1	0	1	1	1	1	1	1
Y3	0	0	0	0	0	0	1	0	0	1

2.

Ytrue	1	0	0	1	0	0	0	1	0	1
Y1	0	1	1	1	0	1	0	0	0	1
Y2	1	1	1	0	1	1	1	0	1	1
Y3	0	0	0	1	0	0	0	1	0	0

3.

Ytrue	1	0	0	1	0	0	0	1	0	1	1	0	0	0	1
Y1	0	0	1	1	0	0	0	1	1	0	1	0	1	0	1
Y2	1	1	1	0	1	1	1	0	1	1	0	0	1	1	0
Y3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y4	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1

4.

Ytrue	0	1	1	1	0	0	1	0	1	0	0	1	1	0	0
Y1	1	1	1	0	0	0	0	0	0	0	1	0	1	1	0
Y2	1	1	1	1	1	1	1	1	0	0	1	0	1	0	0
Y3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y4	0	1	0	1	0	0	1	0	1	0	0	0	1	0	0

Exercise 28 Consider n observations for testing a 0/1 classification model so that the data available are true observations (labels) $Y_{t1}, Y_{t2}, \dots, Y_{tn}$ (the subindex t in Y_t stands for 'true' data) and predicted observations (labels) $Y_1, Y_2, \dots, Y_n$.

1. Explain the rationale behind each of the following identities:

$$\mathrm{TP} = \sum_{i=1}^{n} Y_{ti} Y_i, \quad \mathrm{TN} = \sum_{i=1}^{n} (1 - Y_{ti})(1 - Y_i), \quad \mathrm{FP} = \sum_{i=1}^{n} (1 - Y_{ti}) Y_i,$$
$$\mathrm{FN} = \sum_{i=1}^{n} Y_{ti}(1 - Y_i), \quad \mathrm{N} = \sum_{i=1}^{n} (1 - Y_{ti}), \quad \text{and} \quad \mathrm{P} = \sum_{i=1}^{n} Y_{ti}.$$

   As help, note that being 0/1 classification data, each of the observed Y_{ti} and predicted Y_i are integers that can only take either zero value or one for all $i = 1, 2, \dots, n$. Also that n and N are different integer quantities, only related in the inequality $0 \leqslant \mathrm{N} \leqslant n$.

2. Prove the following result.

   Theorem Let MSE be mean squared error of validation, defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_{ti} - Y_i)^2 .$$

   Then MSE equals the error rate, that is

$$\text{MSE} = \frac{\text{FP+FN}}{\text{P} + \text{N}}.$$

Exercise 29 For each of the data cases given in Exercise 27 show numerically (by hand or using `R`) that the identities above hold.

Exercise 30 Ditto numerical verification for the Theorem above and data from Exercise 27.

Exercise 31 This exercise is concerned with further identities.

1. The error rate is a linear combination of False Positive Rate and the False Negative Rate. Find the coefficients, that is determine constants ω_1, ω_2 such that

$$\text{Error rate} = \omega_1 \cdot \text{FPR} + \omega_2 \cdot \text{FNR}.$$

2. Ditto concerning Accuracy as linear combination of Sensitivity and Specificity, that is find constants ω_1, ω_2 such that

$$\text{Accuracy} = \omega_1 \cdot \text{Sensitivity} + \omega_2 \cdot \text{Specificity}.$$

3. Show that

$$\text{Error rate} = 1 - \text{Accuracy}.$$

Important points and concepts of week seven:

1. Become familiar with the concept of **splitting the data** in training and testing for evaluation of a model. Ditto the concept of k-fold cross-validation and leave-one-out cross-validation.

2. The concept of classification with binary response (positives, negatives) and the mistakes (misclassification) that may occur.

3. Definition of the **confusion matrix** and computations with it for a given data set, including the figures TN, FP, FN, TP, P, N.

4. The summary measures **TPR** and **FPR**, aka as sensitivity and (1-)specificity.

5. Computations of summary measures with a given data set and plotting them and interpreting in the **ROC graph**.

Week eight

4.3 The ROC curve and AUC

The ROC graph is used to compare classifiers. Each classifier is represented with a point in the graph, i.e. a point in the coordinates (FPR, TPR). The point (FPR, TPR) $= (0, 1)$ corresponds to an ideal classifier so that the closer the classifier is to this top left corner, the closer it is to an ideal classifier.

There are classification methods that allow for some tuning through some parameters. In such cases, the plot of points in the ROC graph depends on these tuning parameters and the resulting plot is known as the ROC curve.

To motivate our development, consider the data set below with $n = 12$ observations. One explanatory variable x is available and the responses y take the usual labels 0/1. For modeling purposes, the data is split in two instances (50 : 50) for training and testing a classifier (model).

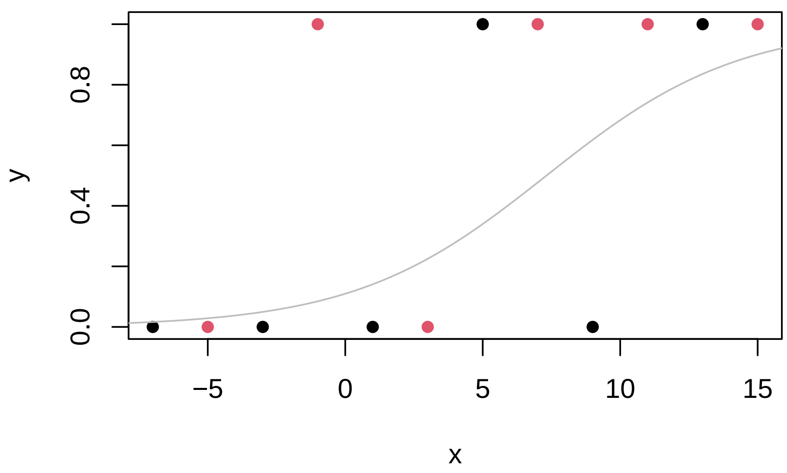
Training data		Test data	
x	y	x	y
1	0	3	0
-7	0	15	1
-3	0	-1	1
5	1	-5	0
13	1	11	1
9	0	7	1

Logistic regression (**seen later in this Chapter**) will be the classifier used to predict labels, and the proposed classifier (model) is the **standard logistic model with linear trend** $\beta_0 + \beta_1 x$.

Using the training data, the estimated parameters were $\hat{\beta}_0 = -2.0948$ and $\hat{\beta}_1 = 0.2862$. For a given value x, the estimated probability $\hat{\Pr}(Y = 1|x)$ is computed using the standard formula inverted log-odds and estimated coefficients so that

$$\hat{\Pr}(Y = 1|x) = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 x)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 x)} = \frac{\exp(-2.0948 + 0.2862x)}{1 + \exp(-2.0948 + 0.2862x)}.$$

The following plot has the data (training data black dots, test data red dots) together with the line of estimated probabilities.



In order to study the performance of this logistic model, we need to compute the estimated probabilities for the test data set. For $x = 3$, evaluate

$$\frac{e^{-2.0948+0.2862\cdot 3}}{1+e^{-2.0948+0.2862\cdot 3}} = \frac{e^{-1.2362}}{1+e^{-1.2362}} = 0.2251.$$

For $x = 15$, evaluate

$$\frac{e^{-2.0948+0.2862\cdot 15}}{1+e^{-2.0948+0.2862\cdot 15}} = \frac{e^{2.1982}}{1+e^{2.1982}} = 0.9.$$

For $x = -1$, evaluate

$$\frac{e^{-2.0948+0.2862\cdot -1}}{1+e^{-2.0948+0.2862\cdot -1}} = \frac{e^{-2.381}}{1+e^{-2.381}} = 0.0846.$$

For $x = -5$, evaluate

$$\frac{e^{-2.0948+0.2862\cdot -5}}{1+e^{-2.0948+0.2862\cdot -5}} = \frac{e^{-3.5258}}{1+e^{-3.5258}} = 0.0286.$$

For $x = 11$, evaluate

$$\frac{e^{-2.0948+0.2862\cdot 11}}{1+e^{-2.0948+0.2862\cdot 11}} = \frac{e^{1.0534}}{1+e^{1.0534}} = 0.7414.$$

For $x = 7$, evaluate

$$\frac{e^{-2.0948+0.2862\cdot 7}}{1+e^{-2.0948+0.2862\cdot 7}} = \frac{e^{-0.0914}}{1+e^{-0.0914}} = 0.4771.$$

These estimated probabilities are shown in the left hand side in the table below. To build points in the ROC curve, each probability has to be converted to 0/1 values. This is achieved by fixing a threshold. Probabilities below the fixed threshold are turned into a zero, and probabilities higher that this threshold are turned into one. For each threshold we compute the confusion matrix and summary measures FPR, TPR. If we repeat this computation as we vary the threshold in the interval $[0, 1]$, we retrieve the **ROC curve**.

Test data and probabilities

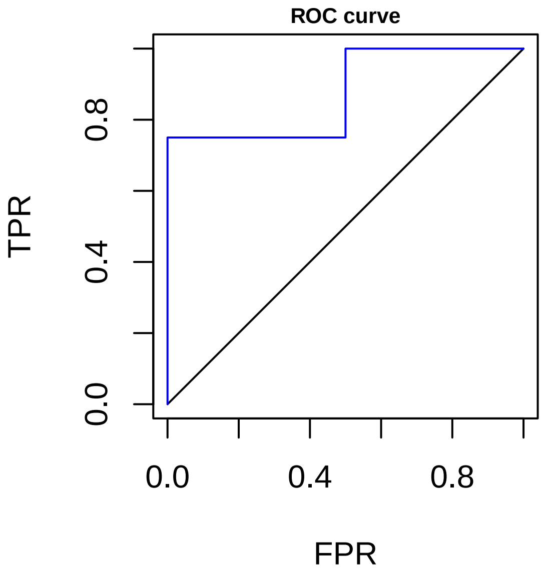
x_i	y_i	$\Pr(Y_i = 1 \mid x_i)$
3	0	0.2251
15	1	0.9
-1	1	0.0846
-5	0	0.0286
11	1	0.7414
7	1	0.4771

Threshold and classification

	0	0.05	0.2	0.3	1
	1(FP)	1(FP)	1(FP)	0(TN)	0
	1(TP)	1(TP)	1(TP)	1(TP)	0
	1(TP)	1(TP)	0(FN)	0(FN)	0
	1(FP)	0(TN)	0(TN)	0(TN)	0
	1(TP)	1(TP)	1(TP)	1(TP)	0
	1(TP)	1(TP)	1(TP)	1(TP)	0
FPR	1	0.5	0.5	0	0
TPR	1	1	0.75	0.75	0

To show how this works, this process is shown above for the values of threshold $0, 0.05, 0.2, 0.3, 1$. For each case the measures FPR and TPR are computed and given in the right hand side of the table.

For this data and model, the ROC curve is given next. In the plot, the line FPR=TPR is added as it represents the behaviour of classifier that is equivalent to classifying at random.

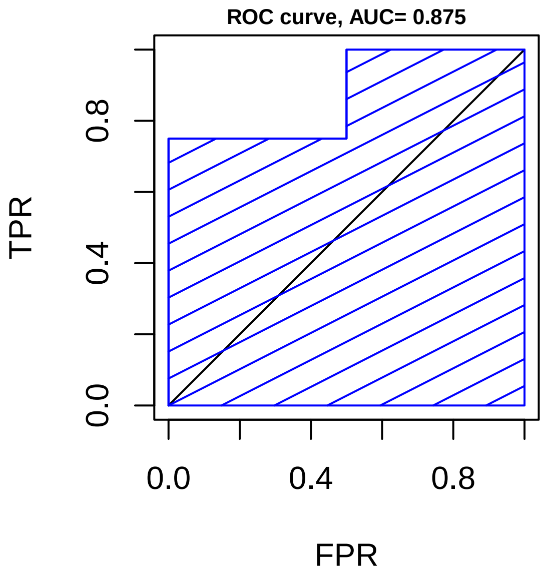


The ROC curve shows the different patterns of classification by a classifier, when moving the threshold. As this is computed using validation data, the changes in (FPR, TPR) are given in fractions of P and N, the numbers of positives and negatives in the validation data. Hence the ROC has jumps of multiples of 1/N in the horizontal axis (FPR) and of 1/P in the vertical axis (TPR). The ROC of a good classifier passes very close to the top left corner of the diagram.

The **area under the ROC curve** (**AUC**) is computed over all values of the threshold and it is the area under the ROC curve for values of Sensitivity (TPR) and Specificity (1-FPR) between zero and one. The AUC is also known as the *c-statistic* and is a useful quantity to compare overall performance of a classifier under a variety of thresholds. Furthermore, the statistic AUC is closely related to other statistics for cathegoric data tests such as the Mann-Whitney and the Wilcoxon statistic.

If the AUC is close to one, this signifies a very good classifier. If the AUC is close to 0.5, this is what we would expect from a classifier that is no better than classifying at random. A value of AUC lower than 0.5 is a sign of very poor classification.

For our initial small example, the AUC is 0.875 so this is a good classificator. The plot below has the AUC in shading. When dealing with real data though, it is rare to have a value of AUC bigger than 0.7 or 0.8.



Models for classification

In what follows we describe some classifiers. The emphasis in classification is in the **comparison** between classifiers rather than on the details of the models.

4.4 Linear classifier

The simplest way to model a 0/1 response is using a linear regression model with expectation $\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$. Using the fitted coefficients $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ and the test data, we could build predicted labels $\hat{y}$. A simple rule would then predict a label on one is $\hat{y} > 0.5$ and zero otherwise.

This is a simple and intuitive regression approach to classification and it can be shown that the estimate thus obtained $\hat{y}$ is in fact an estimate of $\Pr(Y = 1|\mathbf{x})$. However note that this model may not be entirely adequate as some of the label estimates $\hat{y}$ may be outside the interval $[0, 1]$. Another potential drawback is that this model cannot be easily adapted to classification problems with more than two labels.

Despite these drawbacks, linear regression is simple and widely known. In particular, note that this statistical method is relatively more robust than e.g. logistic classifier and it may work in cases the logistic doesn't. The linear classifier is implemented in `R` with the `lm` function.

4.5 Logistic classifier

Logistic regression is an established statistical technology that belongs to a general family of models called **generalized linear models** (GLM). The GLM family of models includes amongst many, standard linear regression, regression for count data and regression for binary data, also known as logistic regression.

A main drawback of the linear classifier was that probability estimates could lie outside $[0, 1]$. To avoid this drawback, the probability $p(\mathbf{x}) = \Pr(Y = 1|\mathbf{x})$ should be modelled by a function that guarantees that probabilities lie in $[0, 1]$ for any value of $\mathbf{x}$. The logistic model is achieved when using the Bernoulli trials to describe data, and a special relation between covariates and Bernoulli probabilities that guarantees the latter property.

The data y_i are assumed to be independent realizations of a Bernoulli random variable with probabilities that obbey the logistic function

$$p(\mathbf{x}_i) = \frac{e^{\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}}}{1 + e^{\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}}}.$$

The inversion of the logistic function represents the log-odds as a linear function of the values of $\mathbf{x}_i$:

$$\log\left(\frac{p(\mathbf{x}_i)}{1-p(\mathbf{x}_i)}\right) = \beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}.$$

This is multiple logistic regression, meaning that the log-odds are modelled as a linear function of several variables. Estimation of the model coefficients in this model is achieved by the method of maximum likelihood. This topic is beyond the scope of this Module and we use its implementation as a method of estimation for the analysis of data in `R` in the function `glm` with option `family="binomial"`.

Once the parameter estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ are available, estimates of probabilities of the type $p(\mathbf{x})$ can be computed using e.g. a test observation $\mathbf{x}_i$ as follows

$$\hat{p}(\mathbf{x}_i) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \ldots + \hat{\beta}_p x_{pi}}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \ldots + \hat{\beta}_p x_{pi}}}.$$

Together with existing true test observations, these probability estimates are used to build summary measures that describe the performance of the classifier thus obtained and to compare the logistic classifier with other classifiers.

Importantly, note that in general, statistics for testing classifiers with fresh data (such as ROC and AUC) may not be informative about significance of variables in terms in traditional statistical analysis. In the case of logistic regression, good clasification performance may not relate for example to model deviance.

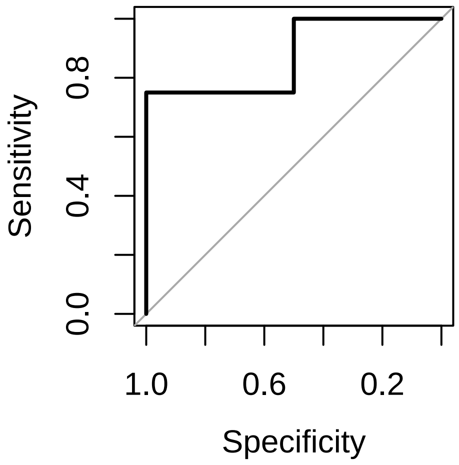
4.5.1 Logistic code for the small example

Code for training the logistic model and for building predictions using test data for the example first seen in Page 71.

```
DAT ## x,y values and FF identifies 1-training, 2-testing

##      x y FF
## 1    1 0  1
## 2   -7 0  1
## 3    3 0  2
## 4   15 1  2
## 5   -3 0  1
## 6    5 1  1
## 7   -1 1  2
## 8   -5 0  2
## 9   11 1  2
## 10  13 1  1
## 11   7 1  2
## 12   9 0  1

##
## Indexes for the training and test data sets
Train<-DAT$FF==1
Test<-DAT$FF==2
M1<-glm(y~x,data=DAT[Train,],family = "binomial")
M1$coefficients

## (Intercept)           x
##  -2.0948121   0.2861654

P1<-predict.glm(object = M1,newdata = DAT[Test,],type = "response")
P1 ## predicted probabilities

##          3          4          7          8          9         11
## 0.22507793 0.90004002 0.08463481 0.02859146 0.74135203 0.47710251
```

We next compute the ROC, plot the ROC curve and compute AUC.

```
library(pROC)
roc(response=DAT$y[Test],predictor=P1)->RR
RR$sensitivities ## this is TPR

## [1] 1.00 1.00 0.75 0.75 0.50 0.25 0.00

RR$specificities ## this is 1-FPR

## [1] 0.0 0.5 0.5 1.0 1.0 1.0 1.0

plot(RR)
```



```
auc(RR)

## Area under the curve: 0.875
```

In the plot, note the horizontal axis “Specificity”, defined as 1-FPR.

4.5.2 Comparison glm and lm for the small example

Here we compare the **logistic** results with the **linear** classifier. We do not expect very good performance from this classifier yet we include it as linear regression is a standard statistical technique.

```
M2<-lm(y~x,data=DAT[Train,]);
M2$coefficients

## (Intercept)           x
##  0.20476190  0.04285714

P2<-predict.lm(object = M2,newdata = DAT[Test,])
```

Here are the predicted responses. Note the negative value, which is **perfectly valid** for a linear regression but **inadequate for estimating a probability**.

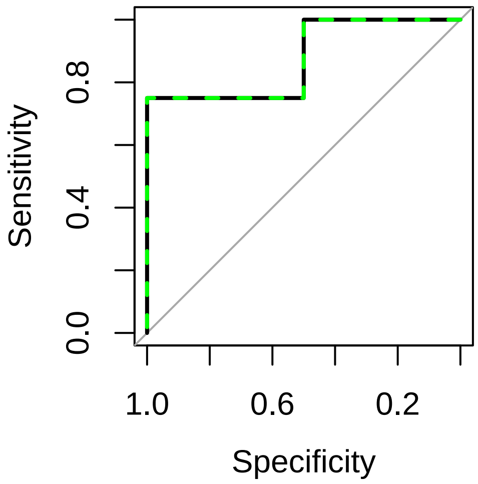
```
P2

##           3           4           7           8           9          11
##  0.33333333  0.84761905  0.16190476 -0.00952381  0.67619048  0.50476190
```

We compute the ROC and show sensitivities, specificities and AUC below.

```
roc(response=DAT$y[Test],predictor=P2)->RR2

## Setting levels:  control = 0, case = 1
## Setting direction:  controls < cases

RR2$sensitivities ## this is TPR

## [1] 1.00 1.00 0.75 0.75 0.50 0.25 0.00

RR2$specificities ## this is 1-FPR

## [1] 0.0 0.5 0.5 1.0 1.0 1.0 1.0

auc(RR2)

## Area under the curve: 0.875
```

Perhaps surprisingly, the values of TPR and FPR coincide with that of the logistic classifier, that is the two classifiers (linear and logistic) are equivalent. The following plot shows the two ROC curves together.

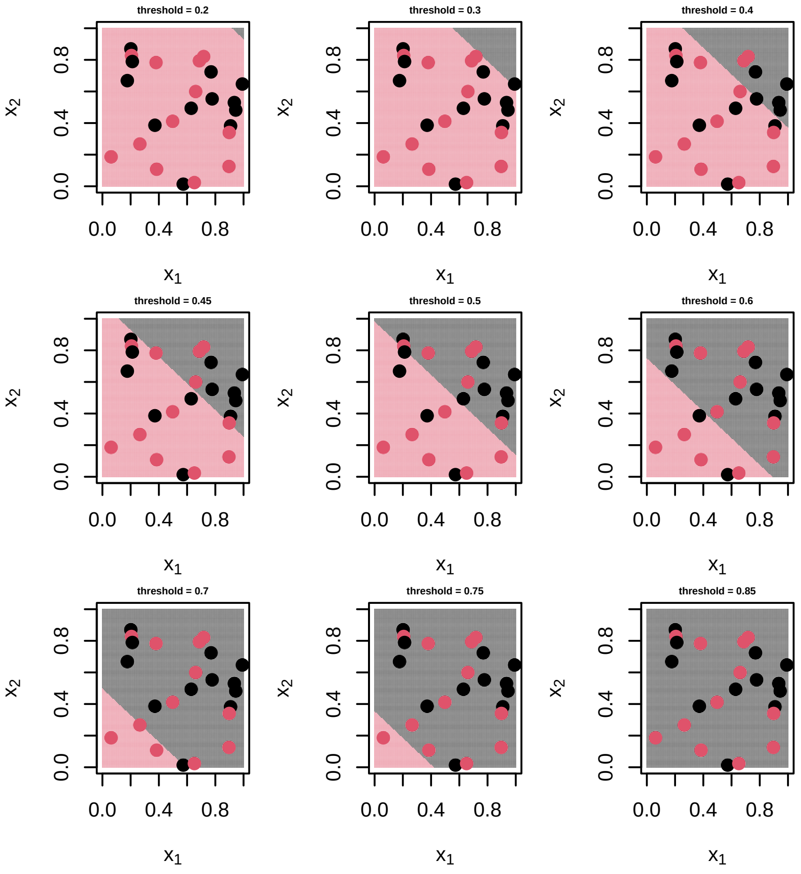
```
plot(RR) ## logistic
plot(RR2,add=TRUE,col="green",lty=2) ## linear
```



4.5.3 Synthetic example

The plots in the following page illustrate logistic classification for a synthetic example with $n = 24$ observations in $p = 2$ variables. The variables x_1, x_2 will be used to classify. The plot shows the regions in which the space is separated, according to the value of the threshold selected.

The region colored in black corresponds to predicted zeroes or “negatives”, while the red region is for ones or “positives”. The observations are added, using the same color coding. This allows for simple graphical depiction of false positives (black dot in red region), false negatives (red dot in black region). Dots in a region of their own color are either true positives (red) or true negatives (black).


threshold = 0.2
threshold = 0.3
threshold = 0.4
threshold = 0.45
threshold = 0.5
threshold = 0.6
threshold = 0.7
threshold = 0.75
threshold = 0.85
x_1
x_2
0.0
0.4
0.8

4.5.4 Logistic code: `Default` data

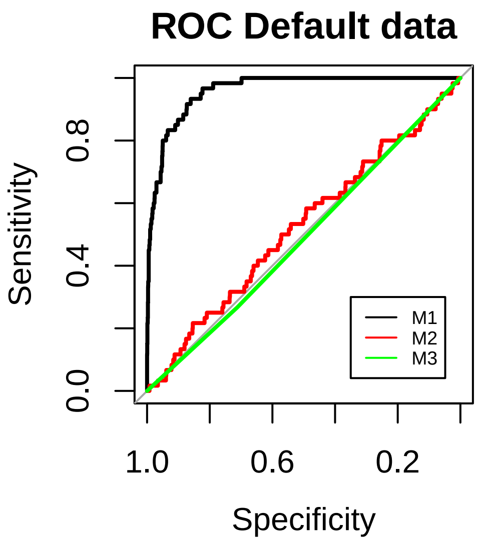
This code partly revisits the lab material for week eight with **`Default`** data. Recall that this dataset has default data (persons who could not pay debt) with continuous variables **`income`** and **`balance`** (how much money is owed) and categorical variables **`student`** and **`default`** (the response variable). To depart from the lab material, we consider simple predictive models for each of the three variables individually.

The analysis performed concentrates on the ROC curve and AUC statistic and no output is given for the classifier training stage nor for the stage for building test response (probabilities).

```
## select the fold
library(cvTools)
set.seed(0); n<-10000; K<-5; cvFolds(n=n,K=K)->CV
CV$subsets[CV$which!=5]->Train; CV$subsets[CV$which==5]->Test
## Train classifiers
library(ISLR)
attach(Default)
M1<-glm(default~balance,family = "binomial",data = Default[Train,])
M2<-glm(default~income,family = "binomial",data = Default[Train,])
M3<-glm(default~student,family = "binomial",data = Default[Train,])
## Build validation data
predict.glm(object = M1, newdata =Default[Test,],type="response" ) -> P1
predict.glm(object = M2, newdata =Default[Test,],type="response" ) -> P2
predict.glm(object = M3, newdata =Default[Test,],type="response" ) -> P3
## ROC computation
library(pROC)
roc(response=default[Test],predictor=P1)->RR1
roc(response=default[Test],predictor=P2)->RR2
roc(response=default[Test],predictor=P3)->RR3
```

In the plot below we show ROC curves for the three models. It is clear the overwhelming superiority of M1 over the models M2 and M3 which behave like random classifiers.

```
par(mar=c(4,4,1,1))
plot(RR1,col="black",main="ROC Default data") ## default~balance
plot(RR2,col="red",add=TRUE) ## default~income
plot(RR3,col="green",add=TRUE) ## default~student
legend(x=0.35,y=0.3,legend=c("M1","M2","M3"),lty=1,
  col=c("black","red","green"),cex=0.6)
```



```
auc(RR1); auc(RR2); auc(RR3)

## Area under the curve: 0.961
## Area under the curve: 0.5232
## Area under the curve: 0.4895
```

4.6 K nearest neighbors classifier (KNN)

In the K nearest neighbors classifier (KNN) a new observation is labeled according to the labels of the K nearest neighbors to it. In the case of our concern with two labels, the 0/1 label of a new observation is decided by the label of neighbors around it that has the highest frequency and in this sense, KNN is a form of neighbor voting. Because distance matters, if the variables do not have the same scales, this will impact the results of the KNN classifier and thus standardizing the data is recommended.

The KNN classification methodology does not require the usual two step of training and testing and it is all done in a single step. Also, in a KNN classifier, no parameter estimates are produced, and this feature is in contrast with other classifiers (e.g. linear or logistic), which produce interpretable parameter estimates and allow for significance testing of them.

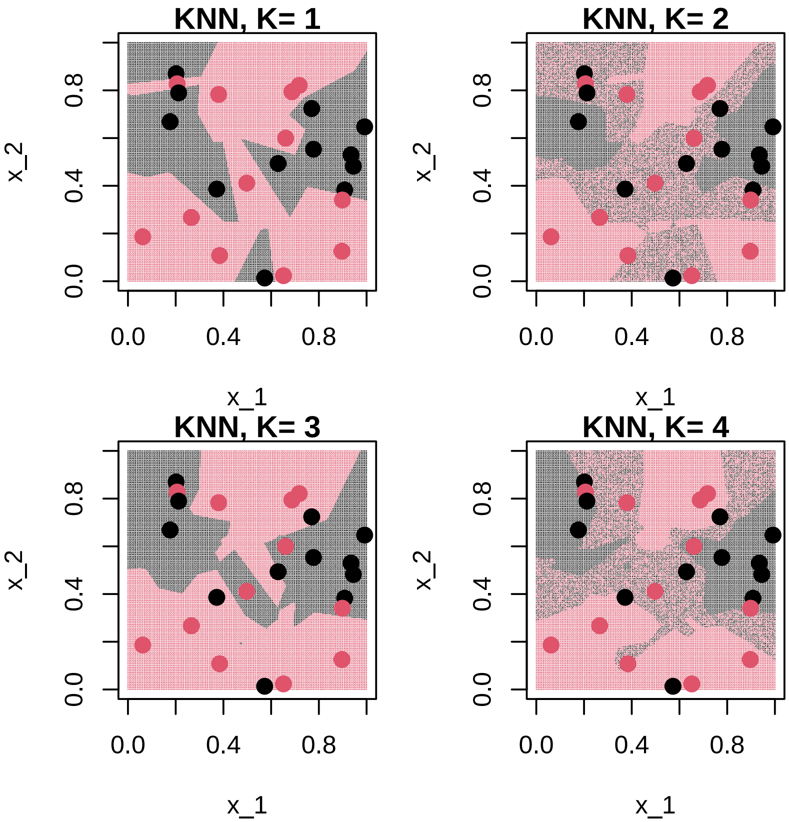
The chosen value of K is usually odd so that for two labels it will not be possible to have voting ties. When there is a tie among neighbor label frequencies, the label is chosen at random. This situation may happen when classifying more than two labels or in the 0/1 case, when K is even.

For our examples and analyses, we use the implementation of this classifier in the function `knn` from the library `class`.

4.6.1 Synthetic example

We apply the KNN classification to the same earlier synthetic example with $n = 24$ observations in $p = 2$ variables. For values of $K = 1, 2, 3, 4$, the plane is split into regions according to labels of the nearest neighbors.

The plots for each of said values of K are given in the next page. For odd values of K the split is unambiguous, but for even values of K and depending on data, there are regions in which the classification is performed at random. The same description of the earlier plots about false and true positives or negatives can be read from these plots.



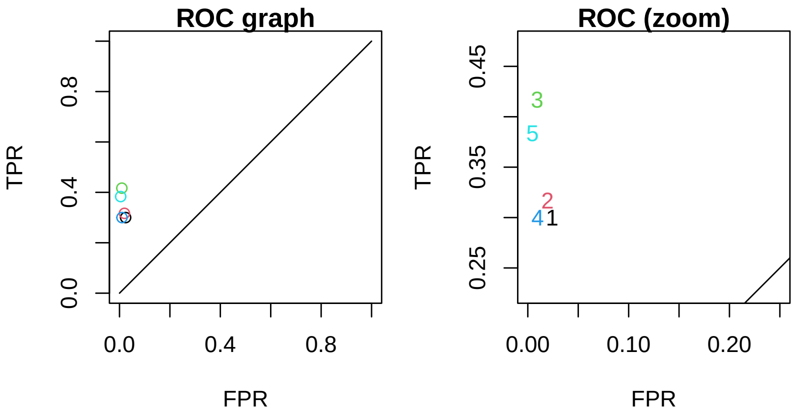
4.6.2 KNN for the `Default` data set

We next apply the KNN classifier for the `Default` data. Recall that the variables are `default, student, balance, income` and that the aim is to classify individuals that default using three other variables. Given that the variable `student` is categorical, the analysis will be performed using the other two variables, which will be standardized. The confusion matrices are shown for $K = 1, 2, 3, 4, 5$. For the data split, we use the same variables `CV`, `Train` and `Test` as before.

```
X<-scale(Default[,3:4],center = TRUE,scale = TRUE) ## Standardize data
library(class)
for(K in 1:5){
   knn(train = X[Train,],test = X[Test,],cl = Default$default[Train],k = K)->KN
   Ytrue<-1*(default[Test]=="Yes"); YKN<-1*(KN=="Yes") ## Adapt labels to 0/1
   print(paste("KNN, K = ",K)); print(table(Ytrue,YKN)) ## Print results
}

## [1] "KNN, K =  1"
##      YKN
## Ytrue    0    1
##     0 1893   47
##     1   42   18
## [1] "KNN, K =  2"
##      YKN
## Ytrue    0    1
##     0 1902   38
##     1   41   19
## [1] "KNN, K =  3"
##      YKN
## Ytrue    0    1
##     0 1922   18
##     1   35   25
## [1] "KNN, K =  4"
##      YKN
## Ytrue    0    1
##     0 1923   17
##     1   39   21
## [1] "KNN, K =  5"
##      YKN
## Ytrue    0    1
##     0 1931    9
##     1   37   23
```

The following table summarizes the performance measures for the different instances of K considered. We put these values in the ROC graph.

K	1	2	3	4	5
FPR	0.02423	0.01959	0.00928	0.00979	0.00464
TPR	0.3	0.31667	0.41667	0.3	0.38333



The best KNN classifier was for $K = 3$. This KNN classifier is better than a random classifier yet quite modest in its performance. Although it is quite capable of singling out negatives with a False Positive Rate (FPR) less than 1%, it has a True Positive Rate (TPR) just above 40%.

Exercise 32 The site `archive.ics.uci.edu/ml/datasets.php` is a good source of data for Machine Learning. In each case below, download data and do a classification analysis. Your analysis should aim to compare the performance of classifiers for a given dataset. Data should be partitioned in training and validation/testing data sets as we have seen in notes and earlier lab. In terms of models/classifiers, at the very least, you should try **logistic** classifier and **KNN** classifier.

1. The **sonar** data. This is data from a study to train a classifier so that it distinguishes between sonar signals bounced by a metal cylinder (`M`) and by a rock (`R`). As a response, define the detection of a mine (metal cylinder) as a positive and detection of a rock as negative and analyze the data. Before attempting analysis, think of ways you would analyse a (partitioned) data set that has $n = 208$ observations in $p = 60$ variables.

2. At the scene of a crime, glass left can be used as evidence . . . if it is correctly identified! This is the **glass** data, from the USA Forensic Science Service. There are $n = 214$ observations in $p = 10$ variables plus the response that indentifies 6 types of glass, defined in terms of their oxide content (i.e. Na, Fe, K, etc). Analyse this data by setting the response to be "window glass" (positive, types of glass equal to $1, 2, 3, 4$) against "non-window glass" (negative).

3. Analyze other datasets from the said site that attract your interest.

Important points and concepts of week eight:

1. Interpret the performance of a classifier using the **ROC curve** and **AUC**.
2. Become familiar with classifiers: **linear**, **logistic** and **KNN**.
3. Practice data analysis with `R`. For this analysis, key `R` functions are the following. For splitting the data

   `cvFolds`;

   for comparisons of classifiers

   `table`, `roc` and `auc`;

   and for building classifiers

   `lm`, `glm`, `predict` and `knn`.

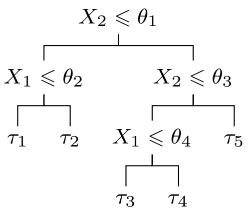
Week nine

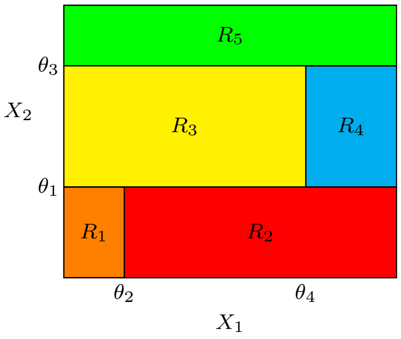
Here we continue the survey of classification methods.

4.7 Classification tree

A classification tree is a structure in which leaves represent class labels and branches represent conjunctions of variables (features) that lead to class labels. For a given set of input values, when moving along the tree, we perform a series of decisions involving the values of the said variables. At the end of this process we have reached a label associated with a leaf and therefore, we have classified the given set of inputs.

As an example, consider the following decision tree below left with variables X_1, X_2. Conventionally, when going down on the tree, in every decision node the 'yes' branch is on the left. The symbols $\tau_1, \tau_2, \tau_3, \tau_4, \tau_5$ are terminal nodes (leaves) and they represent labels. In the diagram below right, each region R_i is attached the label τ_i. Make sure you understand the partitioning induced by the tree, that is that you can relate the branches in the tree to the regions in the diagram.



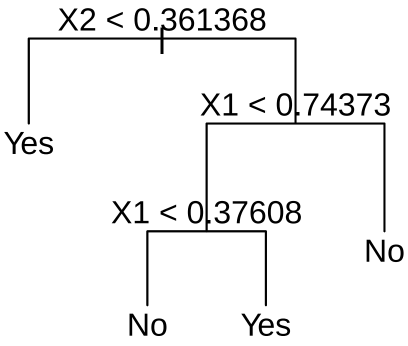


After training, a classification tree provides a series of simple conditions that lead into leaves associated with labels. Trees are **simple to understand and interpret**, as the conditions are simple comparisons of values. Also, trees mirror human decision making more than any other classification approaches. They are able to handle categorical and numerical data without extensive data preparation, do not rely on distributional assumptions and can be validated with testing data.

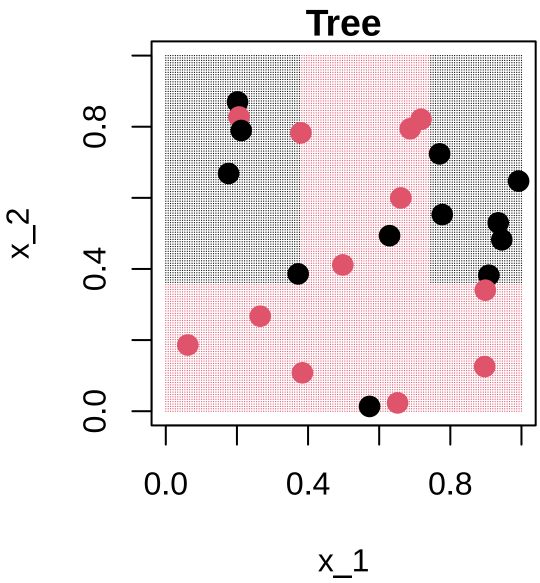
However, trees can have disadvantages. An important one is that training a tree is known to be an **expensive** problem and existing algorithms use greedy searches which in turn do not necessarily guarantee finding an optimum solution. Often the solution obtained over-fits the data and is unnecessarily complex and trees can be very **non robust** and a small change in data may have huge implications on the tree and its predictions. Despite disadvantages, **trees are a valid and useful tool** to consider. We use the function `tree` from the library `ISLR`.

4.7.1 Example synthetic

In this example we use the same synthetic bivariate data seen in an earlier classification example. This data has $n = 24$ points in $p = 2$ variables. After training the classifier on the data, we have the following tree. Recall that 'Yes' labels are positives (red color in scatterplot) and 'No' labels are negatives (black in scatterplot).



This classification tree induces a partition on the plane of variables X_1, X_2. As in previous analyses of the same data, we can identify true positives and true negatives as dots with a background of the same color; false positives and false negatives are dots that are on a background of a different color. This fit has an error rate of 3/24 created by two false positives and one false negative; see these cases in the scatterplot diagram below.



Note the similarities of this partition with the analysis of the same data done earlier with the **KNN** classifier, especially the cases $K = 1$ and $K = 3$. Also note the differences between this partition with the **logistic** classifier seen earlier and with the **linear discriminant classifier** seen later in these notes.

4.7.2 Example `Default` data

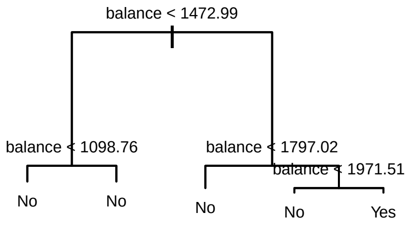
This is the analysis of the `Default` data set we have used earlier. The objective is to classify bank customers in those who will default (positives) and those who won't (negatives). The classifier is a tree, and we use the same partitioning of the data as we have done in the past for this data set .

```
library(ISLR) ## For the data
attach(Default)
library(cvTools) ## For the data split 80:20
set.seed(0); cvFolds(n=nrow(Default),K=5,type = "random")->CV
Train<-CV$subsets[CV$which!=5]; Test<-CV$subsets[CV$which==5]
library(tree)
tree.default<-tree(default~.,Default[Train,]) ## Fit the tree
```

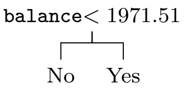
Once the tree has been fitted, we look at summary of the fit and plot the tree.

```
summary(tree.default)

##
## Classification tree:
## tree(formula = default ~ ., data = Default[Train, ])
## Variables actually used in tree construction:
## [1] "balance"
## Number of terminal nodes:  5
## Residual mean deviance:  0.166 = 1327 / 7995
## Misclassification error rate: 0.02788 = 223 / 8000
```

```
plot(tree.default)
text(tree.default,pretty=0)
```



The fitted tree only used `balance` to classify. This is not extremely surprising as for example, the logistic classifier `M1` (see notes Week 9) used it and was the only classifier able to do the task. This tree is highly redundant with four comparisons involving the same variable. The following simple tree is an equivalent one.



As with any method in supervised learning, we need to look at the predictive abilities of the trained method. We use the fresh data we separated for this purpose.

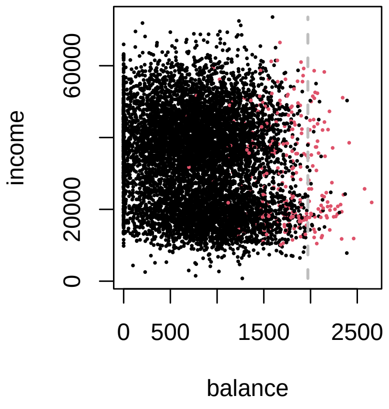
```
predict(tree.default,Default[Test,],type="class")->TR
table(Default$default[Test],TR)

##      TR
##          No  Yes
##   No   1934    6
##   Yes    43   17
```

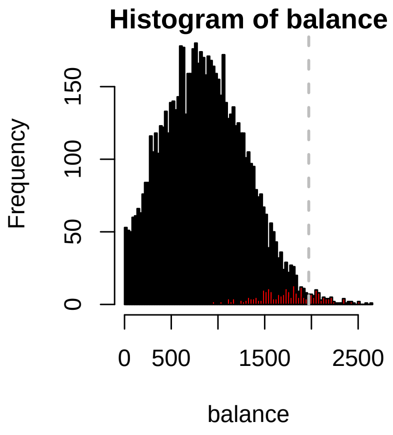
The usual diagnostics are computed for this data and method.

```
##         FPR         TPR
## 0.003092784 0.283333333
```

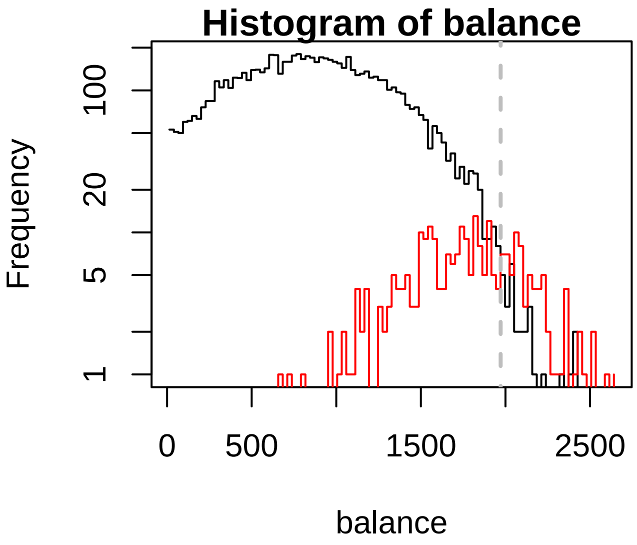
The performance of the tree classifier was not outstanding, indeed it was even worse than the KNN classifier. However this data is a challenging problem.

As additional information, see the scatter plot of variables `income` (unused for the tree) and `balance` for the training sample. We put in red those cases that were defaulted (positive) and added a line which is the threshold value computed by the classification tree. The separation is far from clear.





The histogram of `balance` uses the same color codes and for clarity those cases with zero balance were removed. Two histograms are provided, one with linear and another with logarithmic vertical scale.



Both cases highlight the difficulty of classifying this data set. We want to separate defaults in a situation where those who defaulted are a minority and, in terms of their covariates, are quite mixed with those who didn't.

4.8 Linear discriminant analysis

Linear discriminant analysis is a method that uses a **linear combination** of variables (aka features) to characterize or separate two or more classes of objects or events. This linear combination is used as a classifier (linear classifier).

4.8.1 Developing the classifier

The analysis assumes that the following conditional probability density functions $p(\mathbf{x}|y=0)$ and $p(\mathbf{x}|y=1)$ are both (multivariate) normally distributed with mean and covariance parameters $(\mu_0, \mathbf{\Sigma}_0)$ and $(\mu_1, \mathbf{\Sigma}_1)$. Recall that these multivariate normal densities are

$$p(\mathbf{x}|y=0) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}_0|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_0)^T \mathbf{\Sigma}_0^{-1}(\mathbf{x}-\mu_0)}$$

and

$$p(\mathbf{x}|y=1) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}_1|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_1)^T\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\mu_1)}.$$

In both cases above the variance-covariance matrices are assumed to be full rank.

With the assumptions above, the Bayes optimal solution is to predict points as being from the second class ("positives") if the log of likelihood ratio are bigger than some threshold T. We develop the log of likelihood ratio as follows:

$$\begin{aligned}
\log\left(\frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)}\right) &= \log\left(\frac{|\mathbf{\Sigma}_0|^{1/2}}{|\mathbf{\Sigma}_1|^{1/2}} e^{\frac{1}{2}(\mathbf{x}-\mu_0)^T\mathbf{\Sigma}_0^{-1}(\mathbf{x}-\mu_0)-\frac{1}{2}(\mathbf{x}-\mu_1)^T\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\mu_1)}\right) \\
&= \frac{1}{2}\Big(\log|\mathbf{\Sigma}_0| - \log|\mathbf{\Sigma}_1| + (\mathbf{x}-\mu_0)^T\mathbf{\Sigma}_0^{-1}(\mathbf{x}-\mu_0) \\
&\qquad -(\mathbf{x}-\mu_1)^T\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\mu_1)\Big).
\end{aligned}$$

From the above, we achieve the following condition

$$\log|\mathbf{\Sigma}_0| + (\mathbf{x}-\mu_0)^T\mathbf{\Sigma}_0^{-1}(\mathbf{x}-\mu_0) - \log|\mathbf{\Sigma}_1| - (\mathbf{x}-\mu_1)^T\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\mu_1) > T,$$

where the constant $1/2$ has been absorbed into the threshold T. The condition above is known as quadratic discriminant, as the right hand side above is a quadratic function of $\mathbf{x}$ when the matrices $\mathbf{\Sigma}_0$ and $\mathbf{\Sigma}_1$ are different.

4.8.2 The linear discriminant

In linear discriminant analysis, a further assumption is that the covariance matrices are identical so that $\mathbf{\Sigma} := \mathbf{\Sigma}_0 = \mathbf{\Sigma}_1$. After cancelling terms and developing the inequality above, we have the following inequality

$$\mathbf{x}^T\mathbf{\Sigma}^{-1}(\mu_1 - \mu_0) > \frac{1}{2}\left(T + \mu_1^T\mathbf{\Sigma}^{-1}\mu_1 - \mu_0^T\mathbf{\Sigma}^{-1}\mu_0\right),$$

with T a threshold that will classify (separate) cases. The above condition is just a threshold on the dot product $\mathbf{w}\cdot\mathbf{x} > c$ with quantities $\mathbf{w} = \mathbf{\Sigma}^{-1}(\mu_1 - \mu_0)$ and $c = \frac{1}{2}\left(T + \mu_1^T\mathbf{\Sigma}^{-1}\mu_1 - \mu_0^T\mathbf{\Sigma}^{-1}\mu_0\right)$. In other words, the observation $\mathbf{x}$ belongs to a certain class depending on its location concerning a hyperplane perpendicular to the vector $\mathbf{w}$, and the location of the hyperplane is defined by c.

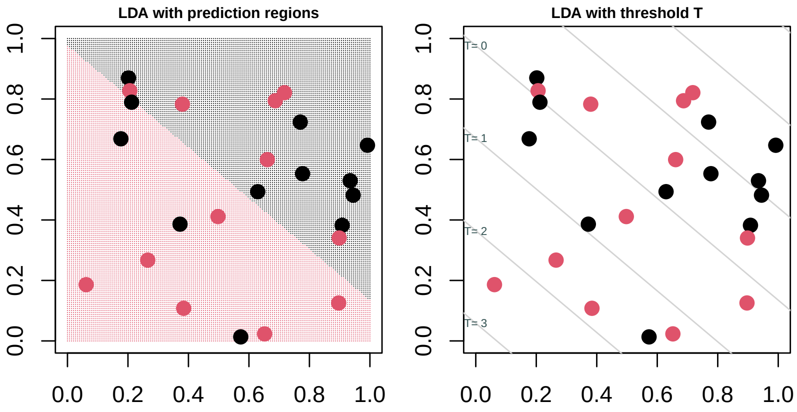
The canonical separating value is when $T = 0$ so that the constant c takes value $c = \frac{1}{2}\left(\mu_1^T\mathbf{\Sigma}^{-1}\mu_1 - \mu_0^T\mathbf{\Sigma}^{-1}\mu_0\right)$. When varying the value of T over the entire real line, we scan over all different patters of classification of the fitted LDA and from this scan, we can build the ROC curve.

In the **Bayesian** version of LDA, each class has a prior probability of individuals belonging to it. The classifiers allocates individuals according to the maximum posterior probability. The case we study is a simple version of the Bayesian linear discriminant analysis, equivalent to the case when the prior probabilities are equal. To perform linear discriminant analysis, we use the function `lda` from the library `MASS`.

4.8.3 Example synthetic

We first have a look at the method using the synthetic data of small size we have used for other classifiers. This dataset has $n = 24$ points in $d = 2$ dimensions.

In the first plot below (left), any observations falling into the red area are classified as positives and negatives otherwise. This plot uses the canonical value $T = 0$. To have a detailed look into linear discriminant analysis, the second plot below (right) is of the same data and LD analysis, when varying the threshold T.



4.8.4 Example `Default` data

This is the classification example we have analysed earlier. The aim is to predict individuals that will default (be unable to pay) using three covariates. The first stage of the analysis involves training the model, using the same split 80 : 20 as earlier.

```
library(ISLR); attach(Default) ## Load the data
library(cvTools) ## To split data 80:20
set.seed(0); cvFolds(n=nrow(Default),K=5,type = "random")->CV
Train<-CV$subsets[CV$which!=5]; Test<-CV$subsets[CV$which==5]
library(MASS) ## Train the model
LD<-lda(default~.,data=Default[Train,]); LD

## Call:
## lda(default ~ ., data = Default[Train, ])
##
## Prior probabilities of groups:
##       No      Yes
## 0.965875 0.034125
##
## Group means:
##     studentYes   balance   income
## No   0.2923515  808.5913 33494.87
## Yes  0.4065934 1737.8864 31489.44
##
## Coefficients of linear discriminants:
##                      LD1
## studentYes -1.269214e-01
## balance     2.235478e-03
## income      2.949078e-06
```

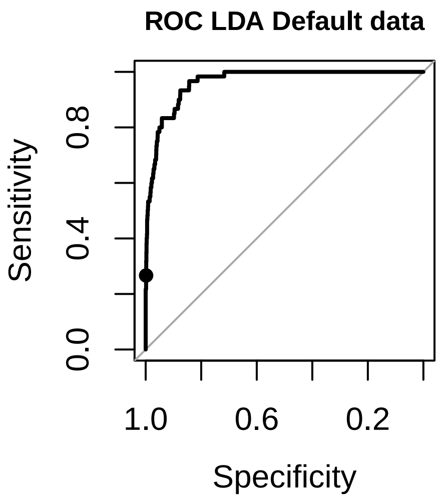
Then predict using fresh data and compute the usual performance statistics.

```
PD<-predict(LD,Default[Test,]); table(default[Test],PD$class)

##
##          No  Yes
##    No  1937    3
##    Yes   44   16
```

```
##         FPR         TPR
## 0.001546392 0.266666667
```

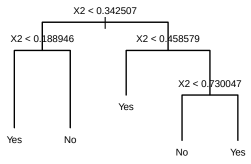
The performance of this method is not extremely different than the other methods we tried previously with the same data, it is even slightly better than the logistic with AUC of 0.96384. A ROC curve can be built for these data and displayed, together with the performance measures for the canonical case.

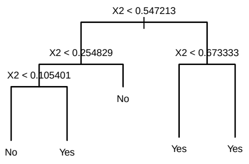
```
library(pROC);     roc(response=default[Test],predictor=PD$x)->RR
plot(RR,col="black",main="ROC LDA Default data")
```

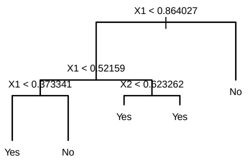


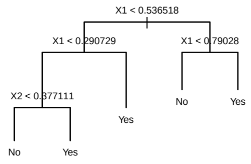
Exercise 33 Consider the data and description of activities of Exercise 25 (week nine). Do the analysis of the **glass** and of **sonar** data using the tree classifier and the linear discriminant classifier. Compare your results with those from earlier classification analysis.

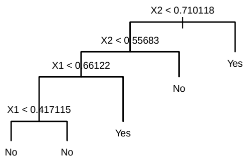
Exercise 34 The following are six fitted classification trees. For each tree, a) sketch the regions of $[0, 1]^2$ that are classified as positives and as negatives and b) (**Extra**) write an alternative tree that achieves the same classification. The alternative tree can be a simplified version of the given tree.

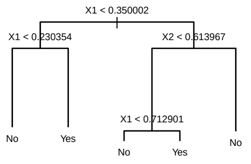












Important points and concepts of week nine:

1. Become familiar with the concepts and ideas behind classifiers: classification tree and linear discriminant analysis

2. Practice data analysis with `R`. The key functions for this part of the Module are `tree` and `lda`.

Weeks ten and eleven

5 Lasso and regularization (Penalized likelihood)

Penalization techniques have been proposed with the objective of improving the performance of traditional estimators. In this chapter we introduce the basics of three penalization proposals that gravitate around regression and extensions: Ridge regression, Lasso and penalized likelihood. In contrast with classification problems, here **the response vector has real valued entries**.

5.0.1 Standard linear regression

Parameter estimation in many statistical problems is performed by minimizing the sum of squared errors, leading to the standard **least squares estimate**. Consider the linear model

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{1}$$

where the matrix $\mathbf{X}$ has dimensions $n \times p$ with $n > p$, and it is assumed that $\mathbf{X}$ is full rank, i.e. $\text{rank}(\mathbf{X}) = p$. The real-valued responses are collected in the vector $\mathbf{Y}$. For this model, the least squares estimate of $\boldsymbol{\beta}$ is the well known regression formula

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}, \tag{2}$$

which minimizes the quadratic criterion

$$\text{SSE} = ||\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}||_2^2.$$

5.0.2 Notation on norms

The symbol $|| \cdot ||_2$ is the usual Euclidean norm as seen elsewhere in mathematics and also known as the L_2 norm. We have used this norm earlier in this Module for clustering as a distance between two points (vectors) and as a norm it is the distance between a vector and the origin.

The notation $|| \cdot ||_2^2$ is the squared euclidean norm (squared L_2 norm). The squared euclidean norm has an appealing development in terms of vectors, indeed for a column vector $\mathbf{x}$, we have $||\mathbf{x}||_2^2 = \mathbf{x}^T\mathbf{x}$. In particular, we have

$$\text{SSE} = ||\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}||_2^2 = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{Y}^T\mathbf{Y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{Y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

As for other norms, note that the general case of norm $|| \cdot ||_m$ is the Minkowski distance (or L_m norm) and in particular $|| \cdot ||_1$ is the Manhattan distance or L_1

norm. In all cases, the argument $\cdot$ of the distances are vectors, which are usually column vectors.

Consider the vector $\mathbf{x} = (4, -1, 2)^T \in \mathbb{R}^3$. Its L_2 norm (Euclidean) is $||\mathbf{x}||_2 = \sqrt{(4)^2 + (-1)^2 + (2)^2} = \sqrt{16+1+4} = \sqrt{21} = 4.58258$. Its squared euclidean norm is $||\mathbf{x}||_2^2 = \mathbf{x}^T\mathbf{x} = 21$. Its L_1 norm ('Manhattan') is $||\mathbf{x}||_1 = |4| + |-1| + |2| = 4+1+2 = 7$.

5.1 Ridge regression

The criterion $||\mathbf{Y} - \mathbf{X}\beta||_2^2$ is a measure of the distance (misfit) between data $\mathbf{Y}$ and the predictor $\mathbf{X}\beta$. Now suppose we are interested as well in the criterion $||\beta||_2^2$, which is a measure of the size of the coefficient vector $\boldsymbol{\beta}$. The goal is to find a good fit, that is to find a value of $\boldsymbol{\beta}$ that achieves a small value of $||\mathbf{Y} - \mathbf{X}\beta||_2^2$ and in such a way that $\boldsymbol{\beta}$ is not too large, that is, attaining a small value of $||\beta||_2^2$.

This search can be formulated as a dual objective problem in optimisation theory. This dual problem can be scalarized to have a single objective function. After simplification, the objective function becomes

$$R = ||\mathbf{Y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_2^2, \tag{3}$$

to be minimised for positive λ. This is a **regularized** (that is, L_2 regularized) least squares problem, whose solution has the following closed form

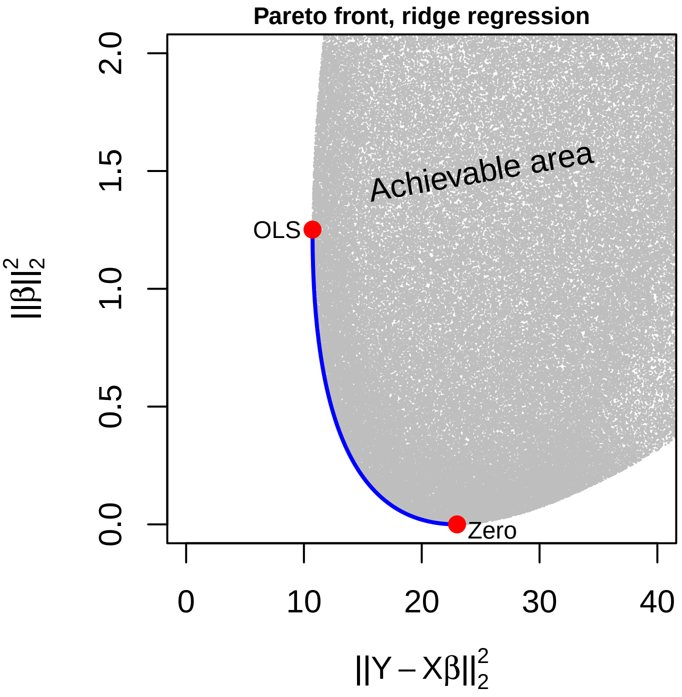
$$\hat{\boldsymbol{\beta}}^R = \hat{\boldsymbol{\beta}}^R(\lambda) = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{Y}. \tag{4}$$

This estimator $\hat{\boldsymbol{\beta}}^R$ is the ridge regression estimator.

5.1.1 The ridge trace

In optimization terminology, the solution $\hat{\boldsymbol{\beta}}^R$ of ridge regression is Pareto-optimal for any positive λ. There are two extremes: when $\lambda \to 0$ and when $\lambda \to \infty$. The first instance corresponds to the ordinary least squares estimator: $\hat{\boldsymbol{\beta}}^R = \hat{\boldsymbol{\beta}}$ of Equation 2, and the second is the solution $\hat{\boldsymbol{\beta}}^R = \mathbf{0}$. Both solutions are also Pareto-optimal.

The following figure shows the behaviour of solutions to a ridge regularized regression problem. The plane in the figure has X-Y coordinates given by the two criteria $||\mathbf{Y} - \mathbf{X}\beta||_2^2$ and $||\beta||_2^2$. The shaded area is the set of achievable values $\left(||\mathbf{Y} - \mathbf{X}\beta||_2^2, ||\beta||_2^2\right)$. The border of this area is termed the **Pareto front**, and the optimal trade-off curve is achieved by the minimizer solution of Equation 3.



The trajectory of ridge regression coefficients $\hat{\boldsymbol{\beta}}^R$ as a function of λ for $\lambda \geqslant 0$ is known as the **ridge trace**. As has been described earlier, for each coefficient $\hat{\beta}_j^R$ the ridge trace **morphs** continuously the value $\hat{\beta}_j^R$ between the ordinary least squares estimate $\hat{\beta}_j$ (obtained when $\lambda = 0$) and zero (when $\lambda \to \infty$).

5.1.2 Ridge and other techniques

Ridge regression provides a useful alternative to least squares. As we move along the ridge trace (increasing λ), the shrinkage of the coefficient leads to a substantial reduction in the variance of predictions, at the expense of an increase in the bias. It also has advantages over subset selection in regression both on the prediction performance and computational complexity. We must be aware that ridge regression is not a variable selection technique, as the coefficients only shrink to zero when $\lambda \to \infty$.

Ridge regression has close links with principal component analysis. To elaborate briefly on this, recall the singular value decomposition of the matrix of covariates $\mathbf{X} = \mathbf{UDV}^T$. We begin with the predictions of standard linear regression, which can be easily shown as

$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{HY} = \mathbf{UU}^T\mathbf{Y},$$

where $\mathbf{H} = \mathbf{UU}^T$ is the hat matrix for predictions. Note that $\mathbf{U}^T\mathbf{Y}$ above are the coordinates of $\mathbf{Y}$ with respect to the orthonormal basis $\mathbf{U}$.

The predictions of ridge regression can be shown to be

$$\mathbf{X}\hat{\boldsymbol{\beta}}^R = \underbrace{\mathbf{UD}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{DU}^T}_{\mathbf{H}_\lambda}\mathbf{Y} = \sum_{j=1}^{p} \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda}\mathbf{u}_j^T\mathbf{Y},$$

where $\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}$ is a diagonal matrix with entries $d_1^2/(d_1^2 + \lambda), \ldots, d_p^2/(d_p^2 + \lambda)$; and the matrix $\mathbf{H}_\lambda = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T$ is the hat matrix for ridge predictions. Indeed, like linear regression (see Exercise 15 in Page 22, week three), ridge regression also computes the coordinates of $\mathbf{Y}$ with respect to the orthonomal basis $\mathbf{U}$. The coordinates are then shrank by factors $d_j^2/(d_j^2 + \lambda)$ so that greater amount of shrinkage is performed in vectors with smaller value d_j^2. Recall that d_j is an eigenvalue of the singular decomposition of $\mathbf{X}$ and that eigenvalues of principal components of $\mathbf{X}$ are related to those of its singular value decomposition via the relation $\boldsymbol{\Lambda} = \mathbf{D}^2/(n-1)$ seen much earlier in this Module, see also Exercise 35. In other words, small singular values correspond to directions in the column space of $\mathbf{X}$ having small variance, and the ridge regression shrinks these directions the most.

5.1.3 Computations

The main problem of ridge regression is to select the value of λ. There are several options for this. A subjective suitable value of λ would be the smallest λ such that the coefficients stabilize in this plot. It is also possible to select λ on the basis of selecting minimal error in a cross-validation procedure. Finally, another proposal is to plot the ridge trace against what is known as the *effective degrees of freedom*, defined as

$$\mathrm{df}(\lambda) = \mathrm{trace}(\mathbf{H}_\lambda) = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda},$$

where d_j are the (singular) eigenvalues of $\mathbf{X}$.

5.1.4 Example: `Credit` dataset

The **`Credit`** dataset has the response variable **`Balance`** (debt) which is to be modelled with ridge regression using the quantitative predictor variables **`Income, Limit, Rating, Cards, Age, Education`**. The data consists of $n = 400$ observations and to select a value of λ we have split the data in $K = 4$ folds, three of which are used for training the model and the last one is used for selecting λ.

```r
library(ISLR)
DAT<-Credit[,c(2:7,12)] ## Loading the data set
DAT<-scale(x = DAT,center=TRUE,scale=TRUE) ## centering and scaling
library(cvTools) ## CV for splitting in folds
set.seed(0);
cvFolds(n=nrow(DAT),K = 4,type = "random")->CV
CV$subsets[CV$which!=4]->Train; CV$subsets[CV$which==4]->Test
```

For the ridge, we use a simple function which is coded below.

```r
ridge<-function(Xtrain,Ytrain,Xval,Yval,lval){
  S<-svd(x=Xtrain);  ## Svd to compute ridge
  N<-length(lval);  ## Number of lambda values
  betav<-matrix(nrow=ncol(Xtrain),ncol=N); ## Objects for the analysis and output
  Ypred<-matrix(nrow=nrow(Xval),ncol=N); DF<-MSE<-matrix(ncol=N,nrow=1)
  rownames(betav)<-colnames(Xtrain)
  for(i in 1:N){
    lambda<-lval[i]
    betav[,i]<-c( (S$v)%*%diag(S$d/(S$d^2+lambda))%*%t(S$u)%*%matrix(ncol=1,Ytrain
    Ypred[,i]<-Xval%*%matrix(ncol=1,betav[,i]) ## Predictions
    MSE[1,i]<-mean( (  Yval -Ypred[,i]    )^2  )  ## MSE
    DF[1,i]<-sum( S$d^2/(S$d^2+lambda) )  ## Effective degrees of freedom
  }
  return(list("beta"=betav,"MSE"=c(MSE),"df"=DF))
}
```

The data was standardized in the training step and we selected values of λ ranging from 10^{-4} to 10^{6}.

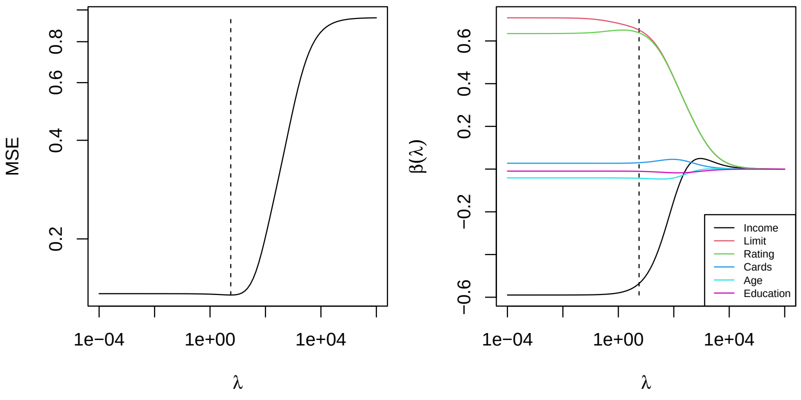
```r
rangelambda<-10^seq(from=-4,to=6,length.out = 100) ## values of lambda
LR<-ridge(Xtrain=DAT[Train,1:6],Ytrain = DAT[Train,7],
          Xval=DAT[Test,1:6],Yval=DAT[Test,7],lval=rangelambda)
```

At this stage, the output `LR$MSE` has the prediction errors. From these values, we select the value of λ:

```
which.min(LR$MSE) ## Index for the value of lambda that minimizes MSE

## [1] 48

LR$MSE[which.min(LR$MSE)]  ## minimum value of MSE

## [1] 0.1348048

rangelambda[which.min(LR$MSE)]->lambdamin; lambdamin ## minimum lambda

## [1] 5.59081
```

The minimum value is $\text{MSE}(\tilde{\lambda}) = 0.1348$, which is achieved at $\tilde{\lambda} = 5.59081$. We next plot the validation MSE as well as the ridge trace.

```
par(mar=c(4,4,1,1),mfrow=c(1,2))
plot(x=rangelambda,y=LR$MSE,log="xy",xlab=expression(lambda),ylab="MSE",type="l")
lines(lambdamin*c(1,1),range(LR$MSE),lty=2)
plot(range(rangelambda),range(LR$beta),type="n",log="x",xlab=expression(lambda),
     ylab=expression(beta(lambda)));  lines(lambdamin*c(1,1),range(LR$beta),lty=2)
for(i in 1:nrow(LR$beta)) lines(rangelambda,LR$beta[i,],col=i)
legend("bottomright",legend=colnames(DAT)[1:6],lty=1,col=1:6,cex = 0.6)
```

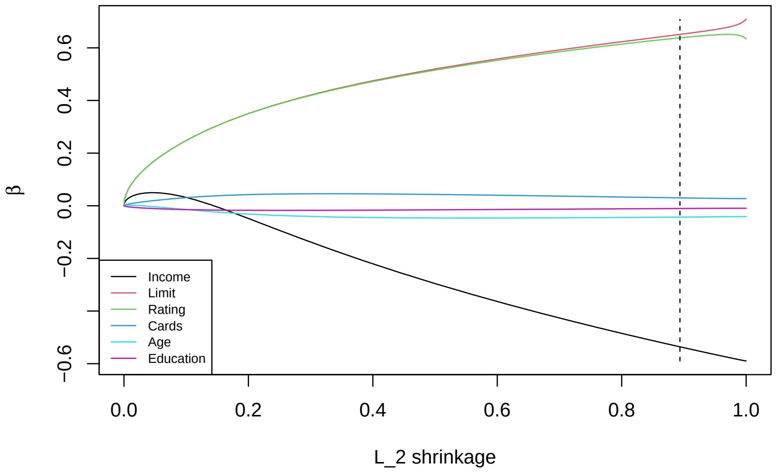


Here we give the coefficients of the selected model

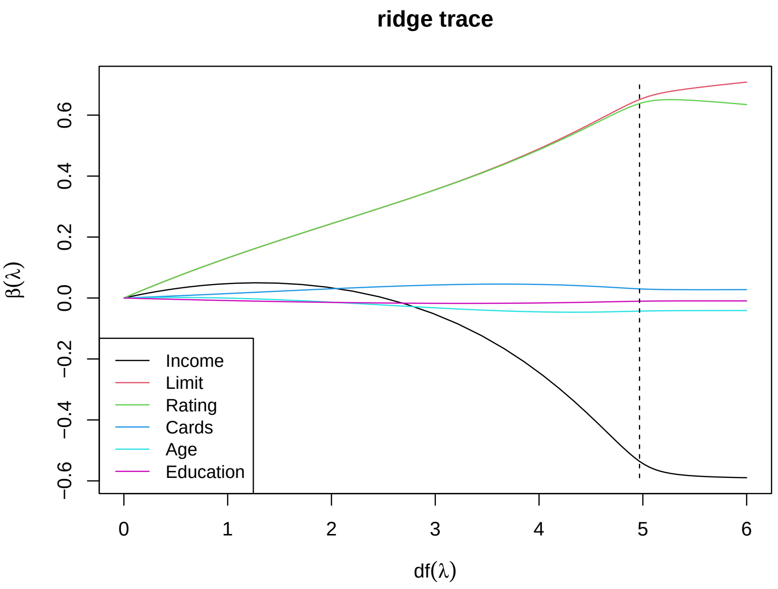
```
LR$beta[,which.min(LR$MSE)]

##      Income       Limit      Rating       Cards         Age   Education
## -0.53586266  0.65104988  0.63814381  0.02997686 -0.04298134 -0.01050584
```

and then a plot of the ridge trace against the proportion of (squared L_2) shrinkage.

```
stdnorm<-apply(X = LR$beta^2,MARGIN = 2,FUN = sum);
stdnorm<-stdnorm/max(stdnorm) ## L2 shrinkage
plot(c(0,1),range(LR$beta),type="n",log="",xlab="L_2 shrinkage",ylab=expression(bet
for(i in 1:nrow(LR$beta)) lines(stdnorm,LR$beta[i,],col=i)
legend("bottomleft",legend=rownames(LR$beta),lty=1,col=1:6,cex = 0.7)
lines(c(1,1)*stdnorm[which.min(LR$MSE)],range(LR$beta),lty=2)
```



As a final plot for the `Credit` data analysis, we plot the ridge trace against effective degrees of freedom $\mathrm{df}(\lambda)$. As in the other plots, the dashed line indicates the value of $\tilde{\lambda}$ (i.e. $\mathrm{df}(\tilde{\lambda})$ in this plot) selected using the validation data.



Note that along the ridge trace, the continuous transformation of $\hat{\boldsymbol{\beta}}^R$ from the least squares estimate $\hat{\boldsymbol{\beta}}$ to zero is often monotonic but not necessarily so, as is evident in the trace for the coefficient of `Income`.

Important points and concepts of weeks ten and eleven:

1. The concept of regularization and the case of ridge regression.
2. Concerning the ridge trace, understand the different versions and the computations involved.
3. Practice data analysis with `R`. In this section we did our own function for this analysis as `ridge`.

Week twelve

5.2 Lasso

Despite its advantages for prediction, a clear drawback of ridge regression is that it generally selects models that involve all the p variables, contrary to model selection techniques like best subset, forward stepwise or backward stepwise regression. The **lasso** (least absolute shrinkage and selection operator) is a relatively recent technique that overcomes this disadvantage by using L_1 penalization.

Using the same principles as for ridge regression, consider a **dual** objective optimization of the criteria $||\mathbf{Y} - \mathbf{X}\beta||_2^2$ and $||\beta||_1$. The first quantity is the residual sum of squares criterion as seen earlier, while the second is the L_1 norm (Manhattan) of the vector of parameters $\boldsymbol{\beta}$. Both quantities are to be minimized and just like in ridge regression, the problem can be scalarized into a single criterion. After simplification, the **lasso** problem is the minimization of

$$L = \frac{1}{2}||\mathbf{Y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_1. \tag{5}$$

This is a L_1 **regularized** least squares problem, in which the quantity L is a convex function for all $\boldsymbol{\beta}$ and positive λ. Contrary to ridge regression, there is **no** general closed form solution for the minimization of lasso criterion L.

5.2.1 The lasso path

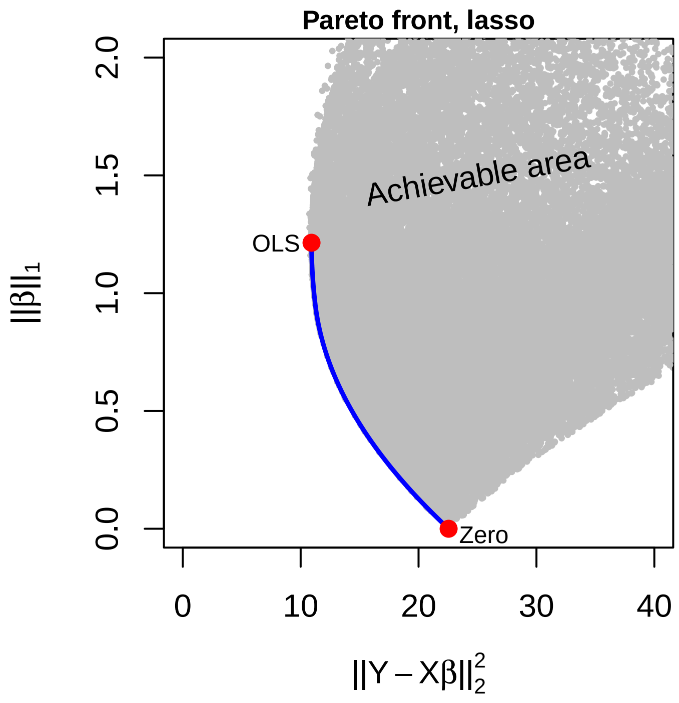
For fixed λ, the lasso estimates are the quantities $\hat{\boldsymbol{\beta}}^L = \hat{\boldsymbol{\beta}}^L(\lambda)$ which minimise L. The set of solutions to the lasso problem is known as the **lasso path**. In the lasso path, the estimates $\hat{\boldsymbol{\beta}}^L(\lambda)$ shrink linearly as function of λ, but these trajectories are piecewise linear and not globally linear. See the later example for the detailed construction of the path.

Mirroring the earlier development of ridge regression, $\hat{\boldsymbol{\beta}}^L$ of lasso is Pareto-optimal for any positive λ. We also have the **same two extremes** as for ridge: when $\lambda \to 0$ and when $\lambda \to \infty$. The first instance corresponds to the ordinary least squares estimator $\hat{\boldsymbol{\beta}}$ of Equation 2, and the second is the solution $\hat{\boldsymbol{\beta}}^L = \mathbf{0}$. Both solutions are also Pareto-optimal.

Despite similarities with ridge regression, the lasso path is very different than the ridge trace and an attractive feature of the lasso path is that coefficients effectively shrink to zero. Indeed the shrinking all of coefficients to zero in the lasso path takes place at a finite λ, as opposed to ridge regression where this only happens as $\lambda \to \infty$.

For the above reason, lasso has a **dual** function as an estimation procedure and a variable selection method. An advantage of this feature of lasso is that models obtained from it are simpler to interpret than those from ridge regression as they are sparser, that is they are smaller in size.

The following figure shows the set of achievable solutions for an example of the dual problem described above. The figure also shows the border (Pareto front) of this dual problem which is precisely the optimal trade-off curve that minimizes L of Equation 5.



As in the case of ridge regression, a paramount problem of the lasso is to select a value of λ. This is usually achieved by splitting the data with some form of cross-validation and minimization of the validation error computed with fresh data.

Also, when performing lasso analyses, it is customary to standardize the data. Specifically, after centering the response, the model fitted involves **no intercept** which in most cases, is of little interest.

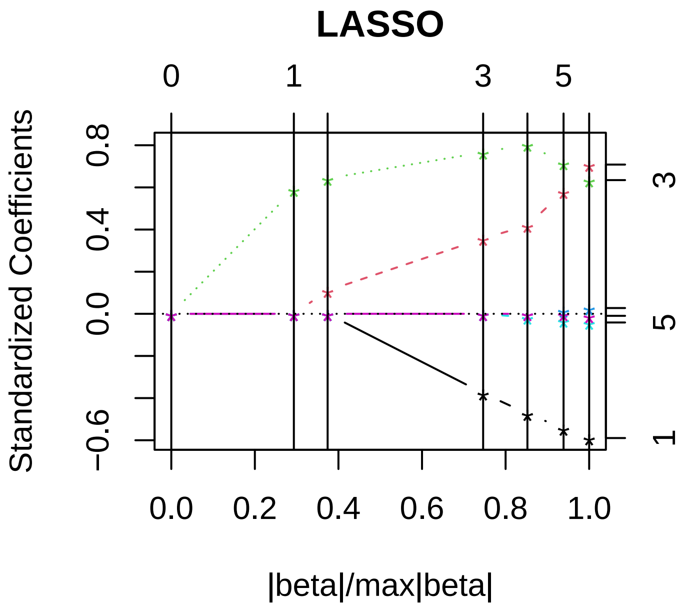
5.2.2 Example: `Credit` data

We now analyze the same `Credit` data set we analysed with ridge regression. The initial part of the code with data preparation remains as before, with a 75:25 split.

```
library(ISLR); library(lars) ## for data and lasso
DAT<-Credit[,c(2:7,12)]; ## loading the data set
library(cvTools) ## CV for splitting in folds
set.seed(0); cvFolds(n=nrow(DAT),K = 4,type = "random")->CV
CV$subsets[CV$which!=4]->Train; CV$subsets[CV$which==4]->Test
DAT<-scale(x=DAT,center=TRUE,scale=TRUE) ## center and scale data
```

The machine learning algorithm is trained. We then plot the lasso path.

```
lars(x=DAT[Train,-c(7)],y=DAT[Train,7],type = "lasso",intercept = FALSE,
     normalize=FALSE)->LS;          plot(LS)
```



There are two versions of the lasso path and they differ in the horizontal axis. In either version, the vertical axes plots the values of the different coefficients so it is a multi-axis. In one version of the path, the horizontal axis is the regularization parameter λ, and in the second version, the horizontal axis is the **proportion of shrinkage**, that is the L_1 shrinkage of $\hat{\beta}^L$, computed with respect to the largest L_1 norm of $\hat{\beta}^L$. The proportion of shrinkage is $||\beta(\lambda)||_1/\max_\lambda||\beta(\lambda)||_1$, which in the plot above is labelled as `|beta|/max|beta|`.

This second plot is the most common version, implemented in `R` when plotting the output of package `lars`. We will next study the lasso path in some detail.

5.2.3 Example `Credit` data: deconstructing the lasso path

The starting point of the lasso path is the vector of ordinary least squares. Here we compute with the function `lm` these coefficients. Note **removal** of the intercept.

```
round(lm(DAT[Train,7]~DAT[Train,-c(7)]-1)$coefficients,5)

##    DAT[Train, -c(7)]Income     DAT[Train, -c(7)]Limit
##                   -0.58959                    0.70833
##    DAT[Train, -c(7)]Rating     DAT[Train, -c(7)]Cards
##                    0.63461                    0.02745
##       DAT[Train, -c(7)]Age DAT[Train, -c(7)]Education
##                   -0.04115                   -0.00954
```

The end of the lasso path is at the point where the coefficients have all shrank to zero. It can be shown that this happens at $\lambda_{\text{max}} = \max_i\{|\mathbf{X}^T\mathbf{Y}|_i\} = 259.78686$

```
abs( t(DAT[Train,-c(7)])%*%DAT[Train,7] )## |X^TY|

##                [,1]
## Income    134.98281
## Limit     259.08415
## Rating    259.78686
## Cards      22.97758
## Age        14.44241
## Education  16.62844
```

The lasso path has a series of breakpoints, and **between** these breakpoints the coefficients move linearly, i.e. the path is a series of piecewise linear trajectories. The following is the table of coefficients at breakpoints. Each **column** is for a coefficient, each **row** is for a breakpoint (λ).

The **last** row is the **least squares estimate**, and the **first** row consists of zeroes, i.e. complete shrinkage of coefficients. Therefore the lasso path is represented in the table from end (top of table) to beginning (bottom of table).

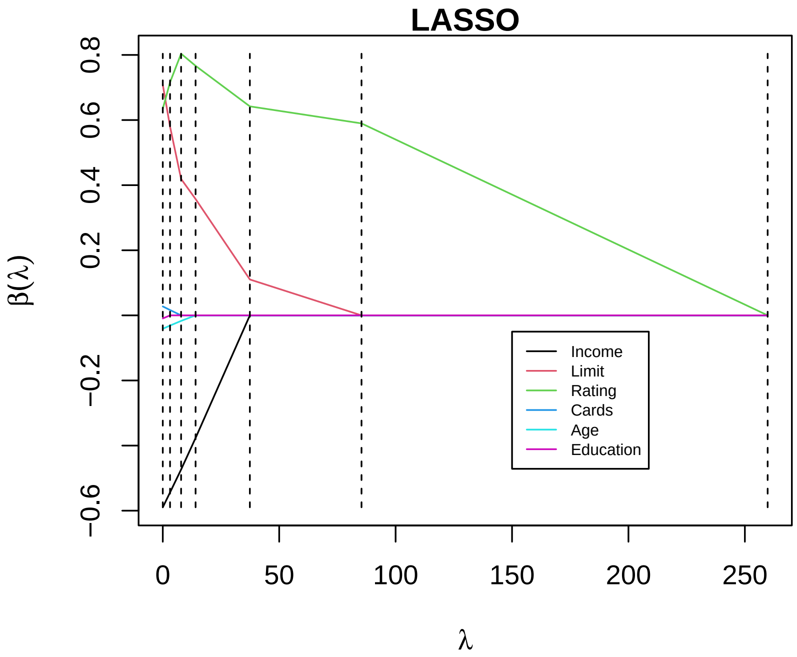
```
round(LS$beta,5)

##      Income   Limit  Rating   Cards      Age Education
## 0   0.00000 0.00000 0.00000 0.00000  0.00000   0.00000
## 1   0.00000 0.00000 0.58972 0.00000  0.00000   0.00000
## 2   0.00000 0.11025 0.64203 0.00000  0.00000   0.00000
## 3 -0.37659 0.35734 0.76649 0.00000  0.00000   0.00000
## 4 -0.47360 0.41909 0.80378 0.00000 -0.01705   0.00000
## 5 -0.54395 0.57971 0.71604 0.01636 -0.03134   0.00000
## 6 -0.58959 0.70833 0.63461 0.02745 -0.04115  -0.00954
## attr(,"scaled:scale")
## [1] 1 1 1 1 1 1
```

The values of λ for the breakpoints are given in the list `LS$lambda`. To this list we need to add $\lambda = 0$ (value of λ for the bottom row). For the current analysis there are $6 + 1 = 7$ breakpoints.

```
LS$lambda

## [1] 259.786861  85.353801  37.402567  14.093052   7.866418   3.124413
```

The plot of coefficients can be done against λ (below) or against the proportion of L_1 shrinkage $||\boldsymbol{\beta}(\lambda)||_1/\max_\lambda||\boldsymbol{\beta}(\lambda)||_1$ (plot in Page 115).

```
LAMB<-c(LS$lambda,0)
plot(range(LAMB),range(LS$beta),type="n",xlab="lambda", ylab="beta(lambda)")
for(i in 1:ncol(LS$beta)) lines(LAMB,LS$beta[,i],col=i)
for(lambda in LAMB) lines(lambda*c(1,1),range(LS$beta),lty=2)
```

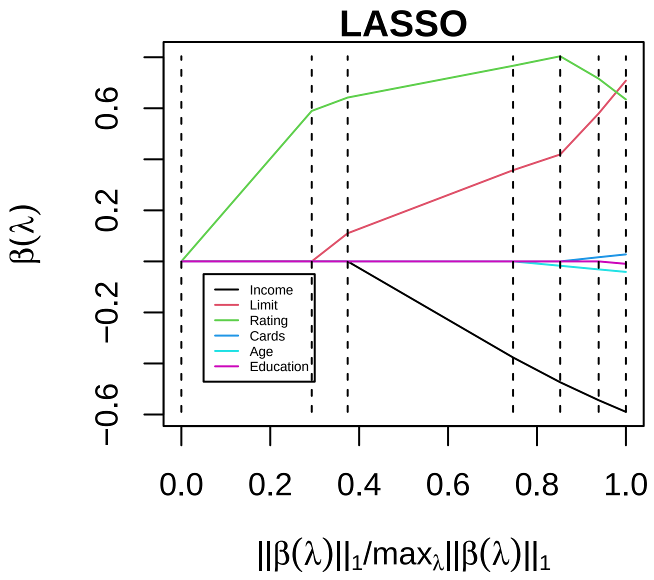


This is the customary lasso path. In the left of the plot we have the **least squares** estimates; on the right the coefficients have all shrank to zero. In between, coefficients move linearly between breakpoints.

We now compute the proportion of shrinkage $||\beta(\lambda)||_1/\max_\lambda||\beta(\lambda)||_1$. Recall that $||\beta(\lambda)||_1$ is the L_1 norm of vector $\hat{\beta}(\lambda)$. In other words, add by row the table of absolute values of `LS$lambda`. Then divide by the maximum, collect in a variable and plot. The plot we constructed below is the same as that in Page 115.

```
(apply(abs(LS$beta),1,sum)->s);  (s/max(s)->s)

##         0         1         2         3         4         5         6
## 0.0000000 0.5897186 0.7522750 1.5004130 1.7135091 1.8873984 2.0106665
##         0         1         2         3         4         5         6
## 0.0000000 0.2932951 0.3741421 0.7462267 0.8522095 0.9386929 1.0000000
```

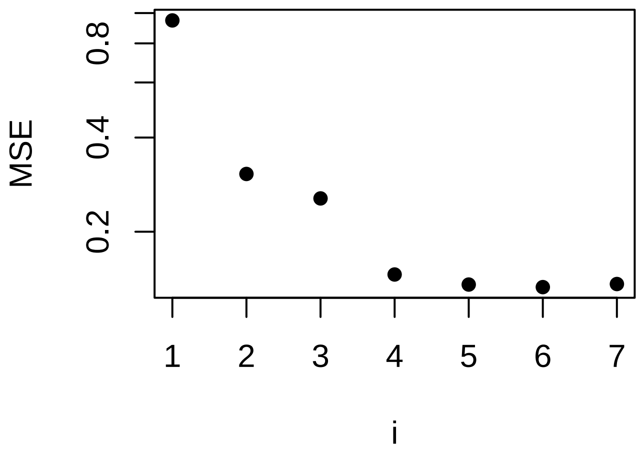
```
plot(c(0,1),range(LS$beta),type="n",xlab="|b|/max|b|",ylab="b",main="LASSO")
for(i in 1:ncol(LS$beta)) lines(s,LS$beta[,i],col=i)
for(mu in s) lines(mu*c(1,1),range(LS$beta),lty=2)
legend(x=0.05,y=-0.05, legend=colnames(LS$beta),lty=1,col=1:6,cex = 0.4)
```



5.2.4 Example `Credit` data set: selecting the value of λ

To construct the validation error (MSE) there are at least two possibilities. One is to use the coefficients at breakpoints with their actual shrunk values. This is done next.

```
predict(object = LS, newx = DAT[Test,-c(7)],type="fit")->PL
matrix(nrow=nrow(DAT[Test,]),ncol=ncol(PL$fit),byrow=FALSE,DAT[Test,7])->Yobs
apply((Yobs-PL$fit)^2,2,mean)->MSE
which.min(MSE); min(MSE) ## breakpoint that minimizes MSE and minimum MSE
plot(PL$s,MSE,xlab="i",ylab="MSE",log="y",pch=16)
```



```
## [1] 6
## [1] 0.1330342
```

The following are the coefficients of the selected model.

```
LS$beta[which.min(MSE),]
```

```
##      Income       Limit      Rating       Cards         Age   Education
## -0.54395274  0.57970850  0.71604051  0.01635805 -0.03133861  0.00000000
```

Another way to estimate the error is to reestimate these coefficients to ameliorate the bias. We next do this, noting that in the lasso path some of the variables may have been removed from the analysis.

Continuing the current example and using the training data, we reestimate model coefficients. Recall that variable `Education` has been removed from the model.

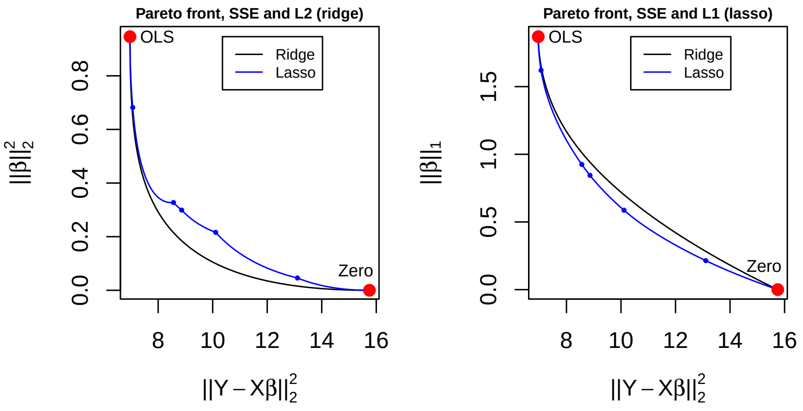
```
##          Income    Limit  Rating      Cards         Age
## [1,] -0.5903093 0.6855393 0.65823 0.02713605 -0.04075592
```

With these reestimated coefficients, the updated Mean squared error using the test data is 0.1344. Compare this value with the earlier (without reestimating coefficients) lasso MSE of 0.13303 and with the MSE results from ridge regression.

5.2.5 Lasso and ridge, elastic nets

Lasso and ridge regression are alternative methodologies to improve regression performance. Ridge regression and lasso are solutions of different optimization problems and the paths (trajectories of coefficients) consequently differ.

For a given data set, we show this contrast by depicting the pareto fronts of L_2 regularization (ridge) and of L_1 regularization (lasso). For the L_2 case (plot on the left), trivially the ridge trace is optimal, while the lasso path also shown is not, and this is shown by its position to the "northeast" of the ridge trace. The situation reverses for L_1 (plot on the right), where the lasso path is the optimal solution and ridge trace becomes suboptimal. In both plots, the dots in lasso path correspond to breakpoints.

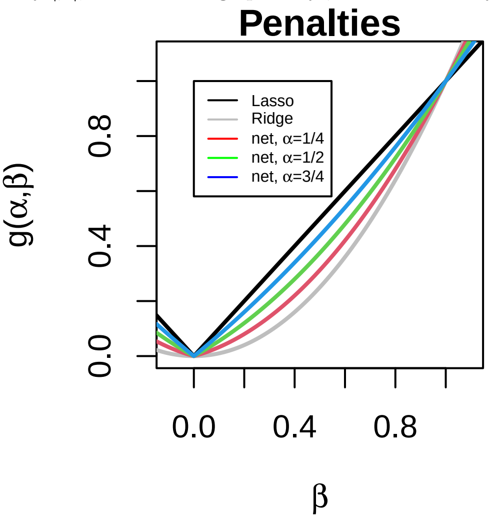


While ridge regression is known to improve prediction capability, lasso can work as estimation and variable selection technique and thus it is an alternative to model selection procedures. Given these relative advantages of lasso (variable selection) and ridge (prediction) an extension of both lasso and ridge are **elastic nets**. These are created by a new penalisation which is a linear combination of L_1 from lasso and L_2 from ridge. For constant α and the vector of parameters $\boldsymbol{\beta}$, the penalization for elastic nets is

$$g(\alpha, \boldsymbol{\beta}) = \left(\alpha||\boldsymbol{\beta}||_1 + (1-\alpha)||\boldsymbol{\beta}||_2^2\right) = \sum_{j=1}^{p} \left(\alpha|\beta_j| + (1-\alpha)\beta_j^2\right).$$

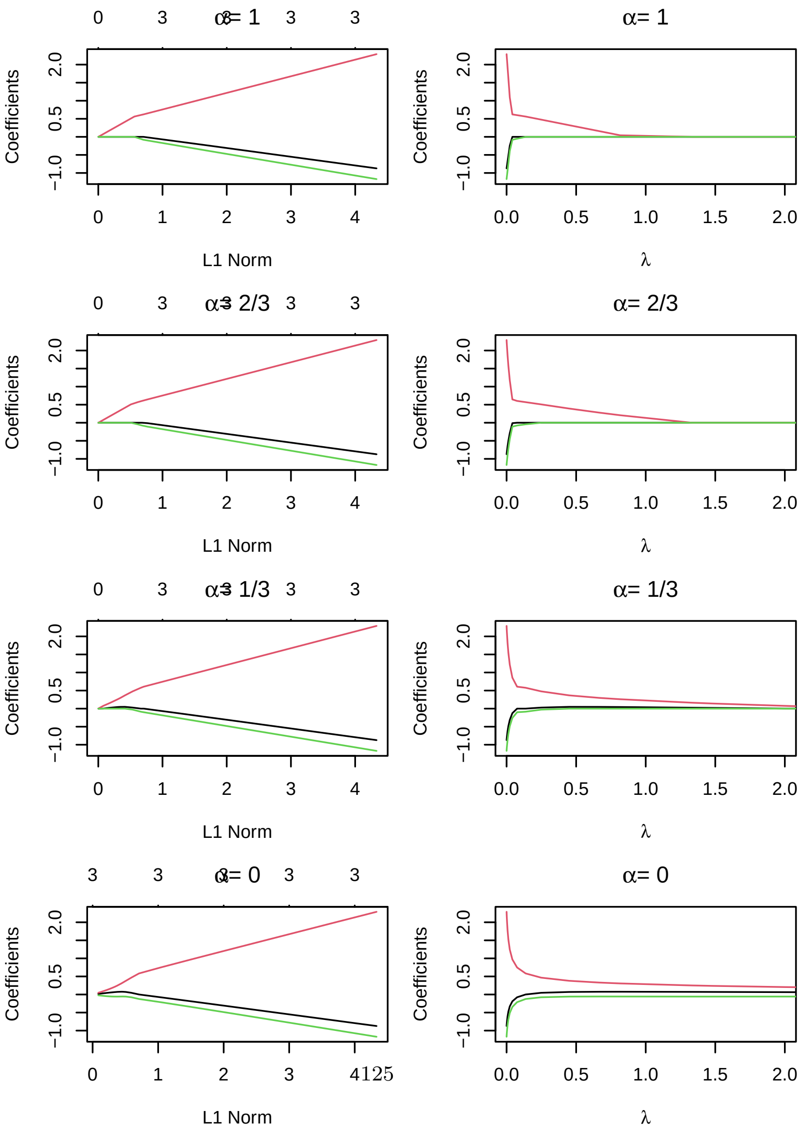
The penalty for elastic net $g(\alpha, \boldsymbol{\beta})$ is a weighted average between Lasso and ridge penalties that, depending on the value of α, will lie closer to either Lasso or ridge, or when $\alpha = 1/2$ it is equidistant between them. An advantage of $g(\alpha, \boldsymbol{\beta})$ is that it has a differentiable component which improves over Lasso while keeping close to it. However, a drawback of elastic net is the need to specify the value of α. Observe in the figure the morphing between the non differentiable (at the origin) absolute value of Lasso and the smooth parabola of ridge.

We give an example of the elastic net penalisation for a single parameter β. The figure below shows the net penalty $g(\alpha, \beta) = \left(\alpha|\beta| + (1-\alpha)\beta^2\right)$ as function of β for $\alpha = 0, 1/4, 1/2, 3/4, 1$. Note that for $\alpha = 1$, the net penalty becomes the Lasso penalty $|\beta|$; while the ridge penalty β^2 is included by considering $\alpha = 0$.



The basic function for this analysis is `glmnet` of the package with the same name. The user supplies a desired value of `alpha`. The parameter values of `lambda` can be also specified and are otherwise automatically calculated.

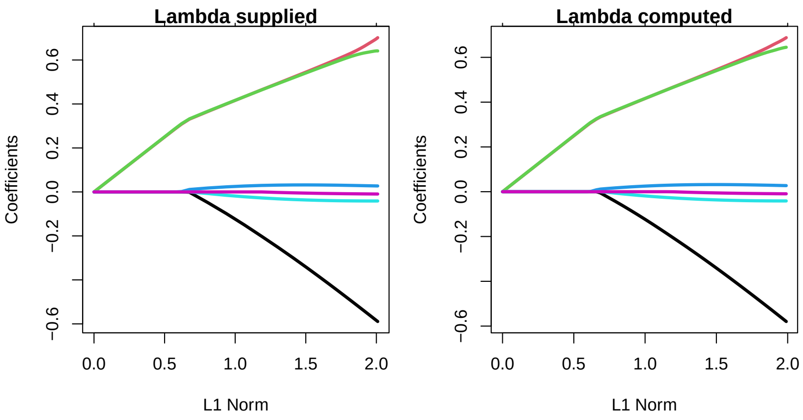
The following plot shows elastic net paths for the data set `smalldata.txt` of the website and $\alpha = 1, 0.66, 0.33, 0$. The plots on the left column have horizontal axis $||\hat{\boldsymbol{\beta}}||_1$ and although look roughly similar, when plotted against λ we see the gradual transition between lasso $\alpha = 1$ to ridge $\alpha = 0$.



α= 1
α= 2/3
α= 1/3
α= 0
Coefficients
L1 Norm
λ
4125

We apply the elastic net to the Credit data and $\alpha = 0.1$ which is close to ridge regression. We do two fits, one supplying λ and another without these values.

```
library(glmnet)
par(mar=c(4,4,1,1),mfrow=c(1,2))
EN1<-glmnet(x=DAT[Train,-c(7)],y=DAT[Train,7],lambda=rangelambda,alpha=0.1,
            intercept=FALSE)
EN2<-glmnet(x=DAT[Train,-c(7)],y=DAT[Train,7],alpha=0.1,intercept=FALSE)
plot(EN1,lwd=3,main="Lambda supplied")
plot(EN2,lwd=3,main="Lambda computed")
```



The elastic net penalty can be applied to any linear model, either regression or even for classification. Other recent developments in supervised learning include Least Angle Regression (LARS), Dantzig selector and smoothly clipped absolute deviation (SCAD).

5.3 Penalized likelihood

The two cases of ridge regression and lasso are instances of penalisation of (log) likelihood. Barring constants and without considering the problem of estimating the variance of errors, the (log) likelihood for the normal model of Equation 1 is $l(\boldsymbol{\beta}) = -||\mathbf{Y} - \mathbf{X}\beta||_2^2$. This likelihood is to be maximized as a function of $\boldsymbol{\beta}$.

The principle of penalised likelihood is to add a function $g(\boldsymbol{\beta})$ that penalizes for large values of the parameter vector $\boldsymbol{\beta}$. The penalised likelihood l_P is obtained by combining the likelihood $l(\boldsymbol{\beta})$ with the penalty function $g(\boldsymbol{\beta})$ so that

$$l_P(\boldsymbol{\beta}) = l(\boldsymbol{\beta}) - \lambda g(\boldsymbol{\beta}),$$

for a positive weight λ. For a range of values of λ, this quantity is to be maximised as a function of $\boldsymbol{\beta}$. Examples of penalties $g(\boldsymbol{\beta})$ are L_1 penalisation $g(\boldsymbol{\beta}) = ||\boldsymbol{\beta}||_1$, that is Manhattan or Lasso like penalty; and L_2 penalisation $g(\beta) = ||\beta||_2^2$ which is squared Euclidean or ridge style. These are penalisations that for increasing values of λ shrink model coefficients towards zero. It is also possible to use the elastic net penalty $g(\alpha, \boldsymbol{\beta})$ described earlier so that for a fixed value of α and values of λ the following criterion is to be maximized

$$l_P(\boldsymbol{\beta}) = l(\boldsymbol{\beta}) - \lambda g(\alpha, \boldsymbol{\beta}).$$

Finally, an alternative to elastic nets appears when for a given problem we must penalise departures from a known value $\boldsymbol{\beta}_0$, we could use a penality such as $g(\boldsymbol{\beta}) = ||\boldsymbol{\beta} - \boldsymbol{\beta}_0||_2^2$. Such a penalty would move the penalised likelihood to become close to **Bayesian** methodology.

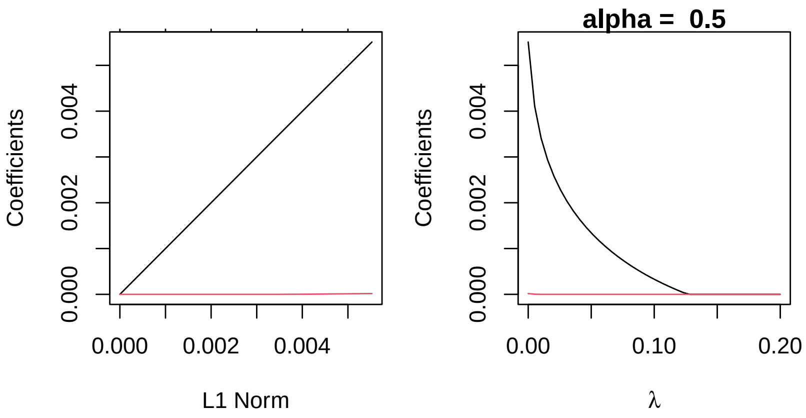
An implementation of penalised likelihood using elastic nets is in the `R` package `glmnet`. This implementation has the flexibility of working with the linear model of Equation 1 as well as the linear predictor of the form $\mathbf{X}\boldsymbol{\beta}$ in generalised linear models such as logistic or multinomial model.

We now present an example of likelihood penalised with elastic net using the `Default` data set seen earlier in the Module. The analysis uses the standard logistic regression, i.e. a Bernoulli (binomial) likelihood whose probabilities are modeled using the inverse logistic transformation of the linear predictor. With the `glmnet` package, we can train this model and build paths for the elastic net fit for these data. We use $\alpha = 0.5$ which is an intermediate value between lasso and ridge penalties and explore for a range of 40 values of λ between 0.2 and 0 (in that order). For the analysis we use the same partition 80:20 training/validation as earlier.

```r
library(glmnet)
alpha<-0.5;   lv<-seq(from=0.2,to=0.0,length.out=40) ## lambda values
ML1<-glmnet(x=as.matrix(Default[Train,-c(1:2)]),
            y = (Default[Train,1]=="Yes")*1,
            family = "binomial",lambda = lv,alpha=alpha)
```

We look at the paths of the trained model. They shrink to zero at $\lambda \approx 0.13$.

```
par(mar=c(4,4,1,1), mfrow=c(1,2));       plot(ML1);
plot(range(lv),range(ML1$beta),type="n", xlab=expression(lambda),
     ylab="Coefficients", main=paste("alpha = ",round(alpha,3)))
for(i in 1:2) lines(lv,ML1$beta[i,],col=i)
```

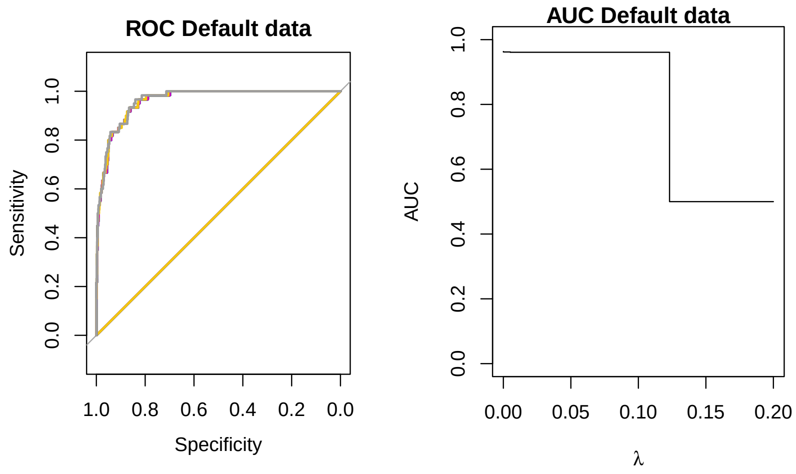


We now create predictions in variable `PL1` using the testing fold. They are allocated in a matrix whose number of rows is the size of the test fold; and the columns are indexed by the values of λ.

```
predict.glmnet(object = ML1, newx = as.matrix(Default[Test,-c(1:2)]),
                 type="response",)->PL1
dim(PL1)

## [1] 2000   40
```

We finish this analysis constructing a series of ROC curves, one for each of the models in the elastic net path. In each case we also compute the statistic AUC, which is plotted separately against λ.

```
library(pROC)
roc(response=default[Test],predictor=PL1[,1])->RR1
plot(RR1,col="black",main="ROC Default data") ## glm(default~balance+income)
for(i in 2:ncol(PL1)){
  roc(response=default[Test],predictor=PL1[,i])->RR1
  plot(RR1,col=i,main="ROC Default data",add=TRUE) ## glm(default~balance+income)
}
```



In this elastic net analysis of `Default` date we see two types of classifiers. For small values of λ we have very good classifiers with AUC over 0.95 and very minor variations amongst them. When λ is larger than about 0.13, the models behave like random classifiers with AUC=0.5. This threshold between the two patterns is clearly the shrinkage of the coefficient for `balance` to zero.

Exercise 35 (Extra) There is a close link between Principal Component Analysis and Ridge regression. Explore it through the following, recall the singular value decomposition of the data matrix $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$.

1. If $\hat{\boldsymbol{\beta}}^R$ is the ridge estimator defined in Equation 4, show that the ridge predictions are

$$\mathbf{X}\hat{\boldsymbol{\beta}}^R = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda} \mathbf{u}_j \mathbf{u}_j^T \mathbf{Y}.$$

2. Compare the ridge prediction against the least squares prediction $\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{U}\mathbf{U}^T\mathbf{Y}$ done in Exercise 15 and describe the influence of the factor $d_j^2/(d_j^2 + \lambda)$. Explain the role of λ in this factor.

3. Show that $\mathrm{df}(\lambda) \to p$ when $\lambda \to 0$ and that $\mathrm{df}(\lambda) \to 0$ when $\lambda \to \infty$.

4. (Extra) Show that $\hat{\boldsymbol{\beta}}^R \to 0$ when $\lambda \to \infty$.

Exercise 36 (Extra) (Hastie et al.) Consider the following augmented regression problem:

$$\left(\begin{array}{c|c} \mathbf{X} & \mathbf{Y} \\ \sqrt{\lambda}\mathbf{I} & \mathbf{0} \end{array} \right)$$

1. Carefully determine the dimensions of the elements involved in this regression problem. In particular, specify what the vector of coefficients should be.

2. Solve by least squares this augmented regression problem and show its relation to ridge regression.

Exercise 37 Consider the `diabetes` dataset, from the `R` package `lars`. This dataset has blood and other measurements in a group of diabetics.

1. Split the data into training and test data set and then perform ridge and lasso analyses for this data set. Compare the results obtained.

2. (Extra) Use the function `cv.lars` for repeated fit and validation analysis (cross-validation) with lasso.

3. (Extra) Explore the library `hierNet` which was developed for modelling with variable interactions within the lasso framework. Use it to fit the `diabetes` data.

Exercise 38 Explore the use of the library `glmnet` for the following machine learning analyses. Data will be partitioned as usual into training and testing sets before using penalized likelihood.

1. For the classification problem with the `Default` dataset. This analysis uses the logistic model with `binomial` option of the generalized linear model.

2. For analysing the `warpbreaks` data set. This is a dataset from the `R` package `datasets` in which the variable `breaks` is the number of warp breaks per loom, where a loom corresponds to a fixed length of yarn. There are two explanatory variables `wool` and `tension`. The analysis is to be carried out using the log-linear model associated with the `poisson` generalized linear model.

3. (Extra) Use the function `cv.glmnet` for repeated fit and validation analysis (cross-validation) in both data sets above.

4. (Extra) Use the library `hierNet` for fitting logistic model with variable interactions within lasso framework.

Important points and concepts of weeks ten to twelve:

1. The variants of regularization: ridge regression, lasso and penalized likelihood. Understand the differences between ridge regression and lasso.

2. Concerning the ridge trace and of the lasso path, understand the different versions of them and the computations involved.

3. The concept of elastic net penalization and its relation with the lasso and ridge penalizations.

4. Practice data analysis with `R`. Key functions for this analysis are `lars`, `ridge` (own) and `glmnet`.

6 Laboratory material R

MTH6101 Introduction to Machine Learning

Laboratory week two

The intention of this laboratory is to do a review of `R`, its functions and some packages. The tasks involved are matrix computations.

1. Open the package `R`. This can be done in a variety of ways. You could have downloaded it and installed it from `https://cran.r-project.org/` A popular version with interactive menus is `RStudio`, that can be downloaded from `https://rstudio.com/products/rstudio/download/` You can run `R` online, with `https://rdrr.io/snippets/` that allows running with libraries. You can access `RStudio` from `Appsanywhere` using the url `appsanywhere.qmul.ac.uk`.

2. Open a new R script file so that you will write your commands there. Use "File>New File>R Script" in `RStudio` and "File>New script" if you use the standard version of `R`. Remember to name this file and to save it in your own folder so that you do not lose your work.

3. Do the activities of Exercise 1 for the matrix seen in lectures, which is the matrix labelled "6". To use the library pracma remember to install it with the command `install.packages("pracma")`.

4. Repeat the activities for same exercise using the matrix labelled "7".

5. Do the activities of Exercise 3, using the matrix labelled "2".

6. In `RStudio`, open a new R markdown file with "File>New File>R Markdown" and select "pdf" output. Remember to erase everything but the header of this new file. Commands in `R` must be inside the brackets ` ```{r} ` and ` ``` `. To view the output you must compile the file using the "knit" button above the document, or the sequence `Ctrl+Shift+K`.

   To help your markdown tasks, have a look at the useful reference card `https://cran.r-project.org/web//packages/knitr/vignettes/knitr-refcard.pdf`

7. Using the code you have already run, experiment with the R markdown capability. Specifically, experiment by

   (a) hiding code from view while still executing it and showing the output, and

   (b) hiding both code and standard `R` output to then create output as part of the text in the markdown document.

MTH6101 Introduction to Machine Learning

Laboratory week two - Comments and partial results

The intention of this laboratory is to do a review of `R`, its functions and some packages. The tasks involved are matrix computations.

1. Open the package `R`. This can be done in a variety of ways. You could have downloaded it and installed it from `https://cran.r-project.org/` A popular version with interactive menus is `RStudio`, that can be downloaded from `https://rstudio.com/products/rstudio/download/` You can run `R` online, with `https://rdrr.io/snippets/` that allows running with libraries. You can access `RStudio` from `Appsanywhere` using the url `appsanywhere.qmul.ac.uk`.

2. Open a new R script file so that you will write your commands there. Use "File>New File>R Script" in `RStudio` and "File>New script" if you use the standard version of `R`. Remember to name this file and to save it in your own folder so that you do not lose your work.

   Comments: Students use `RStudio` so this should work and we would not need the standard `R` or the online version.
   In the worst case scenario (online `R` with limited libraries), students would only be able to do the parts of the lab that involve common `R` functions such as `eigen`, `svd` and matrix operations but nothing involving libraries or Markdown. However, these commands are the substantial part of the lab.

3. Do the activities of Exercise 1 for the matrix seen in lectures, which is the matrix labelled "6". To use the library pracma remember to install it with the command `install.packages("pracma")`.

   Comments: This exercise was done in lectures so students know the results. In any case, here is the `R` material, and the items in the list below are those of the said exercise.

   (a) Compute the eigenvalue decomposition of the matrix.

   ```r
   ## load and show the matrix
   A<-matrix(ncol=2,nrow=2,byrow=TRUE,c(0,1,2,1)); A
   ```

   ```
   ##      [,1] [,2]
   ## [1,]    0    1
   ## [2,]    2    1
   ```

```
E<-eigen(A) ## eigenvalue decomposition
E

## eigen() decomposition
## $values
## [1]  2 -1
##
## $vectors
##            [,1]       [,2]
## [1,] -0.4472136 -0.7071068
## [2,] -0.8944272  0.7071068

E$vectors

##            [,1]       [,2]
## [1,] -0.4472136 -0.7071068
## [2,] -0.8944272  0.7071068

E$values

## [1]  2 -1
```

(b) Checking the reproducing property

```
A%*%E$vectors[,1] ## For the first eigenvalue A\lambda_1

##            [,1]
## [1,] -0.8944272
## [2,] -1.7888544

E$vectors[,1]*E$values[1] ## v_1\lambda_1

## [1] -0.8944272 -1.7888544

A%*%E$vectors[,2] ## For the second eigenvalue A\lambda_2

##            [,1]
## [1,]  0.7071068
## [2,] -0.7071068

E$vectors[,2]*E$values[2] ## v_2\lambda_2

## [1]  0.7071068 -0.7071068
```

(c) Computing the characteristic polynomial

Normally students should be able to install the library.

```
library(pracma) ## the library must be loaded
charpoly(a=A)->CP
CP

## [1]  1 -1 -2
```

This is precisely the same polynomial seen in lectures, but only the coefficients are given. We use the function **poly2str** to put it in a readable form.

```
poly2str(p=CP)

## [1] "1*x^2 - 1*x - 2"
```

Then compute the roots of this polynomial, which are precisely the eigenvalues given and used in lectures.

```
polyroots(p=CP)

##   root mult
## 1    2    1
## 2   -1    1
```

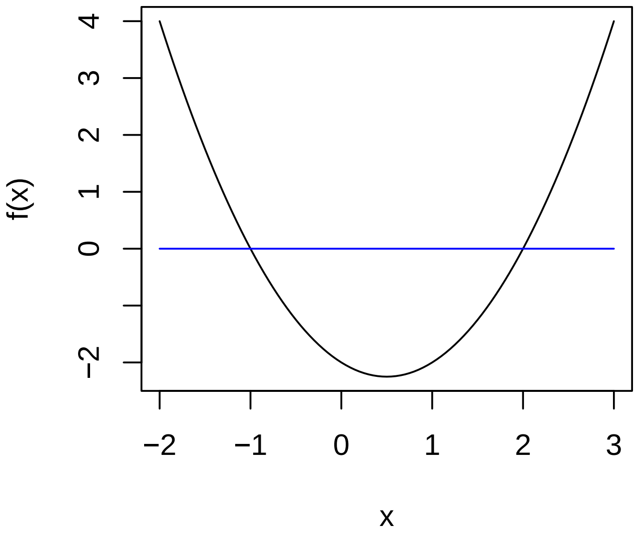
(d) Plot the characteristic polynomial.

Simple evaluation of the characteristic polynomial. Can you check by hand?

```
polyval(p=CP,x=2);

## [1] 0
```

Build a function and plot the characteristic polynomial. The horizontal line shows the zeroes of it (roots of the characteristic polynomial, i.e. eigenvalues of the matrix) which were computed earlier.

```r
f<-function(x) polyval(p=CP,x=x)
curve(f,from=-2,to=3)
lines(c(-2,3),c(0,0),col="blue")
```



4. Repeat the activities for same exercise using the matrix labelled “7”.

   Comments: This computation uses the same commands as the first exercise and the only change is that the matrix involved is a matrix of size 3×3 thus there will be a third eigenvalue involved.

   (a) Compute the eigenvalue decomposition of the matrix.

```
## load and show the matrix
A<-matrix(ncol=3,nrow=3,byrow=TRUE,c(-1,-2,2,1,2,-4,1,1,-3)); A

##      [,1] [,2] [,3]
## [1,]   -1   -2    2
## [2,]    1    2   -4
## [3,]    1    1   -3
```

```
E<-eigen(A) ## eigenvalue decomposition
E

## eigen() decomposition
## $values
## [1] -2 -1  1
##
## $vectors
##               [,1]       [,2]          [,3]
## [1,]  1.665335e-16 -0.5773503  7.071068e-01
## [2,] -7.071068e-01 -0.5773503 -7.071068e-01
## [3,] -7.071068e-01 -0.5773503 -2.775558e-16

E$vectors

##               [,1]       [,2]          [,3]
## [1,]  1.665335e-16 -0.5773503  7.071068e-01
## [2,] -7.071068e-01 -0.5773503 -7.071068e-01
## [3,] -7.071068e-01 -0.5773503 -2.775558e-16

E$values

## [1] -2 -1  1
```

(b) Checking the reproducing property

```
A%*%E$vectors[,1] ## For the first eigenvalue A\lambda_1

##               [,1]
## [1,] -8.881784e-16
## [2,]  1.414214e+00
## [3,]  1.414214e+00

E$vectors[,1]*E$values[1] ## v_1\lambda_1

## [1] -3.330669e-16  1.414214e+00  1.414214e+00

A%*%E$vectors[,2] ## For the second eigenvalue A\lambda_2

##           [,1]
## [1,] 0.5773503
## [2,] 0.5773503
## [3,] 0.5773503

E$vectors[,2]*E$values[2] ## v_2\lambda_2

## [1] 0.5773503 0.5773503 0.5773503

A%*%E$vectors[,3] ## For the second eigenvalue A\lambda_3

##               [,1]
## [1,]  7.071068e-01
## [2,] -7.071068e-01
## [3,]  1.054712e-15

E$vectors[,3]*E$values[3] ## v_3\lambda_3

## [1]  7.071068e-01 -7.071068e-01 -2.775558e-16
```

(c) Computing the characteristic polynomial

```
library(pracma)
charpoly(a=A)->CP
CP

## [1]  1  2 -1 -2
```

Here only the coefficients are given. We use the function **`poly2str`** to put it in a readable form.

```
poly2str(p=CP)

## [1] "1*x^3 + 2*x^2 - 1*x - 2"
```

Then compute the roots of this polynomial, which are precisely the eigenvalues computed earlier.

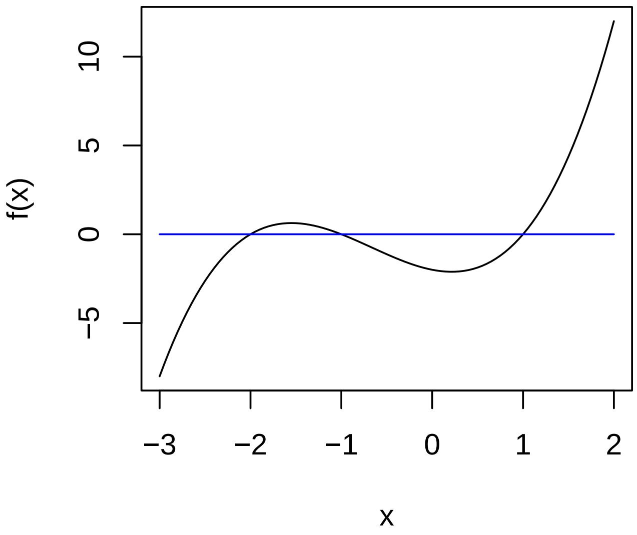
```
polyroots(p=CP)

##   root mult
## 1   -2    1
## 2    1    1
## 3   -1    1
```

(d) Plot the characteristic polynomial.
Simple evaluation of the characteristic polynomial. Can you check by hand?

```
polyval(p=CP,x=-2);

## [1] 0
```

Build a function and plot the characteristic polynomial. The horizontal line shows the zeroes of it (roots of the characteristic polynomial, i.e. eigenvalues of the matrix) which were computed earlier.

```
f<-function(x) polyval(p=CP,x=x)
curve(f,from=-3,to=2)
lines(c(-3,2),c(0,0),col="blue")
```



5. Do the activities of Exercise 3, using the matrix labelled “2”.

   Comments: In lectures we have done the Karhunen-Loeve decomposition which is precisely the **same** eigenvalue computation. The output of `R` is given below, following the items of said exercise.

   (a) Karhunen-Loeve decomposition

```
## Load and show the matrix
A<-matrix(ncol=3,nrow=3,byrow=TRUE,c(1,0.13333,0.26667,0.13333,
  0.66667,-0.4,0.26667,-0.4,0.73333))
A

##          [,1]     [,2]     [,3]
## [1,] 1.00000  0.13333  0.26667
## [2,] 0.13333  0.66667 -0.40000
## [3,] 0.26667 -0.40000  0.73333

## Karhunen-Loeve decomposition
eigen(A)->E
E

## eigen() decomposition
## $values
## [1] 1.2000033 0.9999967 0.2000000
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,]  0.6666678 -0.6666653  0.3333339
## [2,] -0.3333356 -0.6666694 -0.6666628
## [3,]  0.6666644  0.3333306 -0.6666703

E$vectors

##            [,1]       [,2]       [,3]
## [1,]  0.6666678 -0.6666653  0.3333339
## [2,] -0.3333356 -0.6666694 -0.6666628
## [3,]  0.6666644  0.3333306 -0.6666703

E$values

## [1] 1.2000033 0.9999967 0.2000000
```

Verifying the KL decomposition.

```
E$vectors%*%t(E$vectors) ## orthogonal eigenvectors

##      [,1]         [,2]         [,3]
## [1,]    1 0.000000e+00 0.000000e+00
## [2,]    0 1.000000e+00 2.775558e-16
## [3,]    0 2.775558e-16 1.000000e+00

E$vectors%*%diag(E$values)%*%t(E$vectors) ## KL decomposition

##         [,1]     [,2]     [,3]
## [1,] 1.00000  0.13333  0.26667
## [2,] 0.13333  0.66667 -0.40000
## [3,] 0.26667 -0.40000  0.73333
```

(b) A series of partial sums of the KL decomposition.
Here the only potential difficulty is to use the correct matrix operations.

```
E$vectors[,1]%*%t(E$vectors[,1])*E$values[1]->M1 ## a_1*a_1^T * lambda_1
E$vectors[,2]%*%t(E$vectors[,2])*E$values[2]->M2 ## a_2*a_2^T * lambda_2
E$vectors[,3]%*%t(E$vectors[,3])*E$values[3]->M3 ## a_3*a_3^T * lambda_3
```

The partial sums. The last one is the original matrix.

```
M1; M1+M2; M1+M2+M3

##            [,1]       [,2]       [,3]
## [1,]  0.5333366 -0.2666696  0.5333339
## [2,] -0.2666696  0.1333355 -0.2666683
## [3,]  0.5333339 -0.2666683  0.5333313
##           [,1]       [,2]       [,3]
## [1,] 0.9777777  0.1777743  0.3111148
## [2,] 0.1777743  0.5777821 -0.4888889
## [3,] 0.3111148 -0.4888889  0.6444401
##         [,1]     [,2]     [,3]
## [1,] 1.00000  0.13333  0.26667
## [2,] 0.13333  0.66667 -0.40000
## [3,] 0.26667 -0.40000  0.73333
```

(c) Partial sums of eigenvector products.

```
E$vectors[,1]%*%t(E$vectors[,1])->B1 ## a_1*a_1^T
E$vectors[,2]%*%t(E$vectors[,2])->B2 ## a_2*a_2^T
E$vectors[,3]%*%t(E$vectors[,3])->B3 ## a_3*a_3^T
```

The partial sums. The last sum is the identity matrix. Why?

```
B1;

##            [,1]       [,2]       [,3]
## [1,]  0.4444459 -0.2222241  0.4444437
## [2,] -0.2222241  0.1111126 -0.2222230
## [3,]  0.4444437 -0.2222230  0.4444415

B1+B2;

##           [,1]       [,2]       [,3]
## [1,] 0.8888885  0.2222213  0.2222238
## [2,] 0.2222213  0.5555607 -0.4444443
## [3,] 0.2222238 -0.4444443  0.5555507

B1+B2+B3

##      [,1]         [,2]         [,3]
## [1,]    1 0.000000e+00 0.000000e+00
## [2,]    0 1.000000e+00 2.775558e-16
## [3,]    0 2.775558e-16 1.000000e+00
```

6. In `RStudio`, open a new R markdown file with “File>New File>R Markdown” and select “pdf” output. Remember to erase everything but the header of this new file. Commands in `R` must be inside the brackets `‘‘‘{r}` and `‘‘‘`. To view the output you must compile the file using the “knit” button above the document, or the sequence `Ctrl+Shift+K`.

   To help your markdown tasks, have a look at the useful reference card `https://cran.r-project.org/web//packages/knitr/vignettes/knitr-refcard.pdf`

7. Using the code you have already run, experiment with the R markdown capability. Specifically, experiment by

   (a) hiding code from view while still executing it and showing the output, and

   (b) hiding both code and standard `R` output to then create output as part of the text in the markdown document.

Comments: Assuming Markdown works in `RStudio` and is able to compile, the task here simply reuses all previous `R` code. Of course we are not sure if this compilation will happen but hopefully will do, assuming you have the required libraries and packages. Your the markdown script file should be saved before compiling it.
For hiding instructions, use the parameter `echo=FALSE` so any `R` code is bracketed between `‘‘‘{r,echo=FALSE}` and `‘‘‘`.
For hiding code and output, do as above with `echo=FALSE` and do not print output. Then assuming you want to exhibit variable `x` inside the text, write in the normal text `‘r x‘` .

MTH6101 Introduction to Machine Learning

Laboratory week three

The intention of this laboratory is to do Principal Component Analysis (PCA) of the air quality data of Exercise 9. The tasks involved are loading, preparing the data and analysing it with the Karhunen-Loeve decomposition.

1. Open `RStudio` and create and save a new `R` script file for your commands. If you have `markdown` on your own computer, you are welcomed to use it.

2. Load the data using the `R` command `data(airquality)` and examine the data. To this end, see the data using `airquality`; see what the entries of the variables are and make notes. Also describe the data using commands such as `pairs`, `summary` and `var`.

3. As part of this initial analysis, you will have noted that the first two variables of the data set have **missing values**. For the rest of the analysis, these need to be removed. Use the command `complete.cases(airquality)` to determine which are the values to be removed and **create** a new variable by allocating `airquality[complete.cases(airquality),]` to this new variable, say `X`. Examine `X` and make sure that you have removed missing values.

4. Your analysis will not use the variables `day` nor `month` which you will remove by `X<-X[,-c(5:6)]`. Then center by `X<-scale(x=X,center=TRUE,scale=FALSE)`.

5. **The data is now ready for PCA**. Compute the variance-covariance matrix of `X` and then do the Karhunen-Loeve decomposition of it with by the now well known commands `var` and `eigen`. Store each result in a new variable.

6. Write the eigenvalues you just obtained and compute and interpret the proportional contribution of each eigenvalue to the total variability (sum of eigenvalues which of course equals the trace of the variance covariance matrix).

7. Do a pairs plot of the Principal Components and compare this with your previous use of the command `pairs` for the data `X`. Recall that the Principal Components are the rotated data, computed as `X%*%E$vectors`, assuming that the variable `E` has the results of the K-L decomposition.

8. Redo all from step 4 by scaling the data as well with `scale=TRUE)`. Compare with the previous analysis.

MTH6101 Introduction to Machine Learning

Laboratory week three - Comments and partial results

The intention of this laboratory is to do Principal Component Analysis (PCA) of the air quality data of Exercise 9. The tasks involved are loading, preparing the data and analysing it with the Karhunen-Loeve decomposition.

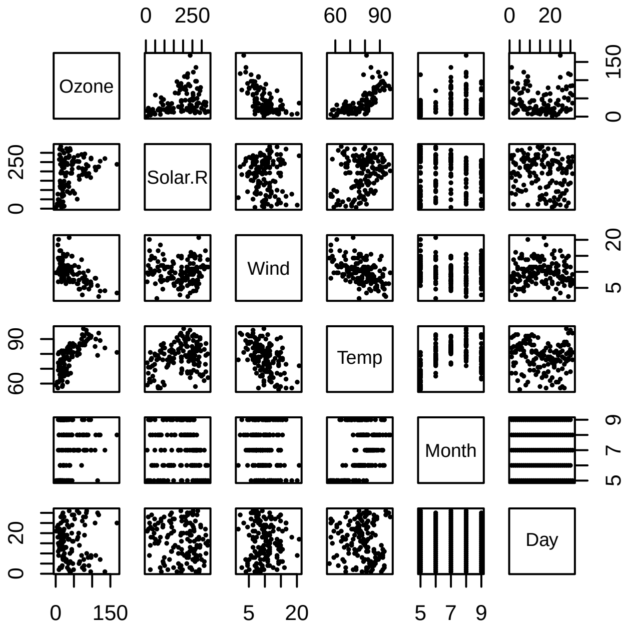
1. Open `RStudio` and create and save a new `R` script file for your commands. If you have `markdown` on your own computer, you are welcomed to use it.

2. Load the data using the `R` command `data(airquality)` and examine the data. To this end, see the data using `airquality`; see what the entries of the variables are and make notes. Also describe the data using commands such as `pairs`, `summary` and `var`.

```
data(airquality); summary(airquality)

##      Ozone           Solar.R           Wind             Temp
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00
##  NA's   :37       NA's   :7
##      Month            Day
##  Min.   :5.000   Min.   : 1.0
##  1st Qu.:6.000   1st Qu.: 8.0
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
##
```

Note missing entries shown at the bottom of summary columns. We plot the data and compute the variance-covariance matrix.

```
pairs(airquality,pch=16,cex=0.5);   var(airquality)
```



```
##              Ozone Solar.R        Wind       Temp      Month        Day
## Ozone          NA      NA          NA         NA         NA         NA
## Solar.R        NA      NA          NA         NA         NA         NA
## Wind           NA      NA  12.4115385 -15.272136 -0.8897532  0.8488519
## Temp           NA      NA -15.2721362  89.591331  5.6439628 -10.9574303
## Month          NA      NA  -0.8897532   5.643963  2.0065359  -0.0999742
## Day            NA      NA   0.8488519 -10.957430 -0.0999742  78.5797214
```

The scatterplot shows clearly the structure of variables `Month` and `Day` which were used when collecting the samples. Note also in the scatterplot the fluctuation of variables with respect to `month` (fifth column of the plot matrix). When computing the variance-covariance matrix it is evident the problem of missing data values.

3. As part of this initial analysis, you will have noted that the first two variables of the data set have **missing values**. For the rest of the analysis, these need to be removed. Use the command `complete.cases(airquality)` to determine which are the values to be removed and **create** a new variable by allocating `airquality[complete.cases(airquality),]` to this new variable, say `X`. Examine `X` and make sure that you have removed missing values.

4. Your analysis will not use the variables `day` nor `month` which you will remove by `X<-X[,-c(5:6)]`. Then center by `X<-scale(x=X,center=TRUE,scale=FALSE)`.

   ```
   sum(complete.cases(airquality))/nrow(airquality)*100

   ## [1] 72.54902

   X<-airquality[complete.cases(airquality),]
   X<-X[,-c(5:6)];  X<-scale(x=X,center=TRUE,scale=FALSE)
   ```

   The percentage shown is of complete data before removal of missing cases.

5. **The data is now ready for PCA**. Compute the variance-covariance matrix of `X` and then do the Karhunen-Loeve decomposition of it with by the now well known commands `var` and `eigen`. Store each result in a new variable.

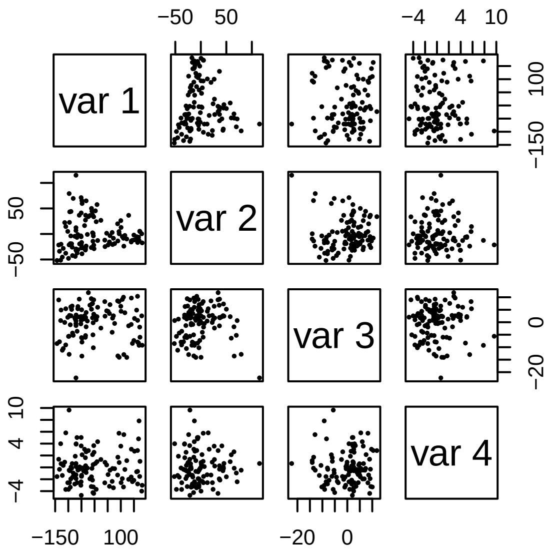
6. Write the eigenvalues you just obtained and compute and interpret the proportional contribution of each eigenvalue to the total variability (sum of eigenvalues which of course equals the trace of the variance covariance matrix).

   ```
   VX<-var(X);   E<-eigen(VX);    E$values/sum(E$values)*100

   ## [1] 88.98078028 10.47105443  0.46819497  0.07997032
   ```

   The first eigenvalue accounts for almost 90% of the total variability in the dataset, while the first two account for well above 99% of the total variability.

7. Do a pairs plot of the Principal Components and compare this with your previous use of the command `pairs` for the data `X`. Recall that the Principal Components are the rotated data, computed as `X%*%E$vectors`, assuming that the variable `E` has the results of the K-L decomposition.

```r
pairs(X%*%E$vectors,pch=16,cex=0.5)
```



Note that the scatterplot of the first two components seems like a rotated version of the first two variables, so we conjecture that the coefficients of these are bigger than the rest for this first compontent. We look at the coefficients (first column below).

```
E$vectors
```

```
##               [,1]        [,2]         [,3]         [,4]
## [1,] -0.143014512  0.96704882 -0.202729051  0.057134580
## [2,] -0.989119298 -0.14698335 -0.004561518 -0.004254707
## [3,]  0.006117191 -0.06845434 -0.050046167  0.996379428
## [4,] -0.033947672  0.19628163  0.977944531  0.062813790
```

Indeed the first component is almost entirely due to the second variable `Solar.R`, while the second component is mainly due to the first variable `Ozone`.

In all this PCA, note the importance of the second variable, seen in the diagonal of `VX`:

```
diag(VX)
```

```
##      Ozone    Solar.R       Wind       Temp
## 1107.29009 8308.74218   12.65732   90.82031
```

8. Redo all from step 4 by scaling the data as well with `scale=TRUE)`. Compare with the previous analysis.

```
X<-airquality[complete.cases(airquality),]
X<-X[,-c(5:6)];  X<-scale(x=X,center=TRUE,scale=TRUE)
VX<-var(X);   E<-eigen(VX);    E$values/sum(E$values)*100
```

```
## [1] 58.99747 22.36691 11.89375  6.74188
```

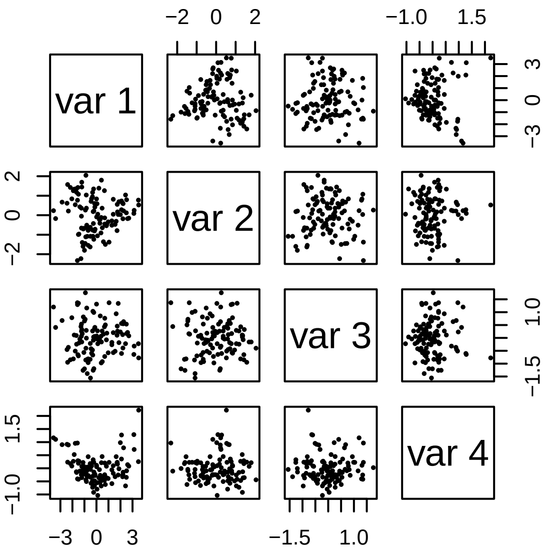
Note the sharp contrast with the first analysis. The first component only accounts for 59% of the total variability, while the first two account for around 80% of it. We look at the coefficients

```
E$vectors
```

```
##             [,1]        [,2]       [,3]       [,4]
## [1,]  0.5890040  0.06279119  0.1146414  0.7974891
## [2,]  0.3169087 -0.89844744 -0.2777312 -0.1233953
## [3,] -0.4970366 -0.43021767  0.6905372  0.3017048
## [4,]  0.5528090  0.06133702  0.6579370 -0.5076996
```

The first component is a comparison between the third variable `Wind` and the rest, while the second component is a weighted average of the second and third variables `Solar.R` and `Wind`.

```r
pairs(X%*%E$vectors,pch=16,cex=0.5)
```



The strong triangular shape of the initial scatterplot of the raw data is gone in the scatterplot of principal components above. The scatterplot of the components may also be useful to detect **potential outliers** or suggest **groups of individuals**, none of which are evident in the current case.

Conclusions obtained from PCA of non-scaled data do not extend to PCA of scaled data.

MTH6101 Introduction to Machine Learning

Laboratory week four

The intention of this laboratory is to do Principal Component Analysis (PCA) of the olive oil data set. We mostly use the recommended function for PCA which is `prcomp`.

1. Open `RStudio` and create and save a new `R` script file for your commands. If you have `markdown` on your own computer, you are welcomed to use it.

2. In order to load the data, install the library `pdfCluster`. Once installed, load the library and load the data using the `R` command `data(oliveoil)`. Examine the data using commands such as `pairs` and `summary`. Also read the help of the library to find out what the data is about.

3. This dataset is complete in the sense that it has **no** missing values and can be used straightaway. Your analysis will **not** use the first two categorical variables `macro.area` and `region` and thus you will remove them by `X<-oliveoil[,-c(1:2)]`. **The data is now ready for PCA**.

4. Center the data but do not scale it, and compute and examine the variance-covariance matrix, with special emphasis in the diagonal so that you can try and predict what will happen with PCA.

5. Do PCA using the function `prcomp` with these data (use parameters in this function to center but not scale the data). Print a summary of the analysis, select a number of components and interpret them. Do a biplot and examine what is being plotted.

6. Repeat all that you did from 4 but now with the data centered and scaled.

7. Recall the relation seen in lectures $\mathbf{\Lambda} = \frac{1}{n-1}\mathbf{D}^2$ between eigenvalues of the K-L expansion of $\mathbf{\Sigma}$ and eigenvalues of svd of the data $\mathbf{X}$. Numerically check that the relation holds for both analyses you did in 4 and 6.

MTH6101 Introduction to Machine Learning

Laboratory week four - Comments and results

The intention of this laboratory is to do Principal Component Analysis (PCA) of the olive oil data set. We mostly use the recommended function for PCA which is `prcomp`.

1. Open `RStudio` and create and save a new `R` script file for your commands. If you have `markdown` on your own computer, you are welcomed to use it.

2. In order to load the data, install the library `pdfCluster`. Once installed, load the library and load the data using the `R` command `data(oliveoil)`. Examine the data using commands such as `pairs` and `summary`. Also read the help of the library to find out what the data is about.

   Comments: In the past it has always been possible to install libraries so this should be the first approach. In case it is not possible to install the library `pdfCluster`, I have loaded the data in `qmplus` so students would have to download and then use the command `read.table`.

```
library(pdfCluster)
data(oliveoil)
summary(oliveoil)
```

```
##         macro.area                region        palmitic      palmitoleic
##  South       :323   Apulia.south    :206   Min.   : 610   Min.   : 15.00
##  Sardinia    : 98   Sardinia.inland: 65   1st Qu.:1095   1st Qu.: 87.75
##  Centre.North:151   Calabria        : 56   Median :1201   Median :110.00
##                     Umbria          : 51   Mean   :1232   Mean   :126.09
##                     Liguria.east    : 50   3rd Qu.:1360   3rd Qu.:169.25
##                     Liguria.west    : 50   Max.   :1753   Max.   :280.00
##                     (Other)         : 94
##     stearic          oleic         linoleic         linolenic
##  Min.   :152.0   Min.   :6300   Min.   : 448.0   Min.   : 0.00
##  1st Qu.:205.0   1st Qu.:7000   1st Qu.: 770.8   1st Qu.:26.00
##  Median :223.0   Median :7302   Median :1030.0   Median :33.00
##  Mean   :228.9   Mean   :7312   Mean   : 980.5   Mean   :31.89
##  3rd Qu.:249.0   3rd Qu.:7680   3rd Qu.:1180.8   3rd Qu.:40.25
##  Max.   :375.0   Max.   :8410   Max.   :1470.0   Max.   :74.00
```

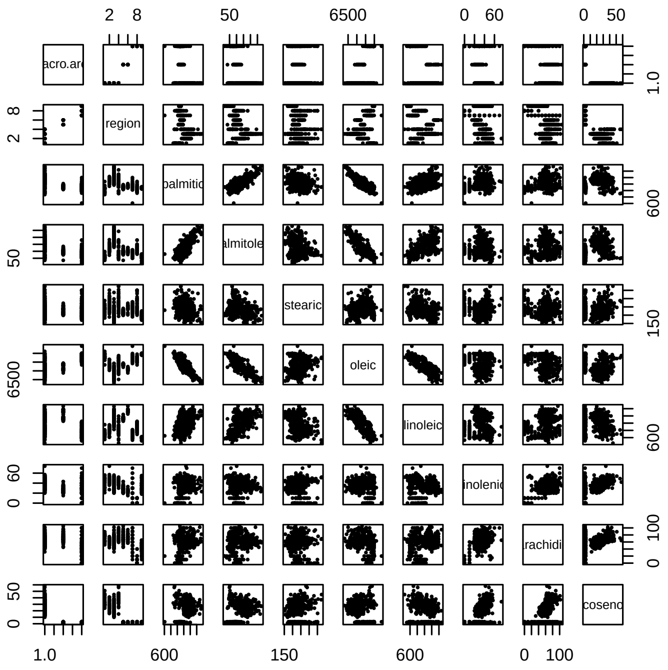
```
##
##    arachidic       eicosenoic
##  Min.   :  0.0   Min.   : 1.00
##  1st Qu.: 50.0   1st Qu.: 2.00
##  Median : 61.0   Median :17.00
##  Mean   : 58.1   Mean   :16.28
##  3rd Qu.: 70.0   3rd Qu.:28.00
##  Max.   :105.0   Max.   :58.00
##
```

```
apply(oliveoil,2,range)
```

```
##      macro.area     region          palmitic palmitoleic stearic oleic  lin
## [1,] "Centre.North" "Apulia.north" " 610"    " 15"       "152"   "6300" " 4
## [2,] "South"        "Umbria"       "1753"    "280"       "375"   "8410" "14
##      linolenic arachidic eicosenoic
## [1,] " 0"      "  0"     " 1"
## [2,] "74"      "105"     "58"
```

It is likely that the huge range of variable `oleic` will play an important role in PCA with centered but not scaled data. The other variables that have big ranges are `linoleic` and `palmitic`.

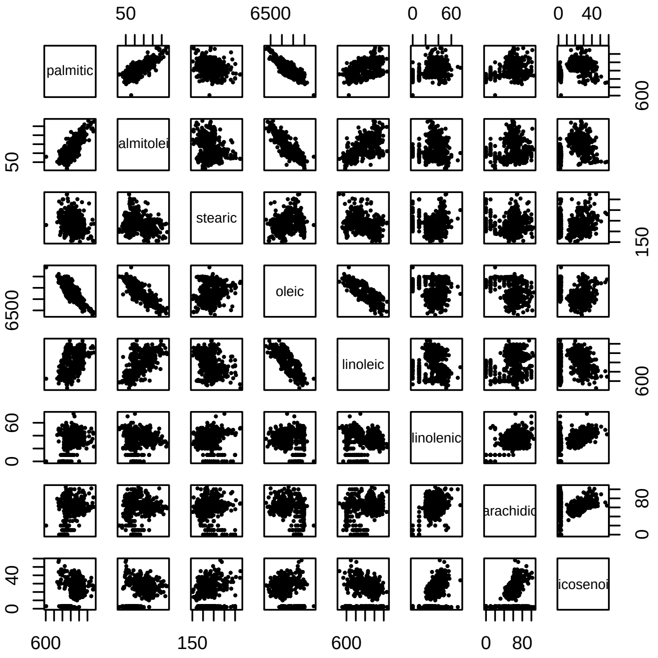
```
pairs(oliveoil,pch=16,cex=0.5)
```



Often details are hard to look at in a `pairs` plot with too many variables.

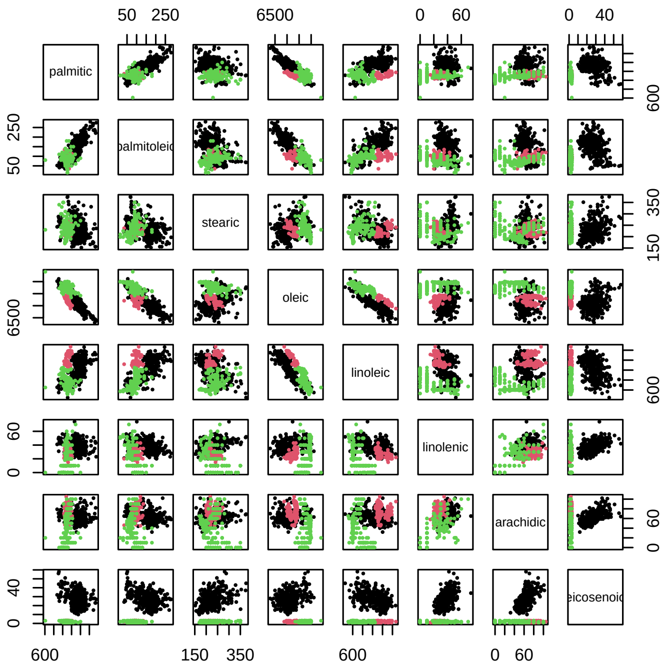
3. This dataset is complete in the sense that it has **no** missing values and can be used straightaway. Your analysis will **not** use the first two categorical variables `macro.area` and `region` and thus you will remove them by `X<-oliveoil[,-c(1:2)]`. **The data is now ready for PCA**.

```
X<-oliveoil[,-c(1:2)];     pairs(X,pch=16,cex=0.5)
```

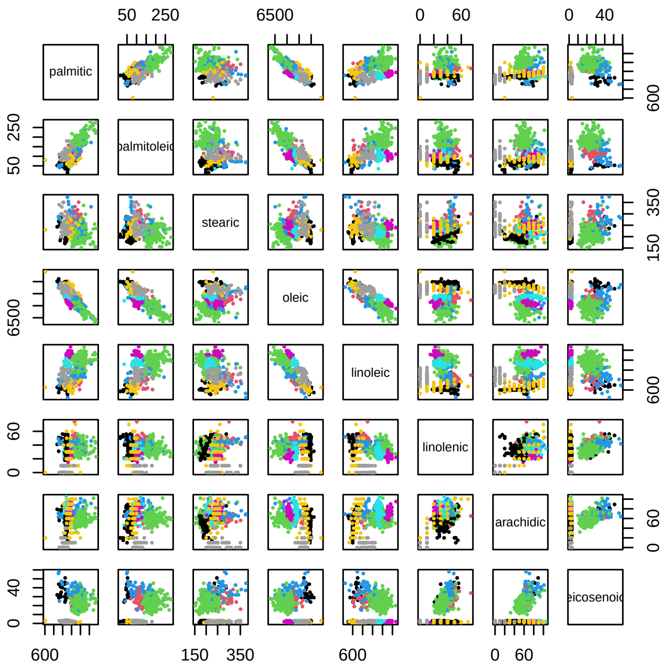


In the `pairs` plot we see the interesting lattice pattern involving the last three variables.Two other `pairs` plots are generated by coloring the scatterplot according to the variables left out. What can you deduce from them?

```
pairs(X,pch=16,cex=0.5,col=oliveoil$macro.area)
```



```r
pairs(X,pch=16,cex=0.5,col=oliveoil$region)
```



4. Center the data but do not scale it, and compute and examine the variance-covariance matrix, with special emphasis in the diagonal so that you can try and predict what will happen with PCA.

```
X<-scale(x=oliveoil[,-c(1:2)],center=TRUE,scale=FALSE)
round(var(X),5)
```

```
##                 palmitic  palmitoleic      stearic      oleic    linoleic
## palmitic      28423.3515   7395.22412  -1055.56204 -57287.524  18857.6903
## palmitoleic    7395.2241   2755.65832   -428.57396 -18159.281   7922.9991
## stearic       -1055.5620   -428.57396   1350.19025   1693.924  -1764.8517
## oleic        -57287.5241 -18159.28092   1693.92402 164681.937 -83782.2136
## linoleic      18857.6903   7922.99911  -1764.85174 -83782.214  58951.4616
## linolenic       698.1829     63.38887      9.01468  -1148.198   -180.8620
## arachidic       847.9326     98.85589    -33.17244  -2860.495   1128.4771
## eicosenoic     1191.8015    307.80526     72.64391  -2424.055    304.4816
##                linolenic   arachidic   eicosenoic
## palmitic       698.18291   847.93256   1191.80150
## palmitoleic     63.38887    98.85589    307.80526
## stearic          9.01468   -33.17244     72.64391
## oleic        -1148.19809 -2860.49545  -2424.05511
## linoleic      -180.86201  1128.47712    304.48161
## linolenic      168.18711   177.20362    105.62524
## arachidic      177.20362   485.33190    101.97064
## eicosenoic     105.62524   101.97064    198.33920
```

```
round(100*diag(var(X))/sum(diag(var(X))),5)
```

```
##    palmitic palmitoleic     stearic       oleic    linoleic   linolenic
##    11.05905     1.07218     0.52534    64.07497    22.93702     0.06544
##   arachidic  eicosenoic
##     0.18883     0.07717
```

Given the big variances, we expect that the first PCs will feature variables `oleic` then `linoleic` then `palmitic`.

5. Do PCA using the function **prcomp** with these data (use parameters in this function to center but not scale the data). Print a summary of the analysis, select a number of components and interpret them.

```
MM<-prcomp(x = X, center = TRUE, scale = FALSE);   summary(MM)

## Importance of components:
##                             PC1       PC2      PC3      PC4      PC5      P
## Standard deviation     480.150 150.96029 45.43418 27.54674 24.80339 11.980
## Proportion of Variance   0.897   0.08867  0.00803  0.00295  0.00239  0.000
## Cumulative Proportion    0.897   0.98568  0.99371  0.99666  0.99905  0.999
##                           PC7     PC8
## Standard deviation     7.1453 6.98180
## Proportion of Variance 0.0002 0.00019
## Cumulative Proportion  0.9998 1.00000
```

A single component is needed to account for over 80% of the total variability in the data, and two components already explain over 95%. Let us look at PC loadings

```
MM$rotation

##                      PC1          PC2         PC3         PC4        PC5
## palmitic     0.284167992  0.637208452 -0.45062836 -0.03857271 -0.4524509
## palmitoleic  0.092012578  0.094554974 -0.16460885 -0.57386935  0.6690989
## stearic     -0.011151773  0.014774824  0.72398889 -0.39748727 -0.4050560
## oleic       -0.842808624 -0.168763310 -0.33652056 -0.09246819 -0.1991783
## linoleic     0.447210266 -0.743751915 -0.30400153 -0.06643083 -0.2472581
## linolenic    0.004751237  0.034724051  0.08433954  0.29315488  0.1110979
## arachidic    0.013770009  0.009109222  0.14165474  0.63629076  0.1923104
## eicosenoic   0.011058482  0.043240557  0.11329544  0.08614438  0.1827288
##                    PC6        PC7         PC8
## palmitic    -0.1462473  0.2384073  0.16038084
## palmitoleic -0.3254595  0.1561341  0.21948984
## stearic     -0.2542609  0.2132920  0.20805981
## oleic       -0.1406756  0.2262900  0.16949136
## linoleic    -0.1066613  0.2203788  0.17007665
## linolenic    0.2156815 -0.1583158  0.90652832
## arachidic   -0.6648742  0.3078877 -0.03097674
## eicosenoic   0.5369327  0.8084912 -0.04905891
```

The first component is a comparison between `oleic` and a weighted average of variables `linoleic` and `palmitic`. In this comparison, the variable that has the highest impact is `oleic`. The second component is a comparison between `palmitic` and a weighted average of `oleic` and `linoleic`. In this comparison the (opposing) weights of `palmitic` and `linoleic` are of smilar magnitude.

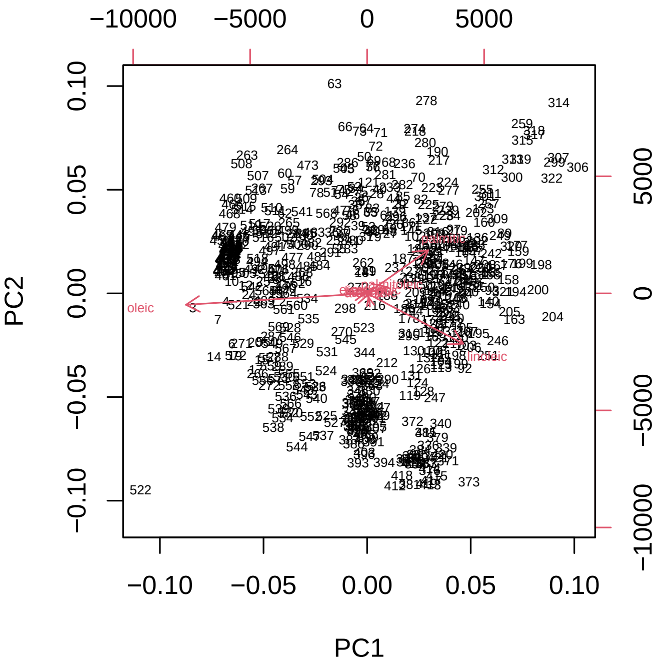
Comments: For interpretation, rounding the PC loadings can help in removing those variables with very small coefficients, see next.

```
round(MM$rotation,1)
```

```
##              PC1  PC2  PC3  PC4  PC5  PC6  PC7 PC8
## palmitic     0.3  0.6 -0.5  0.0 -0.5 -0.1  0.2 0.2
## palmitoleic  0.1  0.1 -0.2 -0.6  0.7 -0.3  0.2 0.2
## stearic      0.0  0.0  0.7 -0.4 -0.4 -0.3  0.2 0.2
## oleic       -0.8 -0.2 -0.3 -0.1 -0.2 -0.1  0.2 0.2
## linoleic     0.4 -0.7 -0.3 -0.1 -0.2 -0.1  0.2 0.2
## linolenic    0.0  0.0  0.1  0.3  0.1  0.2 -0.2 0.9
## arachidic    0.0  0.0  0.1  0.6  0.2 -0.7  0.3 0.0
## eicosenoic   0.0  0.0  0.1  0.1  0.2  0.5  0.8 0.0
```

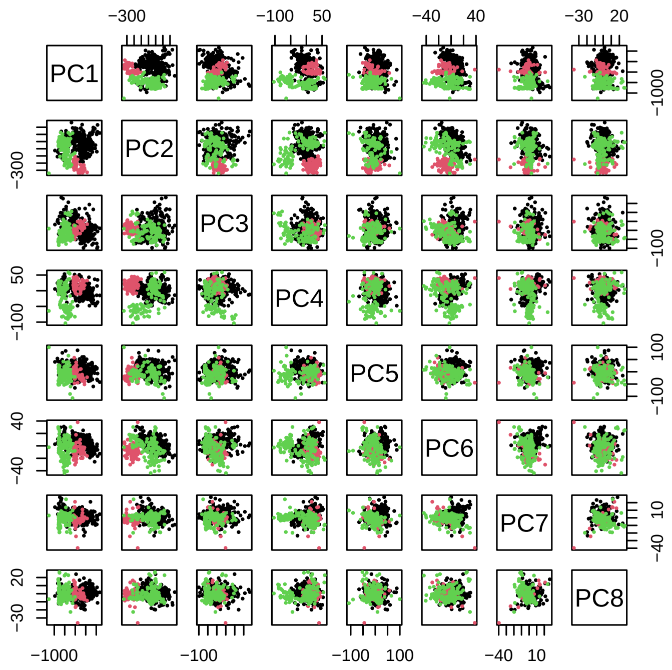
Do a biplot and examine what is being plotted.

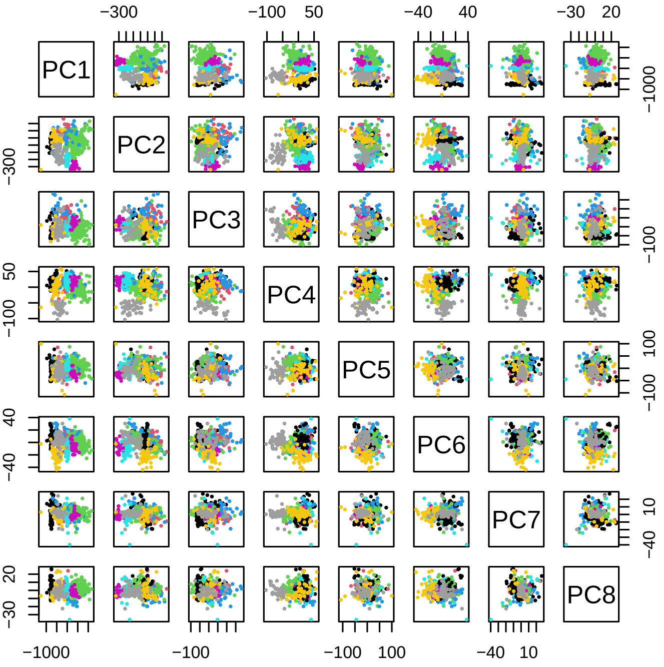
```
biplot(MM,cex=0.5)
```



The biplot shows simultaneously the PC scores for the first two principal components, and in a double axis it has the PC loadings for variables in these first two principal components. The interpretation of arrows is exactly the same as the one for PC loadings given earlier, while plotting the scores allows us to examine which induviduals (measurements as rows of the data) score highly (or lowly) on variables.

As an extra, we plot the PC scores using as colors the variables `macro.area` and `region`.




PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
−300
−100
50
−40
40
−30
20
−1000
−100
100
10
−1000
−100
−100
100
−40
10

6. Repeat all that you did from 4 but now with the data centered and scaled.

   First the variance-covariance matrix.

```
X<-scale(x=oliveoil[,-c(1:2)],center=TRUE,scale=TRUE)
round(var(X),5)

##              palmitic palmitoleic  stearic    oleic linoleic linolenic arac
## palmitic      1.00000     0.83560 -0.17039 -0.83734  0.46068   0.31933   0.
## palmitoleic   0.83560     1.00000 -0.22219 -0.85244  0.62163   0.09311   0.
## stearic      -0.17039    -0.22219  1.00000  0.11360 -0.19782   0.01892  -0.
## oleic        -0.83734    -0.85244  0.11360  1.00000 -0.85032  -0.21817  -0.
## linoleic      0.46068     0.62163 -0.19782 -0.85032  1.00000  -0.05744   0.
## linolenic     0.31933     0.09311  0.01892 -0.21817 -0.05744   1.00000   0.
## arachidic     0.22830     0.08548 -0.04098 -0.31996  0.21097   0.62024   1.
## eicosenoic    0.50195     0.41635  0.14038 -0.42415  0.08904   0.57832   0.
##              eicosenoic
## palmitic        0.50195
## palmitoleic     0.41635
## stearic         0.14038
## oleic          -0.42415
## linoleic        0.08904
## linolenic       0.57832
## arachidic       0.32866
## eicosenoic      1.00000

round(100*diag(var(X))/sum(diag(var(X))),5)

##    palmitic palmitoleic     stearic       oleic    linoleic   linolenic
##        12.5        12.5        12.5        12.5        12.5        12.5
##   arachidic  eicosenoic
##        12.5        12.5
```

   With scaled data every variable has unit variance. Thus it is **no longer** possible to determine a priori the impact of variables.

Then PCA for these data.

```
MM<-prcomp(x = X, center = TRUE, scale = FALSE);   summary(MM)

## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5    PC6     PC7
## Standard deviation     1.9291 1.3288 1.0081 0.89045 0.57777 0.4988 0.34470
## Proportion of Variance 0.4652 0.2207 0.1270 0.09911 0.04173 0.0311 0.01485
## Cumulative Proportion  0.4652 0.6859 0.8129 0.91206 0.95378 0.9849 0.99974
##                            PC8
## Standard deviation     0.04563
## Proportion of Variance 0.00026
## Cumulative Proportion  1.00000
```

We need three to account for over 80% of the total variability in the data, and five components to explain over 95%. Let us look at PC loadings

```
MM$rotation

##                     PC1         PC2         PC3         PC4         PC5
## palmitic     0.46074351  0.04958406 -0.11445834 -0.28043124  0.53473943
## palmitoleic  0.45022576  0.24090732 -0.14260264 -0.21182252  0.13841908
## stearic     -0.09864471 -0.25837844 -0.80215910  0.47082168  0.21340068
## oleic       -0.49417494 -0.15866175  0.08011486 -0.20010742 -0.01552215
## linoleic     0.36569539  0.34339930  0.08747773  0.51249093 -0.40127538
## linolenic    0.21898707 -0.60483760  0.19103316 -0.09881321  0.12507081
## arachidic    0.22830362 -0.44719396  0.42664494  0.48165441  0.14659527
## eicosenoic   0.31186781 -0.40476916 -0.30085585 -0.33222211 -0.67153429
##                     PC6         PC7        PC8
## palmitic    -0.07699892 -0.52540418 0.35438653
## palmitoleic -0.16728954  0.78680816 0.08856309
## stearic      0.03064009  0.07722664 0.07703841
## oleic       -0.11309403  0.18074878 0.79903372
## linoleic     0.30497855 -0.07768793 0.46687817
## linolenic    0.69784174  0.19096065 0.02943890
## arachidic   -0.55365142  0.06527504 0.03996552
## eicosenoic  -0.25657629 -0.13959613 0.04168750
```

Interpretation of PC is as standard, although in this case many variables are involved.

The first component is a comparison between `oleic` and a weighted average of all the remaining variables except `stearic`. The weight of `oleic` are similar to (opposing) weights of `palmitic`, `palmotoleic` and `linoleic`.

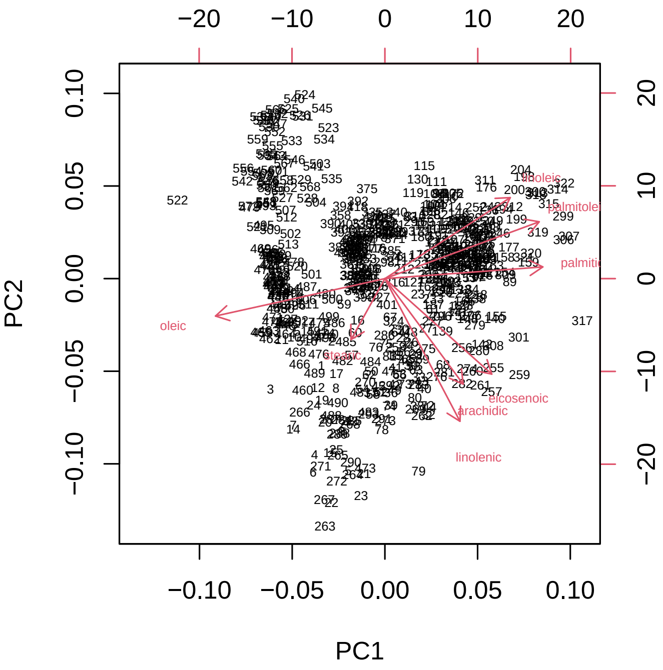
The second component is a comparison between an average of variables `linolenic`, `arachidic`, `eicosenoic`, `stearic` and `oleic` against an average of the rest of variables except `palmitic`. Here the bigger weights are for `linolenic`, `arachidic` and `eicosenoic`.

```
round(MM$rotation,1)
```

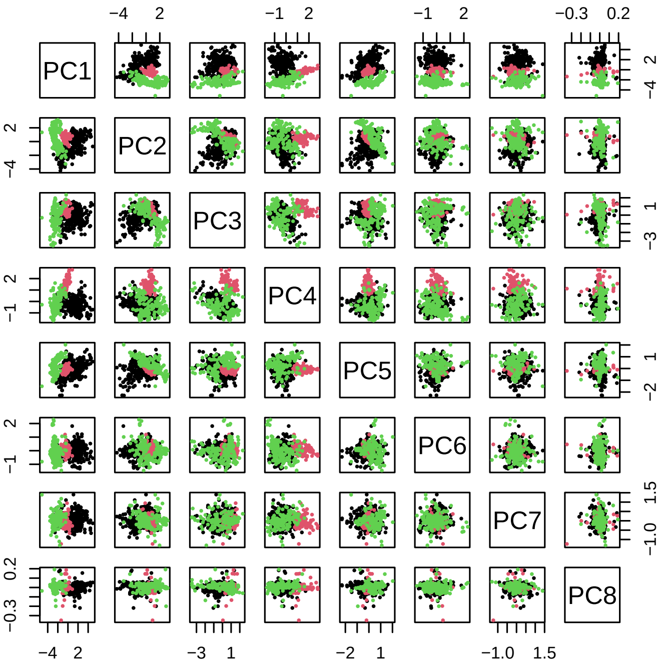
```
##              PC1  PC2  PC3  PC4  PC5  PC6  PC7 PC8
## palmitic     0.5  0.0 -0.1 -0.3  0.5 -0.1 -0.5 0.4
## palmitoleic  0.5  0.2 -0.1 -0.2  0.1 -0.2  0.8 0.1
## stearic     -0.1 -0.3 -0.8  0.5  0.2  0.0  0.1 0.1
## oleic       -0.5 -0.2  0.1 -0.2  0.0 -0.1  0.2 0.8
## linoleic     0.4  0.3  0.1  0.5 -0.4  0.3 -0.1 0.5
## linolenic    0.2 -0.6  0.2 -0.1  0.1  0.7  0.2 0.0
## arachidic    0.2 -0.4  0.4  0.5  0.1 -0.6  0.1 0.0
## eicosenoic   0.3 -0.4 -0.3 -0.3 -0.7 -0.3 -0.1 0.0
```

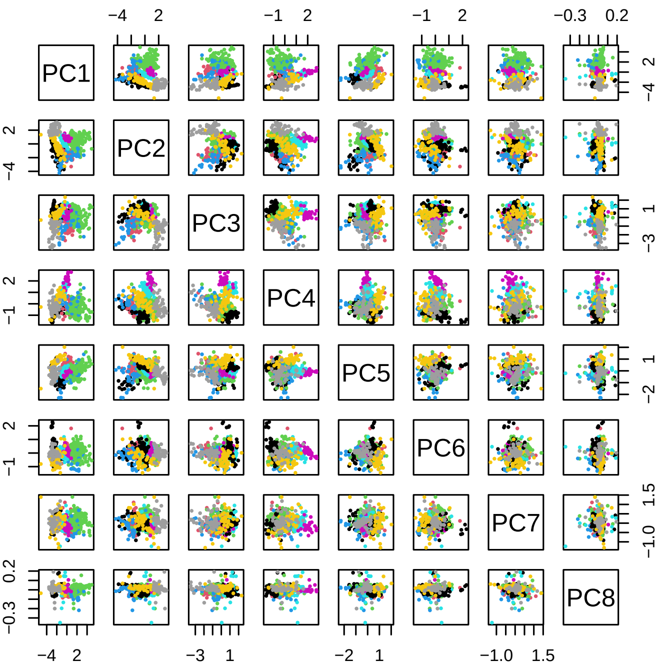
The biplot is added, which is in sharp contrast with the previous result. One striking feature of the biplot is that arrows for variables are larger which is a result of the PC loadings for this scaled data.

```
biplot(MM,cex=0.5)
```



As an extra, we can plot the PC scores using as colors the variables `macro.area` and `region`.




PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
−4
2
−1
2
−1
2
−0.3
0.2
−4
2
−3
1
−2
1
−1.0
1.5
2
−4
1
−3
2
−1
1
−2
2
−1
1.5
−1.0
0.2
−0.3

7. Recall the relation seen in lectures $\mathbf{\Lambda} = \frac{1}{n-1}\mathbf{D}^2$ between eigenvalues of the K-L expansion of $\mathbf{\Sigma}$ and eigenvalues of svd of the data $\mathbf{X}$. Numerically check that the relation holds for both analyses you did in 4 and 6.

   First for non-scaled data.

```
X<-scale(x=oliveoil[,-c(1:2)],center=TRUE,scale=FALSE)
eigen(var(X))$values  ## from Karhunen-Loeve


## [1] 230543.82788  22789.01058   2064.26492    758.82269    615.20792
## [6]    143.52118     51.05564     48.74556


(svd(X)$d)^2/(nrow(X)-1)  ## from svd


## [1] 230543.82788  22789.01058   2064.26492    758.82269    615.20792
## [6]    143.52118     51.05564     48.74556
```

   Then for scaled data

```
X<-scale(x=oliveoil[,-c(1:2)],center=TRUE,scale=TRUE)
eigen(var(X))$values ## from Karhunen-Loeve


## [1] 3.721410009 1.765797520 1.016355435 0.792898832 0.333817667 0.24881866
## [7] 0.118820109 0.002081762


(svd(X)$d)^2/(nrow(X)-1)  ## from svd


## [1] 3.721410009 1.765797520 1.016355435 0.792898832 0.333817667 0.24881866
## [7] 0.118820109 0.002081762
```

MTH6101 Introduction to Machine Learning

Laboratory week five

The intention of this laboratory is to do an introduction to agglomerative clustering in `R`. The tasks involve the library `cluster`.

1. Open `RStudio` and a new `R` script. You need to install the library `cluster`.

2. Type the data of lectures in a matrix called `X`:

x_1	x_2
0	4
3	6
6	2
0	5
1	1

3. Load the library `cluster` and use the function `dist` to build a distance matrix. With this function, build distance matrices exploring different distances such as `manhattan`, `euclidean` and `minkowski` (with exponent equal to four). **Verify** that you can reproduce some the distances between points one and three. Continue exploring this function with different values of the **logical** input flags `diag` and `upper`.

4. Using **euclidean** distance and **complete** linkage, perform aglomerative clustering with these data. To this end, use the function `agnes` from the library you just loaded and save the output in an object termed `A`.

5. Use `plot(A,which.plot=2)` to plot the dendrogram of this analysis.

6. It is clear that the first cluster is formed at distance equal to one. By manipulation of the elements in the distance table, coerced to be a matrix, verify that the (complete linkage) distance between clusters '14' and '2' is precisely the one in the dendrogram. Recall that the distances in the dendrogram are retrieved by `A$height`.

7. Consider the dataset `ruspini` from the same library. Do a scatterplot of these data and use the `text` function to put individual labels in it.

8. Perform agglomerative clustering using **euclidean** distance and **average** linkage. Plot the cluster and try an interpret the dendrogram. Large stems suggest places to cut the diagram and detect clusters.

9. Compare with the dendrogram using the same distance and **single** linkage. Comment on the differences and similarities.

MTH6101 Introduction to Machine Learning

Laboratory week five - Comments and code

The intention of this laboratory is to do an introduction to agglomerative clustering in `R`. The tasks involve the library `cluster`.

1. Open `RStudio` and a new `R` script. You need to install the library `cluster`.

2. Type the data of lectures in a matrix called `X`:

x_1	x_2
0	4
3	6
6	2
0	5
1	1

3. Load the library `cluster` and use the function `dist` to build a distance matrix. With this function, build distance matrices exploring different distances such as `manhattan`, `euclidean` and `minkowski` (with exponent equal to four). **Verify** that you can reproduce some the distances between points one and three. Continue exploring this function with different values of the **logical** input flags `diag` and `upper`.

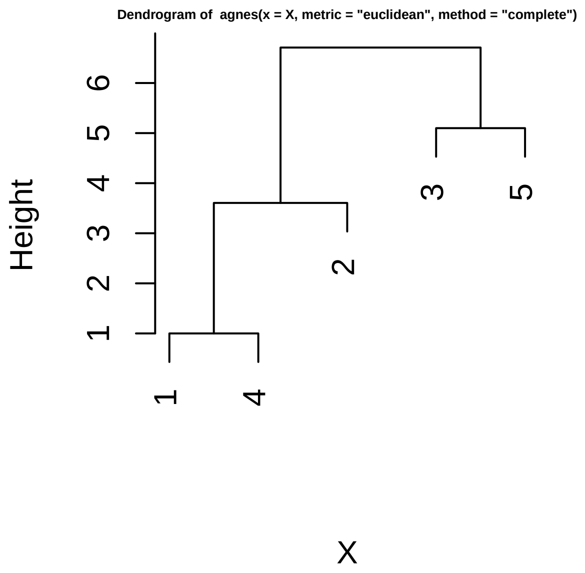
```
library(cluster)
dist(x=X,method="manhattan")

##   1 2 3 4
## 2 5
## 3 8 7
## 4 1 4 9
## 5 4 7 6 5

dist(x=X,method="manhattan",diag=TRUE,upper=TRUE)

##   1 2 3 4 5
## 1 0 5 8 1 4
## 2 5 0 7 4 7
## 3 8 7 0 9 6
## 4 1 4 9 0 5
## 5 4 7 6 5 0

dist(x=X,method="euclidean")

##          1        2        3        4
## 2 3.605551
## 3 6.324555 5.000000
## 4 1.000000 3.162278 6.708204
## 5 3.162278 5.385165 5.099020 4.123106

## this is the (e)distance between points 1 and 3
sqrt(sum((X[1,]-X[3,])^2))

## [1] 6.324555

dist(x=X,method="minkowski",p=4)

##          1        2        3        4
## 2 3.138289
## 3 6.018433 4.284572
## 4 1.000000 3.009217 6.091630
## 5 3.009217 5.031697 5.001999 4.003901

## this is the (mi) distance between points 1 and 3
sqrt(sqrt(sum((X[1,]-X[3,])^4)))

## [1] 6.018433
```

4. Using **euclidean** distance and **complete** linkage, perform aglomerative clustering with these data. To this end, use the function `agnes` from the library you just loaded and save the output in an object termed `A`.

```r
library(cluster)
A<-agnes(x=X,method = "complete",metric="euclidean")
```

5. Use `plot(A,which.plot=2)` to plot the dendrogram of this analysis.

```r
par(mar=c(4,4,1,1))
plot(A,which.plot=2,cex.main=0.4)
```

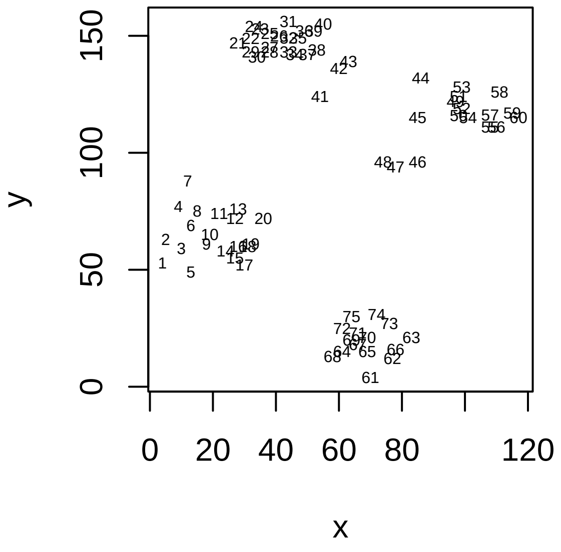


6. It is clear that the first cluster is formed at distance equal to one. By manipulation of the elements in the distance table, coerced to be a matrix, verify that the (complete linkage) distance between clusters ‘14‘ and ‘2‘ is precisely the one in the dendrogram. Recall that the distances in the dendrogram are retrieved by **`A$height`**.

```r
as.matrix(dist(x=X,method="euclidean",diag=TRUE,upper=TRUE))->D
D[c(1,4),2] ## the relevant distances

##        1        4
## 3.605551 3.162278

max(D[c(1,4),2]) ## the distance using complete linkage

## [1] 3.605551

A$height

## [1] 1.000000 3.605551 6.708204 5.099020
```

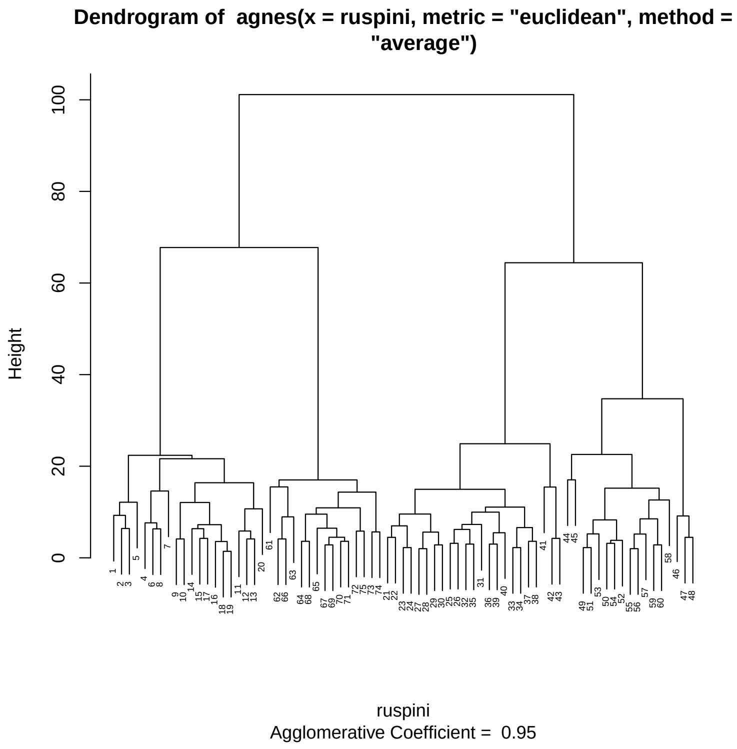
7. Consider the dataset **ruspini** from the same library. Do a scatterplot of these data and use the **text** function to put individual labels in it.

```r
head(ruspini) ## the first few values of the data

##    x  y
## 1  4 53
## 2  5 63
## 3 10 59
## 4  9 77
## 5 13 49
## 6 13 69

par(mar=c(4,4,1,1))
plot(ruspini,type="n")
text(x=ruspini$x,y=ruspini$y,labels = row.names(ruspini),cex=0.5)
```



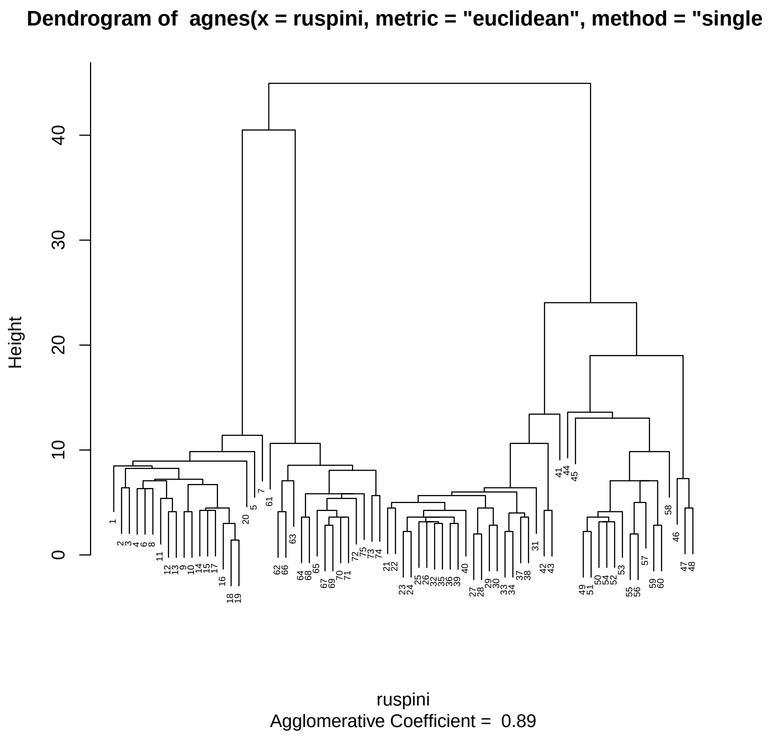
8. Perform agglomerative clustering using **euclidean** distance and **average** linkage. Plot the cluster and try an interpret the dendrogram. Large stems suggest places to cut the diagram and detect clusters.

```r
A<-agnes(x=ruspini,method = "average",metric="euclidean")
plot(A,which.plot=2,cex=0.5)
```



9. Compare with the dendrogram using the same distance and **single** linkage. Comment on the differences and similarities.

```
A<-agnes(x=ruspini,method = "single",metric="euclidean")
plot(A,which.plot=2,cex=0.5)
```



MTH6101 Introduction to Machine Learning

Laboratory week six

The intention of this laboratory is to do a second take on clustering with functions `kmeans` and `pam`, this second from the library `cluster`.

1. Open `RStudio` and create a new `R` script. Load library `cluster`.

2. Consider the data set `USArrests` of arrests statistics in states of the USA. Load it into a variable termed `X1`, taking care of centering and scaling the data. Perform `kmeans` clustering of `X1` for values of $k = 2, \dots, 10$ and record the total sum of squares within clusters. Plot this quantity.

3. Plot the data using text labels and color using the clusters you have found (`$cluster`). To this end do a scatterplot with `type="n"` and then add labels using `text`. Clearly we need to select some variables for the plot so use `Assault` and `UrbanPop`. Interpret your result.

4. You will now redo all the analysis for the Principal Component scores of this data. To this end, redo PCA for the data `USArrests` (centered, scaled), select a number of components and store the PC Scores in variable `X2`. Then do cluster analysis using **`kmeans` on the scores**. Plot ESS for a selection of k between 2 and 10.

5. Do two scatter plots with text labels, one with the variables `Assault` and `UrbanPop` and the colors given by the clustering on PC you just did. The second plot is of `X2` which are PC scores, and the same coloring. Interpret the results.

6. Build `X3` to be the centered and unscaled `iris` data set (no fifth column). Using the medoid clustering method `pam` cluster with $k = 2, 3, 4, 5$ medoids and plot the clusters for sepal variables, coloring according to the clusters made (`$clustering`). Can you suggest some clusters?

7. Repeat the previous step for `X3` not just centered but scaled `iris` data set without the fifth column. Are clusters neater?

8. Repeat the previous `pam` clustering with `X4` which are first two PC scores of the centered and scaled `iris` data set (no fifth column). Can you determine how many clusters would be suitable?

MTH6101 Introduction to Machine Learning

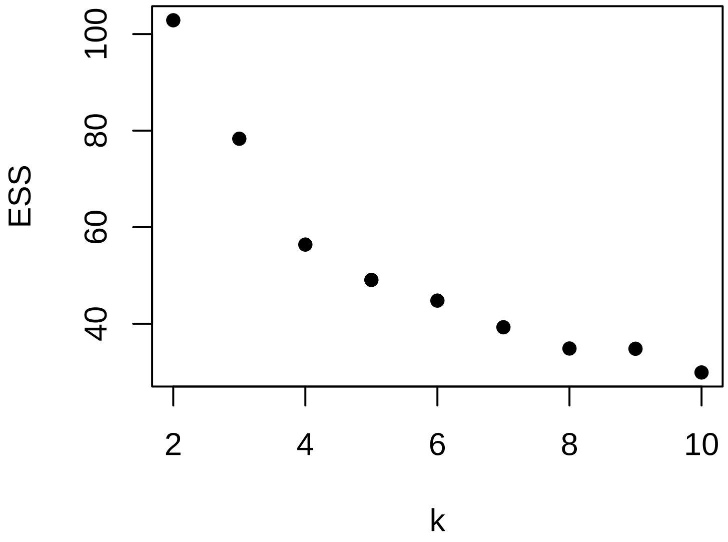
Laboratory week six - Comments and code

The intention of this laboratory is to do a second take on clustering with functions `kmeans` and `pam`, this second from the library `cluster`.

1. Open `RStudio` and create a new `R` script. Load library `cluster`.

2. Consider the data set `USArrests` of arrests statistics in states of the USA. Load it into a variable termed `X1`, taking care of centering and scaling the data. Perform `kmeans` clustering of `X1` for values of $k = 2, \dots, 10$ and record the total sum of squares within clusters. Plot this quantity.

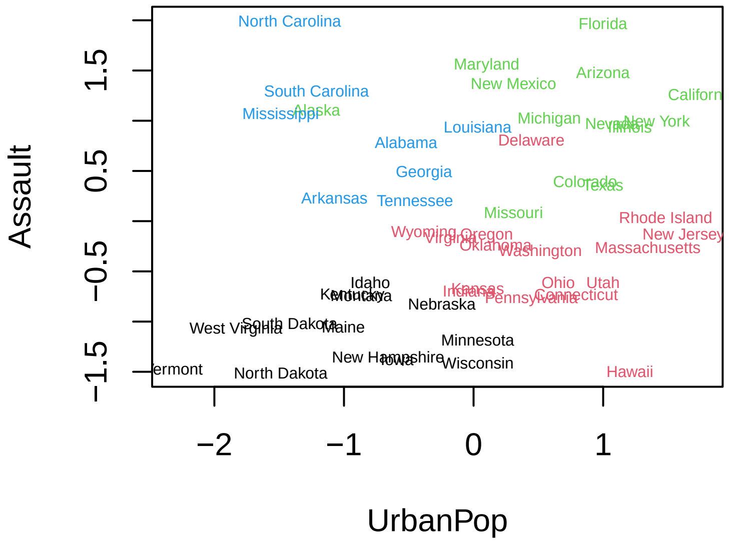
```r
X1<-scale(x=USArrests,center=TRUE,scale=TRUE);   ess<-matrix(nrow=9)
for(k in 1:9)    ess[k]<-kmeans(x=X1,centers=k+1)$tot.withinss
par(mar=c(4,4,1,1));  plot(2:10,ess,xlab="k",ylab="ESS",pch=16)
```



It looks as if 4 clusters could represent the data. Alternatively 8 clusters could be used but it is doubtful to have 8 clusters with 50 data points.

3. Plot the data using text labels and color using the clusters you have found (`$cluster`). To this end do a scatterplot with `type="n"` and then add labels using `text`. Clearly we need to select some variables for the plot so use `Assault` and `UrbanPop`. Interpret your result.

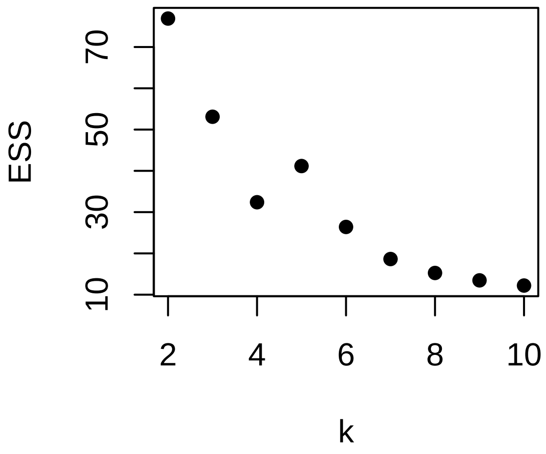
```
par(mar=c(4,4,1,1));
plot(X1[,3],X1[,2],type="n",xlab="UrbanPop",ylab="Assault")
clus<-kmeans(x=X1,centers=4)$cluster
text(x=X1[,3],y=X1[,2],labels=row.names(X1),cex=0.5,col=clus)
```



What would an interpretation be of this data?

4. You will now redo all the analysis for the Principal Component scores of this data. To this end, redo PCA for the data `USArrests` (centered, scaled), select a number of components and store the PC Scores in variable `X2`. Then do cluster analysis using **`kmeans` on the scores**. Plot ESS for a selection of k between 2 and 10.

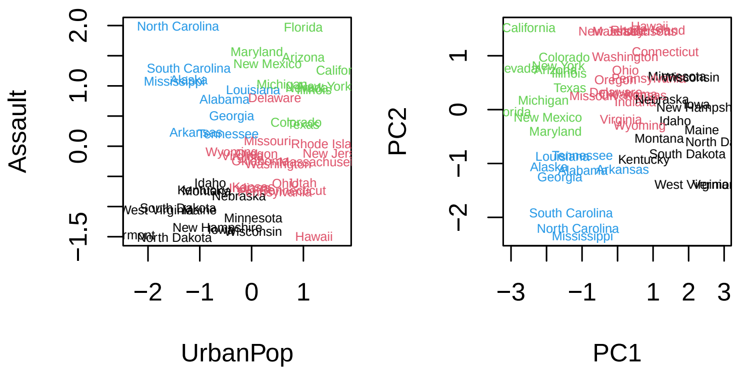
```
A<-prcomp(x=USArrests,center=TRUE,scale=TRUE)
summary(A) ## We take two PC to explain 80% of total variability

## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.5749 0.9949 0.59713 0.41645
## Proportion of Variance 0.6201 0.2474 0.08914 0.04336
## Cumulative Proportion  0.6201 0.8675 0.95664 1.00000

X2<-A$x[,1:2];   ess<-matrix(nrow=9)
for(k in 1:9)   ess[k]<-kmeans(x=X2,centers=k+1)$tot.withinss
par(mar=c(4,4,1,1));  plot(2:10,ess,xlab="k",ylab="ESS",pch=16)
```



Four clusters seem to be the case.

5. Do two scatter plots with text labels, one with the variables `Assault` and `UrbanPop` and the colors given by the clustering on PC you just did. The second plot is of `X2` which are PC scores, and the same coloring. Interpret the results.

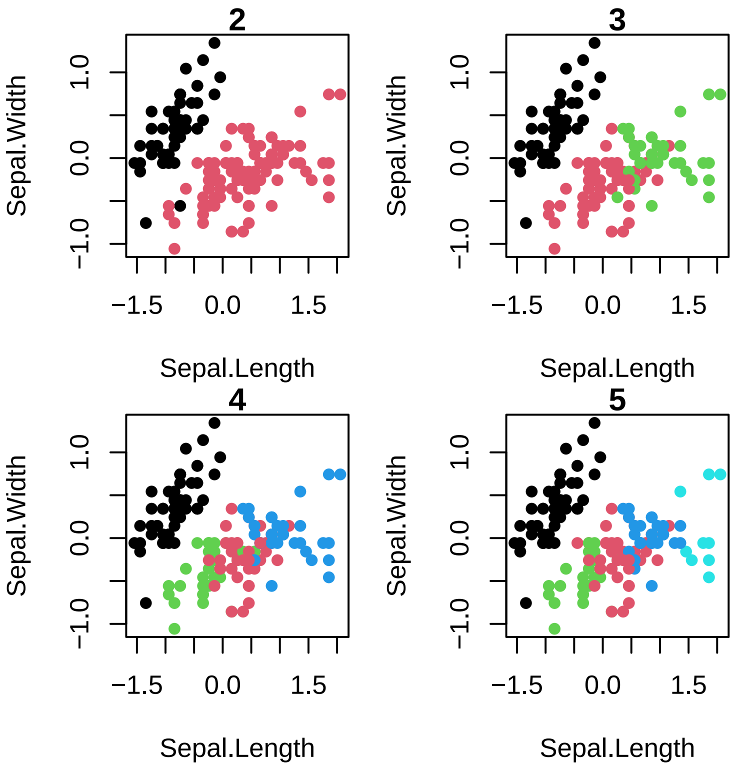
```r
par(mar=c(4,4,1,1),mfrow=c(1,2));
plot(X1[,3],X1[,2],type="n",xlab="UrbanPop",ylab="Assault")
clus<-kmeans(x=X2,centers=4)$cluster
text(x=X1[,3],y=X1[,2],labels=row.names(X1),cex=0.5,col=clus)
plot(X2[,1],X2[,2],type="n",xlab="PC1",ylab="PC2")
text(x=X2[,1],y=X2[,2],labels=row.names(X2),cex=0.5,col=clus)
```



Is this a neater interpretation than the earlier clustering?

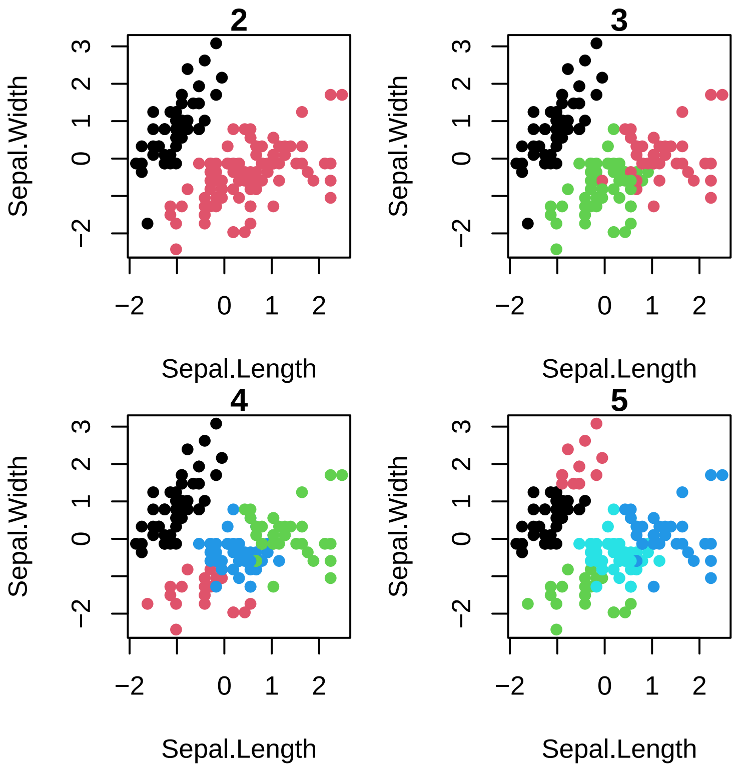
6. Build `X3` to be the centered and unscaled `iris` data set (no fifth column). Using the medoid clustering method `pam` cluster with $k = 2, 3, 4, 5$ medoids and plot the clusters for sepal variables, coloring according to the clusters made (`$clustering`). Can you suggest some clusters?

```
X3<-scale(x=iris[,-5],center=TRUE,scale=FALSE);
library(cluster);   par(mar=c(4,4,1,1),mfrow=c(2,2));
for(i in 1:4)  plot(X3[,1],X3[,2],pch=16,col=pam(x=X3,k=1+i)$clustering,
    main=i+1, xlab=colnames(X3)[1], ylab=colnames(X3)[2])
```



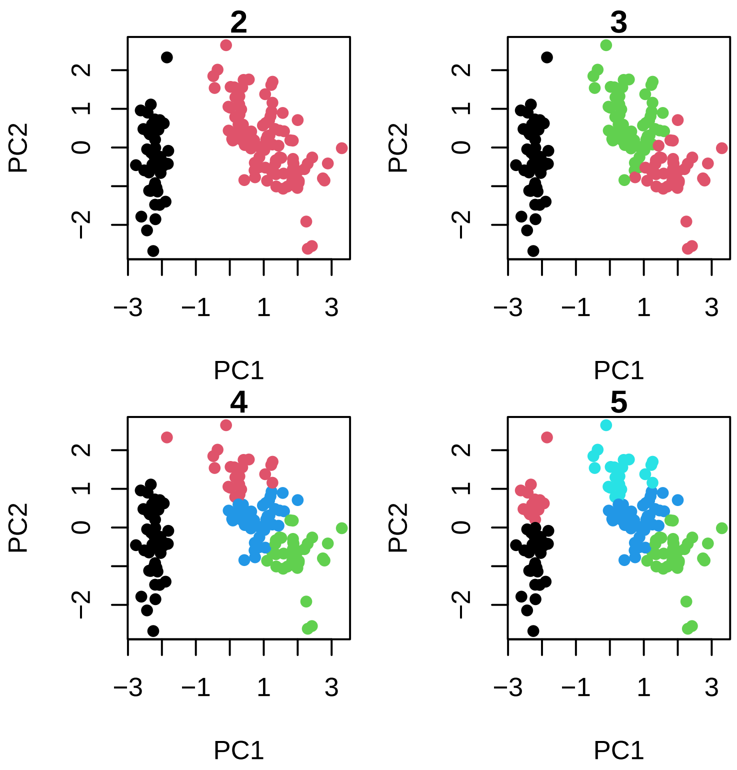
7. Repeat the previous step for `X3` not just centered but scaled `iris` data set without the fifth column. Are clusters neater?

```
X3<-scale(x=iris[,-5],center=TRUE,scale=TRUE);
library(cluster);    par(mar=c(4,4,1,1),mfrow=c(2,2));
for(i in 1:4)  plot(X3[,1],X3[,2],pch=16,col=pam(x=X3,k=1+i)$clustering,
    main=i+1, xlab=colnames(X3)[1], ylab=colnames(X3)[2])
```



8. Repeat the previous `pam` clustering with `X4` which are first two PC scores of the centered and scaled `iris` data set (no fifth column). Can you determine how many clusters would be suitable?

```r
A<-prcomp(x=iris[,-5],center=TRUE,scale=TRUE); X4<-A$x[,1:2]
par(mar=c(4,4,1,1),mfrow=c(2,2));  # summary(A)
for(i in 1:4)  plot(X4[,1],X4[,2],pch=16,col=pam(x=X4,k=1+i)$clustering,
    main=i+1, xlab="PC1", ylab="PC2")
```



MTH6101 Introduction to Machine Learning

Laboratory week eight

The intention of this laboratory is to do a split of data and fit and compare three different classifiers. We will use the dataset `Default` that has customer default records for a credit card company. The aim is to predict whether a customer will default or not.

1. Open `RStudio` and install the libraries `ISLR` and `cvTools`.

2. We will first create a set of indices for and 80/20 data split of data that will be used for latter modelling. To this end, load the library `cvTools` then **do remember to set** the random seed to a value of zero before creating the folds. Use for this the command `set.seed`. Now create the folds by running the function `cvFolds` for a split of $n = 10000$ values in $K = 5$ folds. Save the output of this command in the variable `CV`.

3. Briefly examine and describe the contents of your newly created variable `CV`.

4. We will use the **first four** folds (values of `CV$which` of $1, 2, 3, 4$) to **train** the models. The **fifth** fold, (value of `CV$which` equal to 5) will be used to **test** and compare models. To this end, associate the index `CV$subsets[CV$which!=5]` to a variable called `Train`, and `CV$subsets[CV$which==5]` to a variable called `Test`.

5. Load the library `ISLR` and examine the dataset `Default`: note how many values are available, note and do a summary of the variables of the dataset, do a pairs plot of the data. Note that the variable of interest is `default`. In addition, you may want to do an advanced pairs plot with `ggpairs` from the library `GGally`.

6. Using the function `glm` and using the training data (`data = Default[Train,]`), build the following logistic (`family = "binomial"`) models:

   - A model to predict `default` as function of `balance` (`default∼balance`), stored in `M1`.
   - A model to predict `default` as function of `balance` and `student` (`default∼balance+student`), stored in `M2`.
   - A model to predict `default` as function of all the variables (`default∼.`), stored in `M3`.

7. Using the function `predict.glm` and using the test data (`newdata =Default[Test,]`), build predictions (`type="response"`) for each of the models `M1,M2,M3` you just built. Store your predictions in variables `P1,P2,P3`

8. Now we are ready to build confusion matrices for each case. Using the variable `Ytrue<-Default[Test,]$default=="Yes"` and the indicators e.g. `Y1<-P1>0.5`, use the command `table` to build a matrix for each model.

9. For each of models `M1,M2,M3` compute performance measures True Positive Rate TPR $= \frac{\mathrm{TP}}{\mathrm{P}}$ and False Positive Rate FPR $= \frac{\mathrm{FP}}{\mathrm{N}}$.

10. (**Extra**) Using a loop, repeat all the computations you have done so that you effectively do a 5-fold crossvalidation for the three models. The only extra ingredient you need is a variable to keep the TPR and FPR for the different models. Then plot TPR vs. FPR in $[0,1]^2$, coloring by model.

MTH6101 Introduction to Machine Learning

Laboratory week eight - Comments and code

The intention of this laboratory is to do a split of data and fit and compare three different classifiers. We will use the dataset `Default` that has customer default records for a credit card company. The aim is to predict whether a customer will default or not.

1. Open `RStudio` and install the libraries `ISLR` and `cvTools`.

2. We will first create a set of indices for and 80/20 data split of data that will be used for latter modelling. To this end, load the library `cvTools` then **do remember to set** the random seed to a value of zero before creating the folds. Use for this the command `set.seed`. Now create the folds by running the function `cvFolds` for a split of $n = 10000$ values in $K = 5$ folds. Save the output of this command in the variable `CV`.

```
library(cvTools)
set.seed(0)
n<-10000; K<-5
cvFolds(n=n,K=K)->CV
```

3. Briefly examine and describe the contents of your newly created variable `CV`.

```
head(CV$which)

## [1] 1 2 3 4 5 1

head(CV$subsets)

##      [,1]
## [1,] 9614
## [2,] 1017
## [3,] 8004
## [4,] 4775
## [5,] 9725
## [6,] 8462
```

Comments: The variable `CV` has two elements that will be used to split the data. The first element is `CV$subsets` that contains a random permutation of the data indices $1, 2, \ldots, 10000$, and the second element is `CV$which` which indexes the five folds created with values $1, 2, 3, 4, 5$.

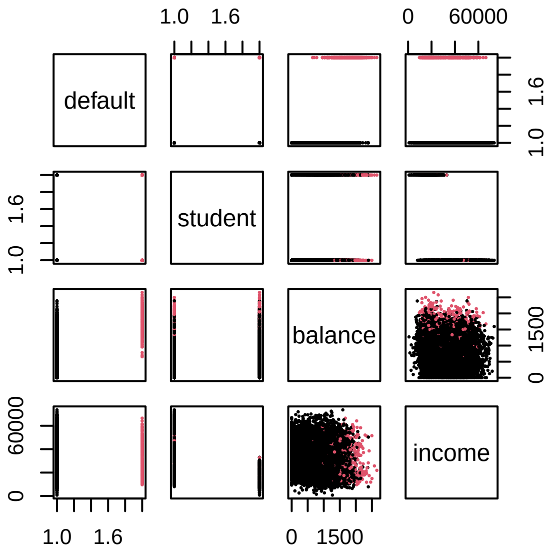
4. We will use the **first four** folds (values of `CV$which` of $1, 2, 3, 4$) to **train** the models. The **fifth** fold, (value of `CV$which` equal to 5) will be used to **test** and compare models. To this end, associate the index `CV$subsets[CV$which!=5]` to a variable called `Train`, and `CV$subsets[CV$which==5]` to a variable called `Test`.

```
CV$subsets[CV$which!=5]->Train
CV$subsets[CV$which==5]->Test
```

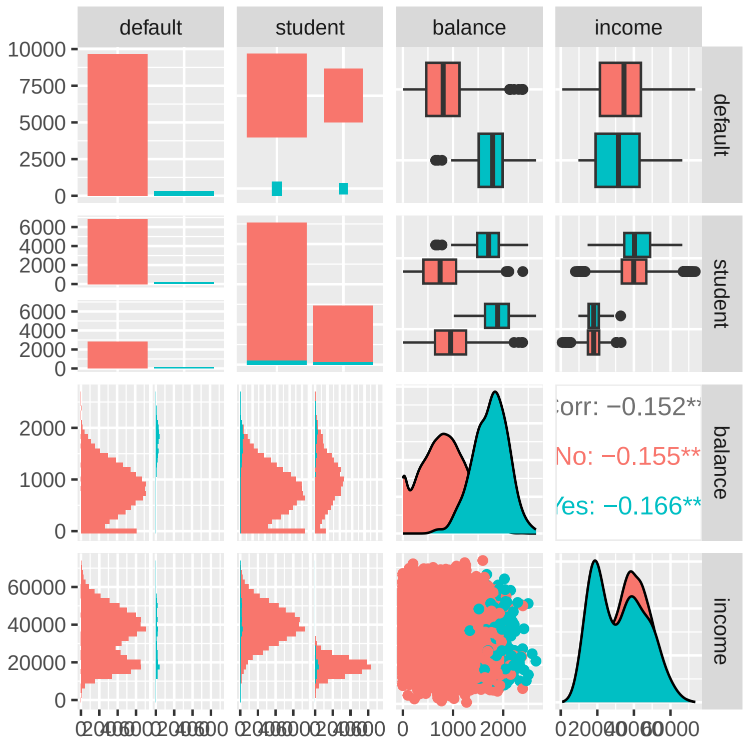
5. Load the library `ISLR` and examine the dataset `Default`: note how many values are available, note and do a summary of the variables of the dataset, do a pairs plot of the data. Note that the variable of interest is `default`. In addition, you may want to do an advanced pairs plot with `ggpairs` from the library `GGally`.

```
library(ISLR)
dim(Default)

## [1] 10000     4

summary(Default)

##  default    student       balance           income     
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772  
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340  
##                        Median : 823.6   Median :34553  
##                        Mean   : 835.4   Mean   :33517  
##                        3rd Qu.:1166.3   3rd Qu.:43808  
##                        Max.   :2654.3   Max.   :73554
```

```
head(Default)

##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559

pairs(Default,cex=0.35,pch=16,col=Default$default)
```



```
library(GGally)
ggpairs(Default,ggplot2::aes(colour=Default$default))
```



Comments: The datase has variables `default`, `student`, `balance`, `income`. The response variable `default` and the variable `student` are categorical, while the other two variables are not. We see that only a small proportion of persons in the database have defaulted. It may be that `balance` can predict `default`, and it is not clear that students default more than the non-students.

6. Using the function `glm` and using the training data (`data = Default[Train,]`), build the following logistic (`family = "binomial"`) models:

   - A model to predict `default` as function of `balance` (`default~balance`), stored in `M1`.
   - A model to predict `default` as function of `balance` and `student` (`default~balance+student`), stored in `M2`.
   - A model to predict `default` as function of all the variables (`default~.`), stored in `M3`.

```
M1<-glm(default~balance,family = "binomial",data = Default[Train,])
M2<-glm(default~balance+student,family = "binomial",data = Default[Train,])
M3<-glm(default~.,family = "binomial",data = Default[Train,])
```

7. Using the function `predict.glm` and using the test data (`newdata =Default[Test,]`), build predictions (`type="response"`) for each of the models `M1,M2,M3` you just built. Store your predictions in variables `P1,P2,P3`

```
predict.glm(object = M1, newdata =Default[Test,],type="response" ) -> P1
predict.glm(object = M2, newdata =Default[Test,],type="response" ) -> P2
predict.glm(object = M3, newdata =Default[Test,],type="response" ) -> P3
```

8. Now we are ready to build confusion matrices for each case. Using the variable `Ytrue<-Default[Test,]$default=="Yes"` and the indicators e.g. `Y1<-P1>0.5`, use the command `table` to build a matrix for each model.

```
Ytrue<-Default[Test,]$default=="Yes"
Y1<-P1>0.5
Y2<-P2>0.5
Y3<-P3>0.5
table(Ytrue,Y1) ## For M1
```

```
##        Y1
## Ytrue   FALSE TRUE
##   FALSE  1933    7
##   TRUE     39   21
```

```
table(Ytrue,Y2) ## For M2
```

```
##        Y2
## Ytrue   FALSE TRUE
##   FALSE  1934    6
##   TRUE     36   24
```

```
table(Ytrue,Y3) ## For M3
```

```
##        Y3
## Ytrue   FALSE TRUE
##   FALSE  1934    6
##   TRUE     36   24
```

9. For each of models `M1,M2,M3` compute performance measures True Positive Rate TPR $= \frac{\mathrm{TP}}{\mathrm{P}}$ and False Positive Rate FPR $= \frac{\mathrm{FP}}{\mathrm{N}}$.

```
sum( Ytrue&Y1 )/sum(Ytrue); sum( !Ytrue&Y1 )/sum(!Ytrue) ## TPR and FPR M1
```

```
## [1] 0.35
## [1] 0.003608247
```

```
sum( Ytrue&Y2 )/sum(Ytrue); sum( !Ytrue&Y2 )/sum(!Ytrue) ## TPR and FPR M2
```

```
## [1] 0.4
## [1] 0.003092784
```

```
sum( Ytrue&Y3 )/sum(Ytrue); sum( !Ytrue&Y3 )/sum(!Ytrue) ## TPR and FPR M3
```

```
## [1] 0.4
## [1] 0.003092784
```

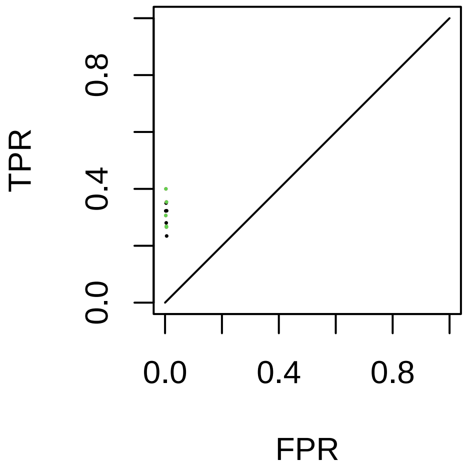
10. (**Extra**) Using a loop, repeat all the computations you have done so that you effectively do a 5-fold crossvalidation for the three models. The only extra ingredient you need is a variable to keep the TPR and FPR for the different models. Then plot TPR vs. FPR in $[0,1]^2$, coloring by model.

```r
result<-matrix(nrow=K,ncol=6) ## variable for the results
for(k in 1:K){
   ## selecting the folds to train and test
   CV$subsets[CV$which!=k]->Train; CV$subsets[CV$which==k]->Test
   ## train the models
   M1<-glm(default~balance,family = "binomial",data = Default[Train,])
   M2<-glm(default~balance+student,family = "binomial",data = Default[Train,]
   M3<-glm(default~.,family = "binomial",data = Default[Train,])
   ## validate the models
   predict.glm(object = M1, newdata =Default[Test,],type="response" ) -> P1
   predict.glm(object = M2, newdata =Default[Test,],type="response" ) -> P2
   predict.glm(object = M3, newdata =Default[Test,],type="response" ) -> P3
   ## Responses to compare
   Ytrue<-Default[Test,]$default=="Yes"; Y1<-P1>0.5; Y2<-P2>0.5; Y3<-P3>0.5
   ## Store the performance measures
   result[k,1]<-sum( Ytrue&Y1 )/sum(Ytrue) ## TPR M1
   result[k,2]<-sum( !Ytrue&Y1 )/sum(!Ytrue) ## FPR
   result[k,3]<-sum( Ytrue&Y2 )/sum(Ytrue) ## TPR
   result[k,4]<-sum( !Ytrue&Y2 )/sum(!Ytrue) ## FPR
   result[k,5]<-sum( Ytrue&Y3 )/sum(Ytrue) ## TPR
   result[k,6]<-sum( !Ytrue&Y3 )/sum(!Ytrue) ## FPR
}
```

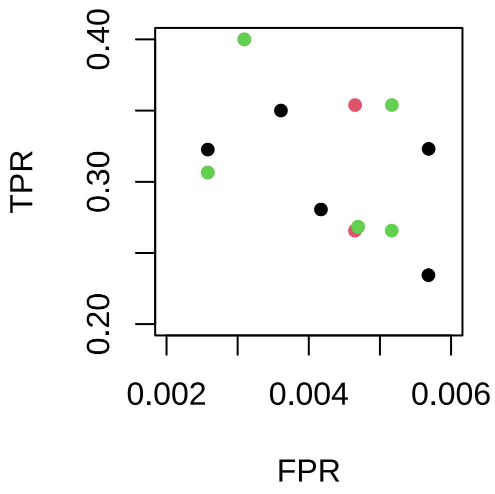
Comments: Note that the code for the loop just requires minor alterations of the code done for earlier analysis. The interpretation of the scatterplot is lecture material.

```r
## Labelling of results for clarity
colnames(result)<-1:6
colnames(result)[c(1,3,5)]<-paste("TPRM",1:3,sep="")
colnames(result)[c(2,4,6)]<-paste("FPRM",1:3,sep="")
```

```
result

##           TPRM1       FPRM1     TPRM2       FPRM2     TPRM3       FPRM3
## [1,] 0.2343750 0.005681818 0.2656250 0.004648760 0.2656250 0.005165289
## [2,] 0.3230769 0.005684755 0.3538462 0.004651163 0.3538462 0.005167959
## [3,] 0.2804878 0.004171011 0.2682927 0.004692388 0.2682927 0.004692388
## [4,] 0.3225806 0.002579979 0.3064516 0.002579979 0.3064516 0.002579979
## [5,] 0.3500000 0.003608247 0.4000000 0.003092784 0.4000000 0.003092784

apply(result,2,mean)

##       TPRM1       FPRM1       TPRM2       FPRM2       TPRM3       FPRM3
## 0.302104075 0.004345162 0.318843090 0.003933015 0.318843090 0.004139680

par(mar=c(4,4,1,1)); plot(c(0,1),c(0,1),type="l",xlab="FPR",ylab="TPR")
for(i in c(1,3,5))
    points(x=result[,i+1],y=result[,i],col=(i+1)/2,pch=16,cex=0.25)
```



Zoom of the previous plot in the region of results TPR, FPR. Note that axes have very different scales.



MTH6101 Introduction to Machine Learning

Laboratory week nine

The aim of this practice is to build different classifiers on the same data set and to compare them. The file `glass.data` has measurements of 10 variables in seven different types of glass. This file has 11 columns, and the last column is the type of glass. We are interested in classifying headlamps (type of glass equals seven) against all the other types of glass.

When you **start** your session, open `RStudio` and install/load the following libraries: `cvTools`, `class`, `tree`, `MASS`, `pROC`.

1. Reading the data and adapting the file for analysis.

   Read the data using the command `read.csv(file = "glass.data",header = FALSE)`, store it in a variable termed `X`. Save the last column of `X` in a variable called `Y` and remove the first column. Then center and scale the matrix `X` using the command `scale`, saving it in `X`.

   Now we will prepare the output for analysis. The variable `Y` will be used twice, first as 0/1 variable then as "Yes"/"No". Replace `Y` by `(Y==7)*1` and substitute the last column of `X` for this value of `Y`.

   At this point, the first 9 columns of `X` have the centered values and the last one has 0/1 values. Give names to the columns of `X` according to
   `colnames(X)<-c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")`
   Finally, convert the variable `Y` to "Yes"/"No" by the commands `Y[Y==1]<-"Yes"; Y[Y==0]<-"No"`. Merge `X` and `Y` in a data frame called `DAT` by `DAT<-data.frame(X,Y)`.

2. Now we create the partition of data into training and testing datasets. For this, we create a partition 66:33. Set seed equal to zero and using `cvFolds` from `cvTools`, create variables `Train` and `Test` for the training and testing partition respectively. See the notes where this has been done in virtually every example.

3. Using the training data, fit the **logistic classifier** for the response variable `Type` using all but the last column of the data (i.e. `DAT[Train,-11]`). Save the fitted model in variable `M1`. Then predict the response using the fitted model `M1` and the test partition. Save this in a variable called `P1`.

4. Examine the fitted classifier and identify variables that are not important for the response. Using only those variables that were found significant in the

first logistic model, fit a second **logistic classifier** and a second prediction set. These are to be called `M11` and `P11`.

5. Fit a K **nearest neighbors classifier**. Here use three nearest neighbors in the function `knn` from the library `class` and recall that only columns `1:9` of the data frame contain variables. Call this model `M2` and recall that `M2` already contains the predicted classes.

6. Fit a **tree classifier** to the data using the function `tree` from library `tree`. The response is `Y` and here the column ten from the data frame is not to be used (i.e. `DAT[Train,-10]`). Save the fitted model in variable called `M3`. Predict output with option `type="class"` and save it in a variable called `P3`.

7. Fit a **linear discriminant classifier** using the function `lda` from library `MASS`. Use the training data except the last column (i.e. `DAT[Train,-11]`) to fit variable `Type` and save the output in a variable termed `M4`. Using the test data, predict output using the fitted model and save results in variable termed `P4`.

8. Now we prepare to compare all classifiers. To this end, recall that we can do ROC curve for **logistic** and **linear discriminant** classifiers. For KNN and tree we will compute only points in ROC graph. Using the function `roc` from the library `pROC`, compute and ROC for models `M1`, `M11` and `M4` and save the results in variables `R1`, `R11` and `R4`.

9. For each of **KNN** and **tree** classifiers, compute the confusion matrix using the command `table`. In each case, compute the figures TPR and FPR. Save results in variables `TPR2,FPR2` and `TPR3,FPR3`.

10. In a single ROC graph, plot ROC curves for **logistic** and **linear discriminant** classifiers; add points for the **KNN** and **tree** classifiers. Compute AUC for the classifiers `M1`, `M11` and `M4` and summarize your results.

11. (**Extra**) Plot the tree for classifier `M3` and interpret it.

MTH6101 Introduction to Machine Learning

Laboratory week nine - Comments and code

The aim of this practice is to build different classifiers on the same data set and to compare them. The file `glass.data` has measurements of 10 variables in seven different types of glass. This file has 11 columns, and the last column is the type of glass. We are interested in classifying headlamps (type of glass equals seven) against all the other types of glass.

When you **start** your session, open `RStudio` and install/load the following libraries: `cvTools`, `class`, `tree`, `MASS`, `pROC`.

1. Reading the data and adapting the file for analysis.

   Read the data using the command `read.csv(file = "glass.data",header = FALSE)`, store it in a variable termed `X`. Save the last column of `X` in a variable called `Y` and remove the first column. Then center and scale the matrix `X` using the command `scale`, saving it in `X`.

   ```
   X<-read.csv(file = "glass.data",header = FALSE)
   dim(X)

   ## [1] 214  11

   Y<-X[,11]; X<-X[,-1]
   dim(X)

   ## [1] 214  10

   X<-scale(x=X,center=TRUE,scale=TRUE)
   ```

   Now we will prepare the output for analysis. The variable `Y` will be used twice, first as 0/1 variable then as “Yes”/“No”. Replace `Y` by `(Y==7)*1` and substitute the last column of `X` for this value of `Y`.

   ```
   (Y==7)*1->Y;  X[,10]<-Y
   ```

   At this point, the first 9 columns of `X` have the centered values and the last one has 0/1 values. Give names to the columns of `X` according to

`colnames(X)<-c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")`
Finally, convert the variable `Y` to "Yes"/"No" by the commands `Y[Y==1]<-"Yes"; Y[Y==0]<-"No"`. Merge `X` and `Y` in a data frame called `DAT` by `DAT<-data.frame(X,Y)`.

```r
colnames(X)<-c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")
Y[Y==1]<-"Yes"; Y[Y==0]<-"No"
DAT<-data.frame(X,factor(Y)); colnames(DAT)[11]<-"Y"
attach(DAT)
```

Data is ready for analysis.

2. Now we create the partition of data into training and testing datasets. For this, we create a partition 66:33. Set seed equal to zero and using `cvFolds` from `cvTools`, create variables `Train` and `Test` for the training and testing partition respectively. See the notes where this has been done in virtually every example.

```r
library(cvTools)
set.seed(0);
n<-nrow(X); K<-3;
cvFolds(n=n,K=K)->CV
CV$subsets[CV$which!=K]->Train;
CV$subsets[CV$which==K]->Test
```

3. Using the training data, fit the **logistic classifier** for the response variable `Type` using all but the last column of the data (i.e. `DAT[Train,-11]`). Save the fitted model in variable `M1`. Then predict the response using the fitted model `M1` and the test partition. Save this in a variable called `P1`.

```r
M1<-glm(Type~.,data=DAT[Train,-11],family="binomial")
P1<-predict.glm(M1,newdata = DAT[Test,],type="response")
```

4. Examine the fitted classifier and identify variables that are not important for the response. Using only those variables that were found significant in the first logistic model, fit a second **logistic classifier** and a second prediction set. These are to be called `M11` and `P11`.

   Comments: With the command `summary(M1)`, a list of the coefficients and associated p-values can be examined. Variables `RI` and `Fe` are not significant and are thus discarded for `M11` and `P11`.

```
M11<-glm(Type~.-RI-Fe,data=DAT[Train,-11],family="binomial")
P11<-predict.glm(M11,newdata = DAT[Test,],type="response")
```

5. Fit a K **nearest neighbors classifier**. Here use three nearest neighbors in the function `knn` from the library `class` and recall that only columns `1:9` of the data frame contain variables. Call this model `M2` and recall that `M2` already contains the predicted classes.

```
library(class)
M2<-knn(train = X[Train,1:9],test = X[Test,1:9],cl = Y[Train],k = 3)
```

6. Fit a **tree classifier** to the data using the function `tree` from library `tree`. The response is `Y` and here the column ten from the data frame is not to be used (i.e. `DAT[Train,-10]`). Save the fitted model in variable called `M3`. Predict output with option `type="class"` and save it in a variable called `P3`.

```
library(tree)
M3<-tree(Y~.,DAT[Train,-10])
P3<-predict(M3,DAT[Test,],type="class")
```

7. Fit a **linear discriminant classifier** using the function `lda` from library `MASS`. Use the training data except the last column (i.e. `DAT[Train,-11]`) to fit variable `Type` and save the output in a variable termed `M4`. Using the test data, predict output using the fitted model and save results in variable termed `P4`.

```
library(MASS)
M4<-lda(Type~.,data=DAT[Train,-11])
P4<-predict(M4,DAT[Test,])
```

Comments: At this point, you should have the following `R` variables with outputs from classifiers.

`R` variables	Classifier	ROC curve doable?
`M1, P1`	Logistic, all variables	Yes
`M11, P11`	Logistic, reduced set of variables	Yes
`M2`	KNN	No
`M3, P3`	Tree	No
`M4, P4`	Linear discriminant	Yes

8. Now we prepare to compare all classifiers. To this end, recall that we can do ROC curve for **logistic** and **linear discriminant** classifiers. For KNN and tree we will compute only points in ROC graph. Using the function `roc` from the library `pROC`, compute and ROC for models `M1`, `M11` and `M4` and save the results in variables `R1`, `R11` and `R4`.

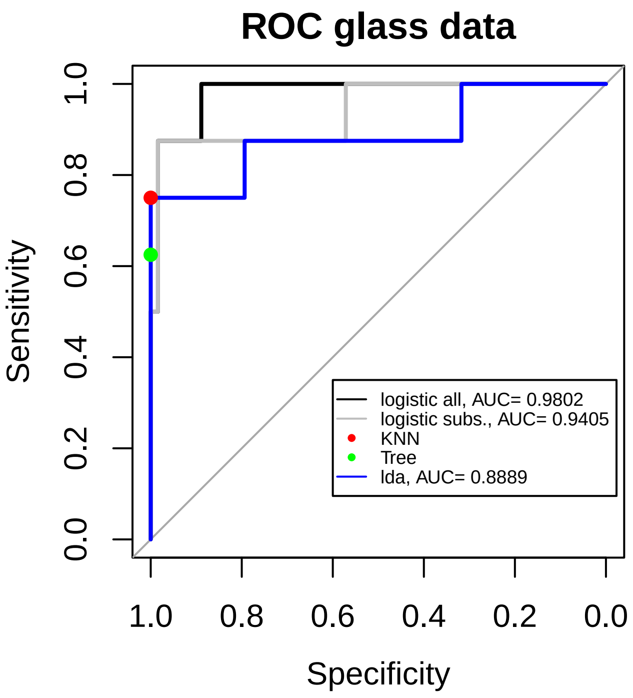
```
library(pROC)
roc(response=Type[Test],predictor=P1)->R1
roc(response=Type[Test],predictor=P11)->R11
roc(response=Type[Test],predictor=c(P4$x))->R4
```

9. For each of **KNN** and **tree** classifiers, compute the confusion matrix using the command `table`. In each case, compute the figures TPR and FPR. Save results in variables `TPR2,FPR2` and `TPR3,FPR3`.

```
(table(Y[Test],M2)->T2) ## KNN

##        M2
##         No Yes
##    No   63   0
##    Yes   2   6

T2[2,2]/sum(T2[2,])->TPR2; T2[1,2]/sum(T2[1,])->FPR2
(table(Type[Test],P3)->T3) ## Tree

##     P3
##       No Yes
##    0 63   0
##    1  3   5

T3[2,2]/sum(T3[2,])->TPR3; T3[1,2]/sum(T3[1,])->FPR3
```

10. In a single ROC graph, plot ROC curves for **logistic** and **linear discriminant** classifiers; add points for the **KNN** and **tree** classifiers. Compute AUC for the classifiers `M1`, `M11` and `M4` and summarize your results.

```r
plot(R1,col="black",main="ROC glass data")
plot(R11,col="grey",add=TRUE)
plot(R4,add=TRUE,col="blue")
points(1-FPR2,TPR2,col="red",pch=16)
points(1-FPR3,TPR3,col="green",pch=16)
## To make things simple, I do *not* write code for legend here
```



The

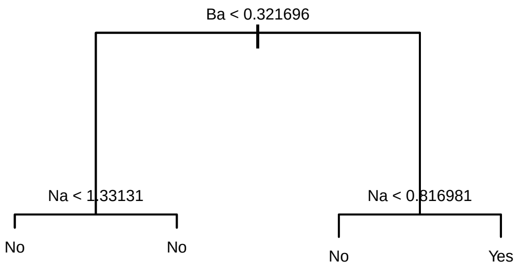
following are AUC values for three of the classifiers:

```
auc(R1) ## logistic with all variables

## Area under the curve: 0.9802

auc(R11) ## logistic with subset of variables

## Area under the curve: 0.9405

auc(R4) ## linear discriminant classifier

## Area under the curve: 0.8889
```

Comments:
The best classifier in this analysis was the **logistic classifier with all variables** (`M1`), . Removing variables to form a **logistic with subset of variables** (`M11`) did not help the predictive capability of this logistics classifier who came as second best. The **linear discriminant analysis** classifier (`M4`) came in third place.
The classifiers **KNN** (`M2`) and **Tree** (`M3`) were very good in classifying negatives, but at most were as good as the linear discriminant for classifying positives.

11. (**Extra**) Plot the tree for classifier `M3` and interpret it.

```r
plot(M3)
text(M3,pretty=0,cex=0.5)
```



Comments: This tree is not minimal and can be simplified by noting that the comparison in the left branch can be removed without changing the result. Only **two variables** are used in this classifier which states:
"A piece of glass which for which its percent content of Barium is bigger than 0.335 (0.3217 in centered and scaled units) **and** whose percent content of Sodium exceeds 14.075 (0.817 in centered and scaled units) is **classified as a headlamp** otherwise it is not."

MTH6101 Introduction to Machine Learning

Laboratory week eleven

In this lab we analyse the `diabetes` data set (Efron et al. 2004). The data has $n = 442$ diabetes patients measured on $p = 10$ baseline variables. A prediction model is needed for the response variable `y` (a measure of disease progression one year after baseline). We use **ridge regression**, the **lasso** and **elastic net**. Before you start your `RStudio` session, install and load the libraries `cvTools`, `lars` and `glmnet`. Also run the function `ridge` as seen in the notes.

Data preparation

1. Load the data with `data(diabetes)` and allocate `diabetes$x` into variable `X` and `diabetes$y` to variable `Y`. Center and scale both `X` and `Y`. Using `as.matrix` and the variables create a data frame `DAT` which will be used as well.

2. Create validation index variables `Train` and `Test` for a $3 : 1$ partition of the data. Use the function `cvFolds` from library `cvTools` as earlier.

Ridge regression

3. Using `seq`, create 200 values in the range -6 to 6 which is exponentiated so that λ ranges from 10^{-6} to 10^{6}. Store this sequence in variable `rangelambda`. Then using the function `ridge`, train the **ridge regression** model and store it in a variable termed `M1`. The function uses training and validation covariates `X[Train,]`, `X[Test,]` and responses `Y[Train,]`, `Y[Test,]` respectively. This function produces MSE, degrees of freedom, and the ridge coefficients.

4. Plot MSE against λ and indicate the location of the minimum with a vertical line. Also plot the **ridge trace**. For both cases use `log="x"` for the horizontal axes. Give the ridge coefficients suggested by minimum MSE.

Lasso

5. Train the **lasso** using the function `lars` and the training data. Store results in variable `M2`. This output `M2` needs postprocessing to create predictions.

6. Create lasso predictions with the lasso coefficients and the validation data so that you can compute the MSE and select a point in the lasso path. Do two plots: one with MSE and the **lasso path**. Give the selected coefficients.

Elastic net

7. Analyze the data with the **elastic net**. Use `glmnet` with `family="gaussian"` and $\alpha = 0.5$ and the same values of λ as stored in variable `rangelambda` from earlier. Store results in variable `M3`.

8. Postprocess `M3` to predict y values and use these to compute MSE. Plot both the MSE and the `ridge trace` and give the selected coefficients.

Compare results

9. Finally, compare your results against that of simple linear regression for the same data, termed `M4`. For comparison, compute the least squares error using the validation data as well.

MTH6101 Introduction to Machine Learning

Laboratory week eleven - Comments and code

In this lab we analyse the `diabetes` data set (Efron et al. 2004). The data has $n = 442$ diabetes patients measured on $p = 10$ baseline variables. A prediction model is needed for the response variable `y` (a measure of disease progression one year after baseline). We use **ridge regression**, the **lasso** and **elastic net**. Before you start your `RStudio` session, install and load the libraries `cvTools`, `lars` and `glmnet`. Also run the function `ridge` as seen in the notes.

Data preparation

1. Load the data with `data(diabetes)` and allocate `diabetes$x` into variable `X` and `diabetes$y` to variable `Y`. Center and scale both `X` and `Y`. Using `as.matrix` and the variables create a data frame `DAT` which will be used as well.

```
## Libraries and function ridge required
library(cvTools); library(lars); library(glmnet)
ridge<-function(Xtrain,Ytrain,Xval,Yval,lval){
  YTrain<-matrix(ncol=1,Ytrain )
  S<-svd(x=Xtrain);  ## Svd to compute ridge
  N<-length(lval);  ## Number of lambda values
  betav<-matrix(nrow=ncol(Xtrain),ncol=N); ## Objects for the analysis and out
  Ypred<-matrix(nrow=nrow(Xval),ncol=N); DF<-MSE<-matrix(ncol=N,nrow=1)
  rownames(betav)<-colnames(Xtrain)
  for(i in 1:N){
    lambda<-lval[i]
    betav[,i]<-c( (S$v)%*%diag(S$d/(S$d^2+lambda))%*%t(S$u)%*%Ytrain) ## ridg
    Ypred[,i]<-Xval%*%matrix(ncol=1,betav[,i]) ## Predictions
    MSE[1,i]<-mean( (  Yval -Ypred[,i]    )^2  )  ## MSE
    DF[1,i]<-sum( S$d^2/(S$d^2+lambda) )  ## Effective degrees of freedom
  }
  return(list("beta"=betav,"MSE"=c(MSE),"df"=DF))
}
## Loading the data
data(diabetes)
X<-as.matrix(diabetes$x); Y<-as.matrix(diabetes$y)
X<-scale(x=X,center=TRUE,scale = TRUE); Y<-scale(x=Y,center=TRUE,scale=TRUE)
DAT<-data.frame(cbind(X,Y)); colnames(DAT)<-c(colnames(X),"Y")
```

2. Create validation index variables **Train** and **Test** for a 3 : 1 partition of the data. Use the function **cvFolds** from library **cvTools** as earlier.

```
set.seed(0);
n<-nrow(X); K<-4;
cvFolds(n=n,K=K)->CV
CV$subsets[CV$which!=K]->Train; CV$subsets[CV$which==K]->Test
```

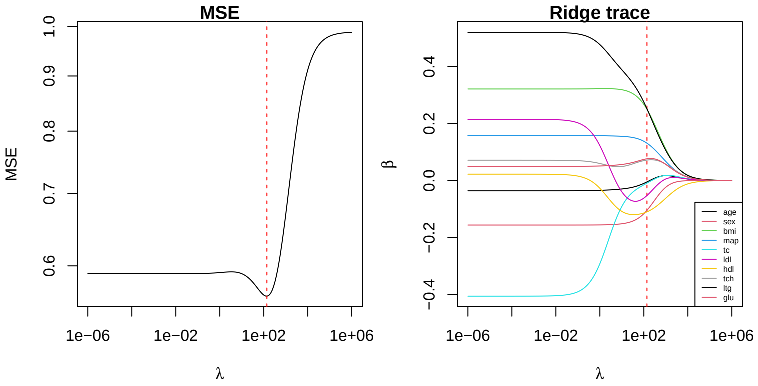
Ridge regression

3. Using **seq**, create 200 values in the range -6 to 6 which is exponentiated so that λ ranges from 10^{-6} to 10^{6}. Store this sequence in variable **rangelambda**. Then using the function **ridge**, train the **ridge regression** model and store it in a variable termed **M1**. The function uses training and validation covariates **X[Train,]**, **X[Test,]** and responses **Y[Train,]**, **Y[Test,]** respectively. This function produces MSE, degrees of freedom, and the ridge coefficients.

```
rangelambda<-10^seq(from=-6,to=6,length.out=200)
M1<-ridge(Xtrain = X[Train,],Ytrain = Y[Train],Xval = X[Test,],
          Yval = Y[Test],lval = rangelambda)
```

4. Plot MSE against λ and indicate the location of the minimum with a vertical line. Also plot the **ridge trace**. For both cases use **log="x"** for the horizontal axes. Give the ridge coefficients suggested by minimum MSE.

```
lambdamin<-rangelambda[which.min(M1$MSE)]
par(mar=c(4,4,1,1),mfrow=c(1,2))
## The MSE
plot(rangelambda,M1$MSE,log="xy",type="l",xlab=expression(lambda),
     ylab="MSE",main="MSE")
abline(v=lambdamin,col="red",lty=2)
## The ridge trace
plot(range(rangelambda),range(M1$beta),log="x",xlab=expression(lambda),
     ylab=expression(beta),type="n", main="Ridge trace")
abline(v=lambdamin,col="red",lty=2)
for(i in 1:nrow(M1$beta)) lines(rangelambda,M1$beta[i,],col=i)
legend("bottomright",legend = rownames(M1$beta),cex=0.5,
       col=1:nrow(M1$beta),lty=1)
```



```
## Results below will be shown later
betaridge<-M1$beta[,which.min(M1$MSE)]
MSEridge<-min(M1$MSE)
```

Lasso

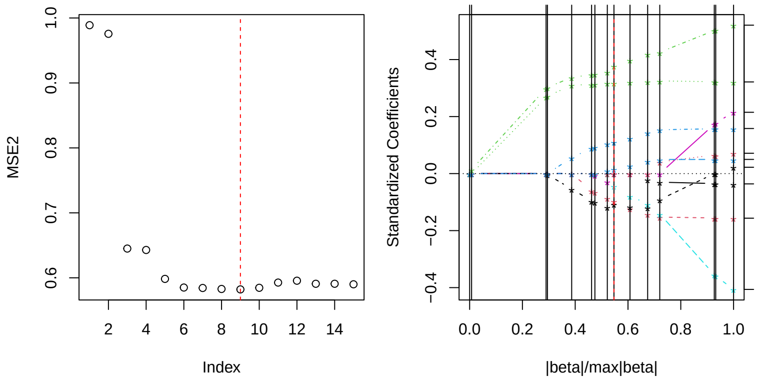
5. Train the **lasso** using the function `lars` and the training data. Store results in variable `M2`. This output `M2` needs postprocessing to create predictions.

```
M2<-lars(x=X[Train,],y = Y[Train],type = "lasso",
         normalize = FALSE,intercept = FALSE)
```

6. Create lasso predictions with the lasso coefficients and the validation data so that you can compute the MSE and select a point in the lasso path. Do two plots: one with MSE and the **lasso path**. Give the selected coefficients.

```
P2<-predict.lars(object = M2,newx = X[Test,],type = "fit")
Yobs<-matrix(nrow=nrow(P2$fit),ncol=ncol(P2$fit),
        byrow=FALSE,Y[Test])
MSE2<-apply((Yobs-P2$fit)^2,2,mean)
par(mar=c(4,4,1,1),mfrow=c(1,2))
plot(MSE2) ## the MSE
```

```
abline(v=which.min(MSE2),col="red",lty=2)
plot(M2)   ## The lasso path
## Results to be shown later
betalasso<-M2$beta[which.min(MSE2),]
shrinklasso<-sum(abs(betalasso))/sum(abs(M2$beta[nrow(M2$beta),]))
abline(v=shrinklasso,col="red",lty=2)
```



```
MSElasso<-min(MSE2)
```

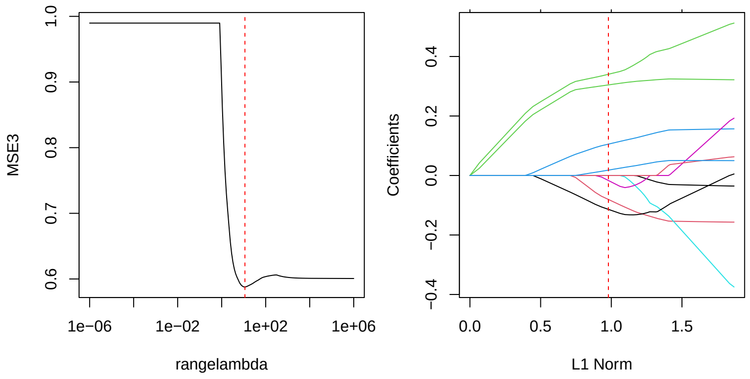
Elastic net

7. Analyze the data with the **elastic net**. Use `glmnet` with `family="gaussian"` and $\alpha = 0.5$ and the same values of λ as stored in variable `rangelambda` from earlier. Store results in variable `M3`.

```
M3<-glmnet(x=X[Train,],y=Y[Train],family="gaussian",alpha = 0.5,
           lambda = rangelambda)
```

8. Postprocess `M3` to predict y values and use these to compute MSE. Plot both the MSE and the `ridge trace` and give the selected coefficients.

```r
P3<-predict.glmnet(object = M3,newx = X[Test,],type="response")
Yobs<-matrix(nrow=nrow(P3),ncol=ncol(P3),
       byrow=FALSE,Y[Test])
MSE3<-apply((Yobs-P3)^2,2,mean)
lambdanet<-rangelambda[which.min(MSE3)]
par(mar=c(4,4,1,1),mfrow=c(1,2))  ## The MSE
plot(rangelambda,MSE3,type="l",log="x")
abline(v=lambdanet,col="red",lty=2)
plot(M3) ## The elastic net path
## Results to be shown later
betanet<-M3$beta[,which.min(MSE3)]
abline(v=sum(abs(betanet)),col="red",lty=2)
```



```r
MSEnet<-min(MSE3)
```

Compare results

9. Finally, compare your results against that of simple linear regression for the same data, termed `M4`. For comparison, compute the least squares error using the validation data as well.

```
M4<-lm(Y~.-1,data=DAT[Train,])
P4<-predict.lm(object = M4,newdata = DAT[Test,],type="response")
betaols<-M4$coefficients
MSEols<-mean( (Y[Test]-P4)^2)
```

Comparison of results **ridge**, **lasso**, **elastic net** and **ols**.

```
MSEall<-cbind(MSEridge,MSElasso,MSEnet,MSEols) ## the MSE
betaall<-cbind(betaridge,betalasso,betanet,betaols) ## Coefficients
round(MSEall,4);

##      MSEridge MSElasso MSEnet MSEols
## [1,]   0.5624   0.5822 0.5878 0.5901

round(betaall,4)

##     betaridge betalasso betanet betaols
## age   -0.0050    0.0000  0.0000 -0.0362
## sex   -0.1048   -0.0967 -0.0806 -0.1564
## bmi    0.2503    0.3176  0.3045  0.3216
## map    0.1311    0.1109  0.1054  0.1582
## tc    -0.0091   -0.0448  0.0000 -0.4063
## ldl   -0.0529    0.0000 -0.0169  0.2151
## hdl   -0.1102   -0.1081 -0.1139  0.0223
## tch    0.0718    0.0000  0.0000  0.0713
## ltg    0.2519    0.3769  0.3403  0.5206
## glu    0.0763    0.0153  0.0180  0.0497
```

The skrinkage achieved with respect to L_1 and L_2 norms.

```
L1s<-apply(X = abs(betaall),MARGIN = 2,FUN = sum)
L2s<-apply(X = betaall^2,MARGIN = 2,FUN = sum)
shrink<-rbind(L1s[1:3]/L1s[4],L2s[1:3]/L2s[4]);
rownames(shrink)<-c("L1","L2"); round(100*shrink,2)

##    betaridge betalasso betanet
## L1     54.32     54.68   50.04
## L2     27.97     43.21   37.18
```