

# Machine Learning

*Mohamad Ghassany*

*2018-12-04*



# Contents

<b>Welcome</b>	<b>7</b>
Course Overview . . . . .	7
<b>Introduction</b>	<b>9</b>
What is Machine Learning ? . . . . .	9
Supervised Learning . . . . .	9
Unsupervised Learning . . . . .	15
 <b>I Supervised Learning</b>	 <b>19</b>
 <b>Regression</b>	 <b>23</b>
<b>1 Linear Regression</b>	<b>23</b>
1.1 Notation . . . . .	23
1.2 Model Representation . . . . .	24
1.3 Why Estimate $f$ ? . . . . .	26
Prediction . . . . .	26
Inference . . . . .	27
1.4 Simple Linear Regression Model . . . . .	27
1.5 Estimating the Coefficients . . . . .	27
1.6 Assessing the Accuracy of the Coefficient Estimates . . . . .	28
Hypothesis testing . . . . .	29
1.7 ANOVA and model fit . . . . .	30
1.7.1 ANOVA . . . . .	30
1.7.2 The $R^2$ Statistic . . . . .	32
 <b>PW 1</b>	 <b>35</b>
1.8 Some R basics . . . . .	35
1.8.1 Basic Commands . . . . .	35
1.8.2 Vectors . . . . .	36
1.8.3 Matrices, data frames and lists . . . . .	36
1.8.4 Graphics . . . . .	39
1.8.5 Distributions . . . . .	40
1.8.6 Working directory . . . . .	42
1.8.7 Loading Data . . . . .	42
1.9 Regression . . . . .	43
1.9.1 The <code>lm</code> function . . . . .	43
1.9.2 Predicting House Value: Boston dataset . . . . .	46
 <b>2 Multiple Linear Regression</b>	 <b>51</b>

2.1	The Model . . . . .	51
2.2	Estimating the Regression Coefficients . . . . .	52
2.3	Some important questions . . . . .	53
2.3.1	Other Considerations in Regression Model . . . . .	56
2.4	How to select the best performing model . . . . .	56
	Use the Adjusted $R^2_{adj}$ for univariate models . . . . .	56
	Have a look at the residuals or error terms . . . . .	57
	Histogram of residuals . . . . .	59
<b>PW 2</b>		<b>61</b>
	Multiple Linear Regression . . . . .	61
	Reporting . . . . .	62
<b>Classification</b>		<b>67</b>
<b>3 Logistic Regression</b>		<b>67</b>
3.1	Introduction . . . . .	67
3.2	Logistic Regression . . . . .	67
3.2.1	The Logistic Model . . . . .	67
3.2.2	Estimating the Regression Coefficients . . . . .	69
3.2.3	Prediction . . . . .	69
3.3	Multiple Logistic Regression . . . . .	70
3.4	Example . . . . .	71
3.4.1	Case study: <i>The Challenger disaster</i> . . . . .	71
<b>PW 3</b>		<b>79</b>
	Social Networks Ads . . . . .	79
<b>4 Discriminant Analysis</b>		<b>85</b>
4.1	Introduction . . . . .	85
4.2	Bayes' Theorem . . . . .	85
4.3	LDA for $p = 1$ . . . . .	86
4.4	Estimating the parameters . . . . .	88
4.5	LDA for $p > 1$ . . . . .	89
4.6	Making predictions . . . . .	90
4.7	Other forms of Discriminant Analysis . . . . .	90
4.7.1	Quadratic Discriminant Analysis (QDA) . . . . .	91
4.7.2	Naive Bayes . . . . .	91
4.8	LDA vs Logistic Regression . . . . .	91
<b>PW 4</b>		<b>93</b>
	Logistic Regression . . . . .	93
	Decision Boundary of Logistic Regression . . . . .	95
	Linear Discriminant Analysis (LDA) . . . . .	96
	Quadratic Discriminant Analysis (QDA) . . . . .	98
	Comparison . . . . .	98
<b>5 Special Session</b>		<b>99</b>
<b>II Dimensionality Reduction</b>		<b>101</b>
<b>6 Principal Components Analysis</b>		<b>103</b>
6.1	Introduction . . . . .	103

6.2 Principal Components . . . . .	103
Notations and Procedure . . . . .	103
First Principal Component ( $PC_1$ ): $Y_1$ . . . . .	105
Second Principal Component ( $PC_2$ ): $Y_2$ . . . . .	105
$i^{th}$ Principal Component ( $PC_i$ ): $Y_i$ . . . . .	106
6.3 How do we find the coefficients? . . . . .	106
Why It May Be Possible to Reduce Dimensions . . . . .	107
Procedure . . . . .	108
6.4 Standardization of the features . . . . .	108
6.5 Projection of the data . . . . .	109
Scores . . . . .	109
Visualization . . . . .	109
Extra . . . . .	109
6.6 Case study . . . . .	110
Employment in European countries in the late 70s . . . . .	110
<b>PW 6</b>	<b>123</b>
The Iris Dataset . . . . .	123
Loading Data . . . . .	124
Exploratory analysis . . . . .	124
PCA using <code>princomp()</code> . . . . .	126
Deeper PCA using <code>factoextra</code> package . . . . .	128
Step-by-step PCA . . . . .	128
<b>III Unsupervised Learning</b>	<b>131</b>
<b>7 Clustering</b>	<b>133</b>
7.1 Unsupervised Learning . . . . .	133
7.2 Clustering . . . . .	133
7.3 Introduction . . . . .	135
Hard clustering . . . . .	135
Fuzzy clustering . . . . .	135
7.4 $k$ -Means . . . . .	136
7.5 $k$ -means in R . . . . .	138
7.5.1 Cluster Validity, Choosing the Number of Clusters . . . . .	140
<b>PW 7</b>	<b>141</b>
Reporting . . . . .	141
Markdown . . . . .	141
R Markdown . . . . .	141
The report to be submitted . . . . .	142
$k$ -means clustering . . . . .	143
pointsCards . . . . .	144
Ligue 1 . . . . .	144
PCA . . . . .	145
Implementing $k$ -means . . . . .	145
<b>8 Gaussian Mixture Models &amp; EM</b>	<b>147</b>
8.1 The Gaussian distribution . . . . .	148
8.2 Mixture of Gaussians . . . . .	148
8.3 EM for Gaussian Mixtures . . . . .	151
<b>PW 8</b>	<b>155</b>
Report template . . . . .	155

8.4	EM using <code>mclust</code> . . . . .	155
	GMM vs $k$ -means . . . . .	155
	EM on 1D . . . . .	158
8.5	EM from scratch . . . . .	159
<b>Appendix</b>		<b>159</b>
<b>A</b>	<b>Introduction to RStudio</b>	<b>161</b>
<b>B</b>	<b>Review on hypothesis testing</b>	<b>163</b>
<b>C</b>	<b>Use of qualitative predictors</b>	<b>165</b>
<b>D</b>	<b>Model Selection</b>	<b>167</b>
	Linear Model Selection and Best Subset Selection . . . . .	167
	Forward Stepwise Selection . . . . .	167
	Backward Stepwise Selection . . . . .	167
	Estimating Test Error Using Mallow's Cp, AIC, BIC, Adjusted R-squared . . . . .	167
	Estimating Test Error Using Cross-Validation . . . . .	167
	Examples . . . . .	167
	Best Subset Selection . . . . .	167
	Forward Stepwise Selection and Model Selection Using Validation Set . . . . .	167
	Model Selection Using Cross-Validation . . . . .	167
<b>E</b>	<b>References</b>	<b>169</b>

# Welcome

Welcome! In this course you will learn about the state of the art of Machine Learning and also gain practice implementing and deploying machine learning algorithms.

The aim of Machine Learning is to build computer systems that can adapt to their environments and learn from experience. Learning techniques and methods from this field are successfully applied to a variety of learning tasks in a broad range of areas, including, for example, spam recognition, text classification, gene discovery, financial forecasting. The course will give an overview of many concepts, techniques, and algorithms in machine learning, beginning with topics such as linear regression and classification and ending up with topics such as kmeans and Expectation Maximization. The course will give you the basic ideas and intuition behind these methods, as well as a more formal statistical and computational understanding. You will have an opportunity to experiment with machine learning techniques in R and apply them to a selected problem.

## Course Overview



# Introduction

## What is Machine Learning ?

What is Machine Learning?

Two definitions of Machine Learning are offered. Arthur Samuel described it as: “the field of study that gives computers the ability to learn without being explicitly programmed.” This is an older, informal definition.

Tom Mitchell provides a more modern definition: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Machine Learning is also called Statistical Learning.

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

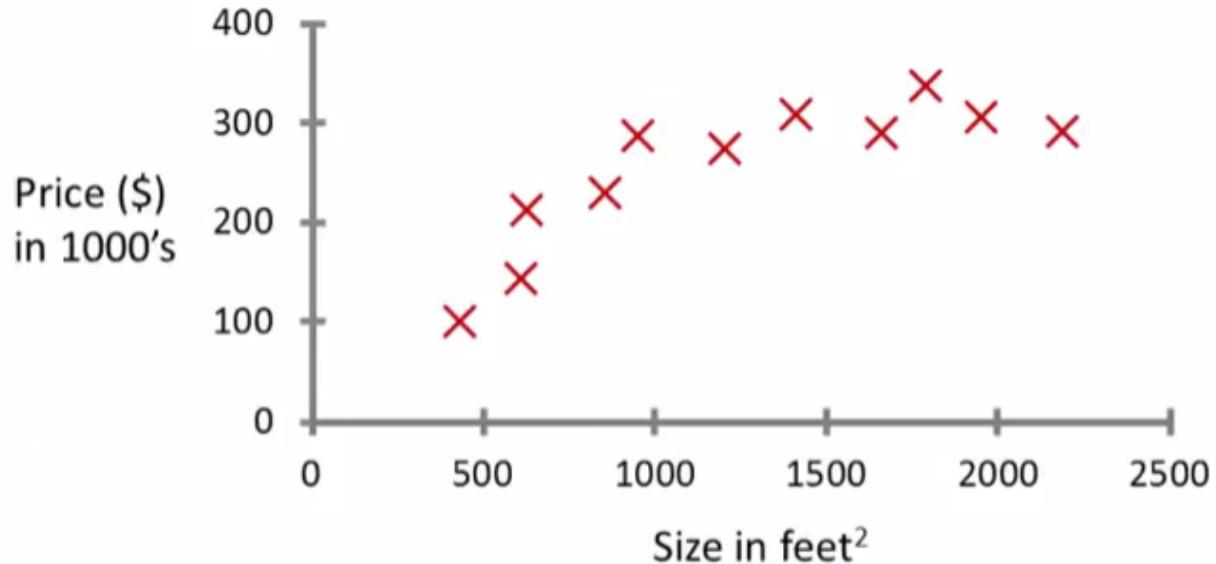
P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of two broad classifications:

Supervised learning and Unsupervised learning.

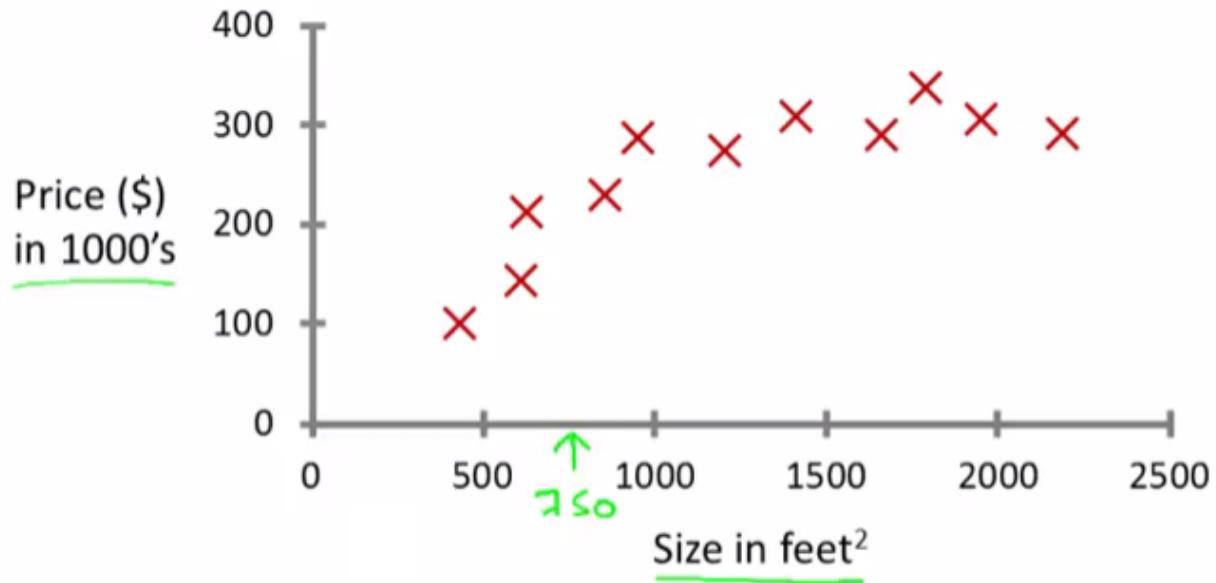
## Supervised Learning

Supervised Learning is probably the most common type of machine learning problem. Let’s start with an example of what is it. Let’s say we want to predict housing prices. We plot a data set and it looks like this.



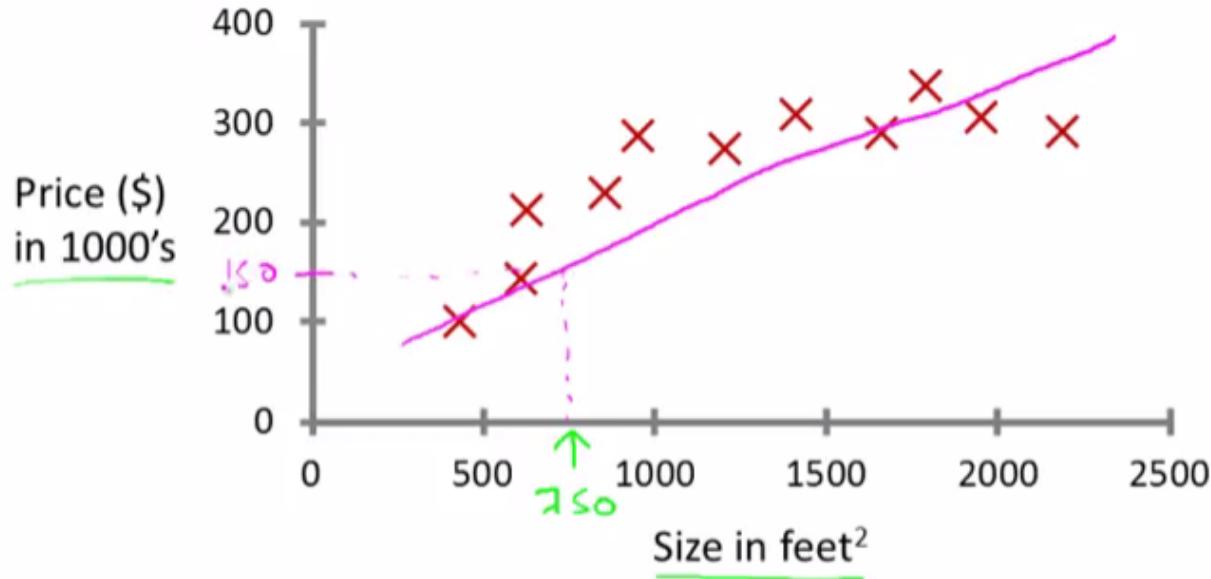
Here on the horizontal axis, the size of different houses in square feet, and on the vertical axis, the price of different houses in thousands of dollars.

So. Given this data, let's say we own a house that is, say 750 square feet and hoping to sell the house and we want to know how much we can get for the house.

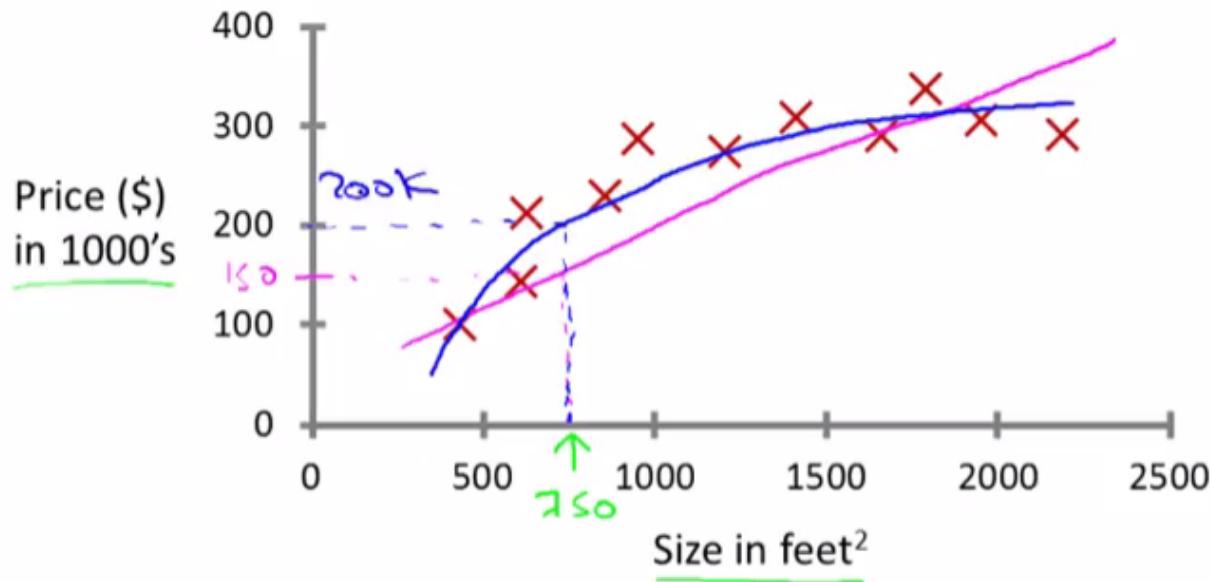


So how can the learning algorithm help?

One thing a learning algorithm might be able to do is put a straight line through the data or to “fit” a straight line to the data and, based on that, it looks like maybe the house can be sold for maybe about \$150,000.



But maybe this isn't the only learning algorithm we can use. There might be a better one. For example, instead of sending a straight line to the data, we might decide that it's better to fit a *quadratic function* or a *second-order polynomial* to this data.



If we do that, and make a prediction here, then it looks like, well, maybe we can sell the house for closer to \$200,000.

This is an example of a supervised learning algorithm.

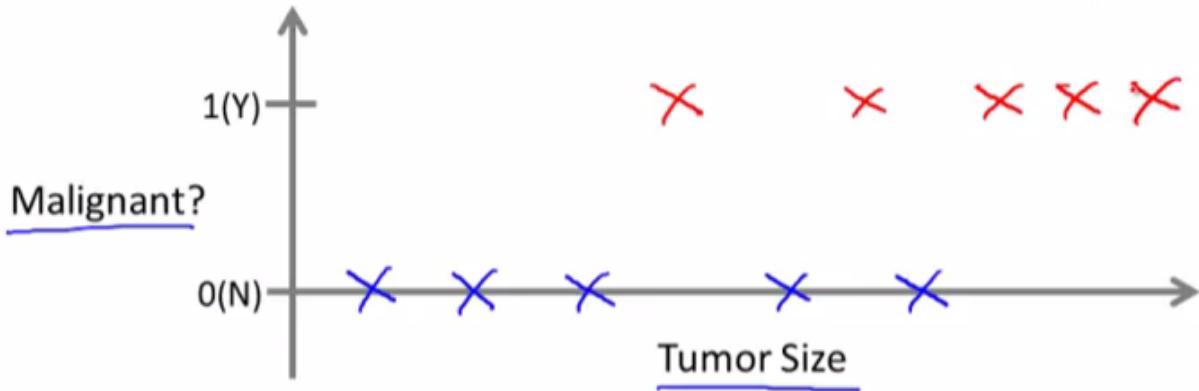
The term supervised learning refers to the fact that we gave the algorithm a data set in which the “right answers” were given.

The example above is also called a regression problem. A regression problem is when we try to predict a **continuous** value output. Namely the price in the example.

Here's another supervised learning example. Let's say we want to look at medical records and try to predict

of a breast cancer as malignant or benign. If someone discovers a breast tumor, a lump in their breast, a malignant tumor is a tumor that is harmful and dangerous and a benign tumor is a tumor that is harmless. Let's see a collected data set and suppose in the data set we have the size of the tumor on the horizontal axis and on the vertical axis we plot one or zero, yes or no, whether or not these are examples of tumors we've seen before are malignant (which is one) or zero if not malignant or benign.

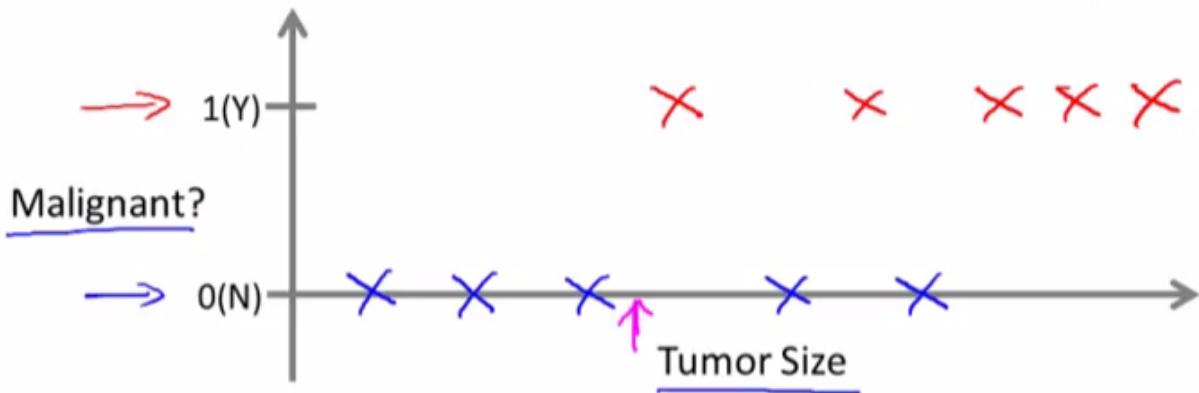
## Breast cancer (malignant, benign)



In this data set we have five examples of benign tumors, and five examples of malignant tumors.

Let's say a person who tragically has a breast tumor, and let's say her breast tumor size is known (rose arrow in the following figure).

## Breast cancer (malignant, benign)



The machine learning question is, can you estimate what is the probability that a tumor is malignant versus benign? To introduce a bit more terminology this is an example of a **classification** problem.

The term classification refers to the fact that here we're trying to predict a **discrete** value output: zero or one, malignant or benign. And it turns out that in classification problems sometimes you can have more than two values for the two possible values for the output.

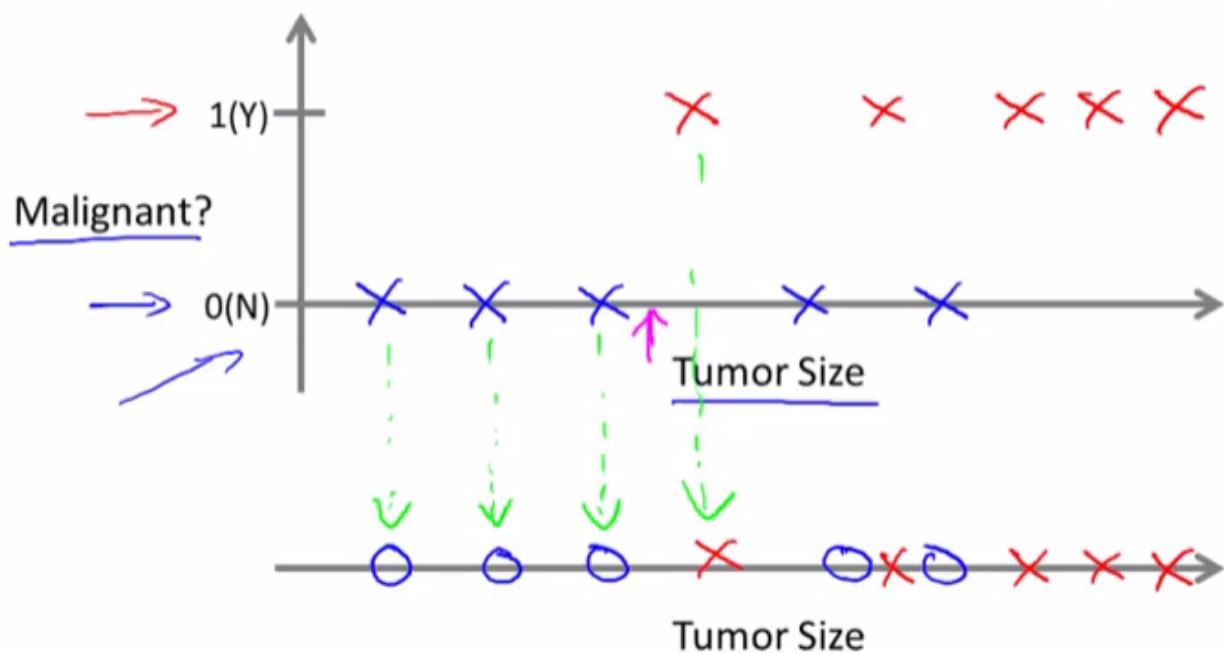
In classification problems there is another way to plot this data. Let's use a slightly different set of symbols to plot this data. So if tumor size is going to be the attribute that we are going to use to predict malignancy

or benignness, we can also draw the data like this.



All we did was we took the data set on top and just mapped it down using different symbols. So instead of drawing crosses, we are now going to draw ○'s for the benign tumors.

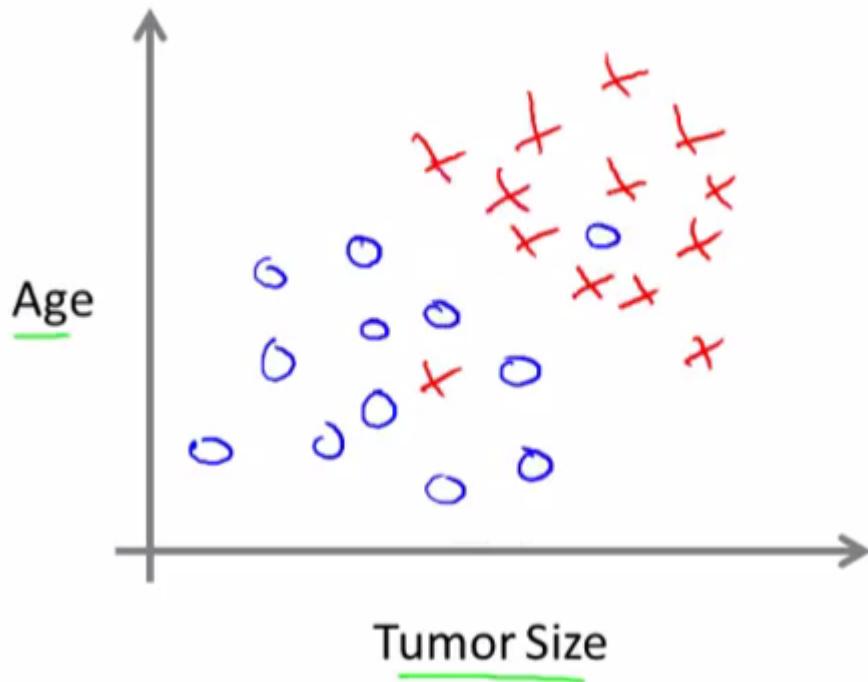
## Breast cancer (malignant, benign)



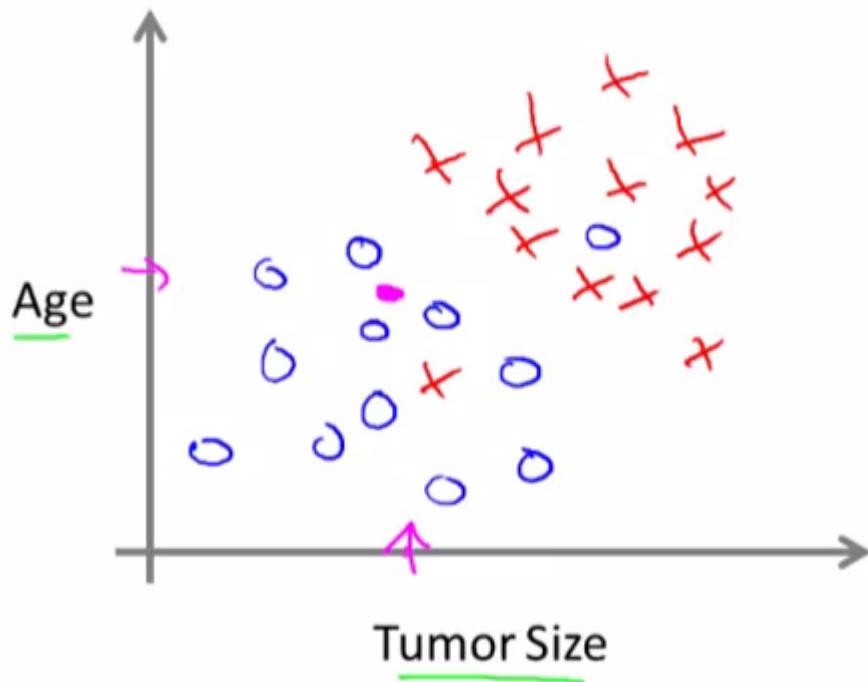
Now, in this example we use only one **feature** or one attribute, mainly, the *tumor size* in order to predict whether the tumor is malignant or benign.

In other machine learning problems we may have more than one feature.

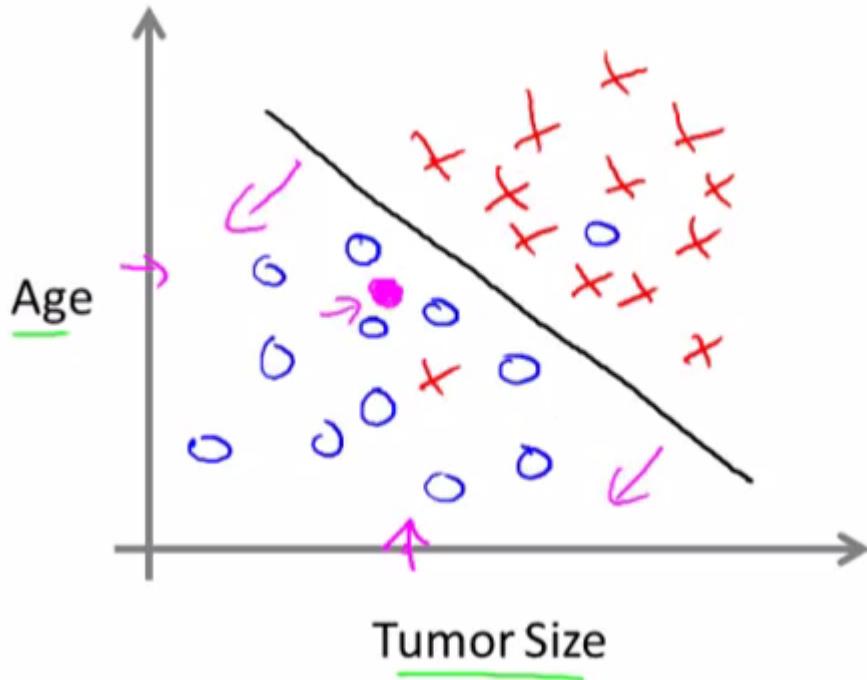
Here's an example. Let's say that instead of just knowing the tumor size, we know both the age of the patients and the tumor size. In that case maybe the data set will look like this.



So, let's say a person who tragically has a tumor. And maybe, their tumor size and age falls around there (rose point):



So given a data set like this, what the learning algorithm might do is throw a straight line through the data to try to separate out the malignant tumors from the benign ones. And with this, hopefully we can decide that the person's tumor falls on this benign side and is therefore more likely to be benign than malignant.



In this example we had **two features**, namely, the age of the patient and the size of the tumor. In other machine learning problems we will often have more features.

Most interesting learning algorithms is a learning algorithm that can deal with, not just two or three or five features, but an **infinite number of features**. So how do you deal with an infinite number of features. How do you even store an infinite number of things on the computer when your computer is gonna run out of memory.

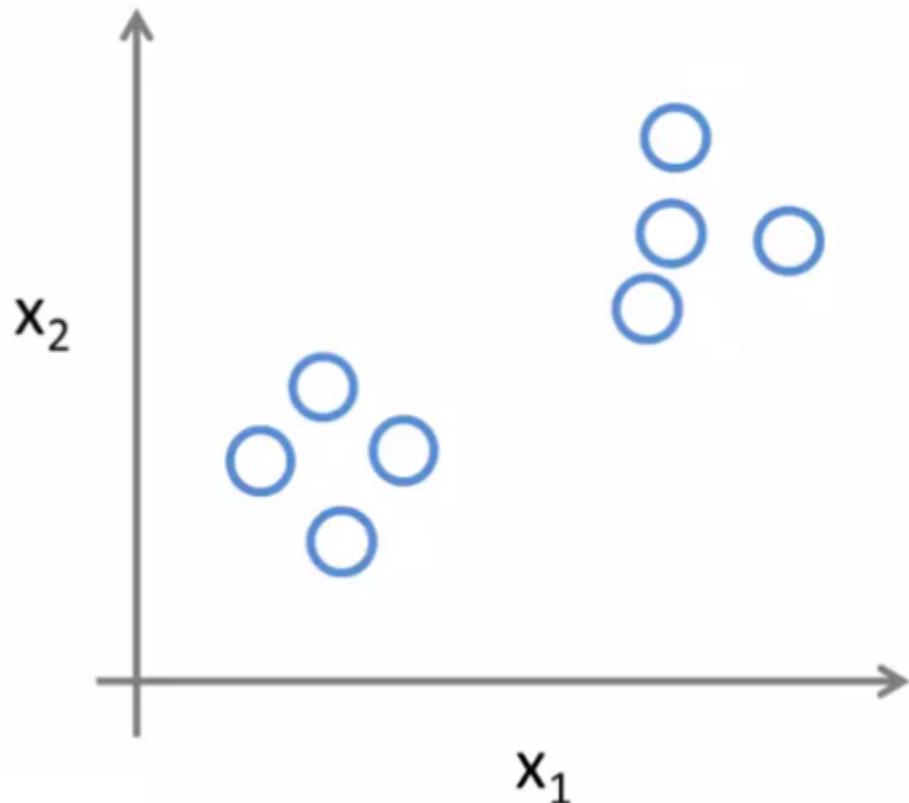
## Unsupervised Learning

The second major type of machine learning problem is called Unsupervised Learning.

The difference between Unsupervised Learning and Supervised Learning is that in Supervised Learning we are told explicitly what is the so-called right answers (data are labeled).

In Unsupervised Learning, we're given data that doesn't have any labels or that all has the same label or really no labels. Like in this example:

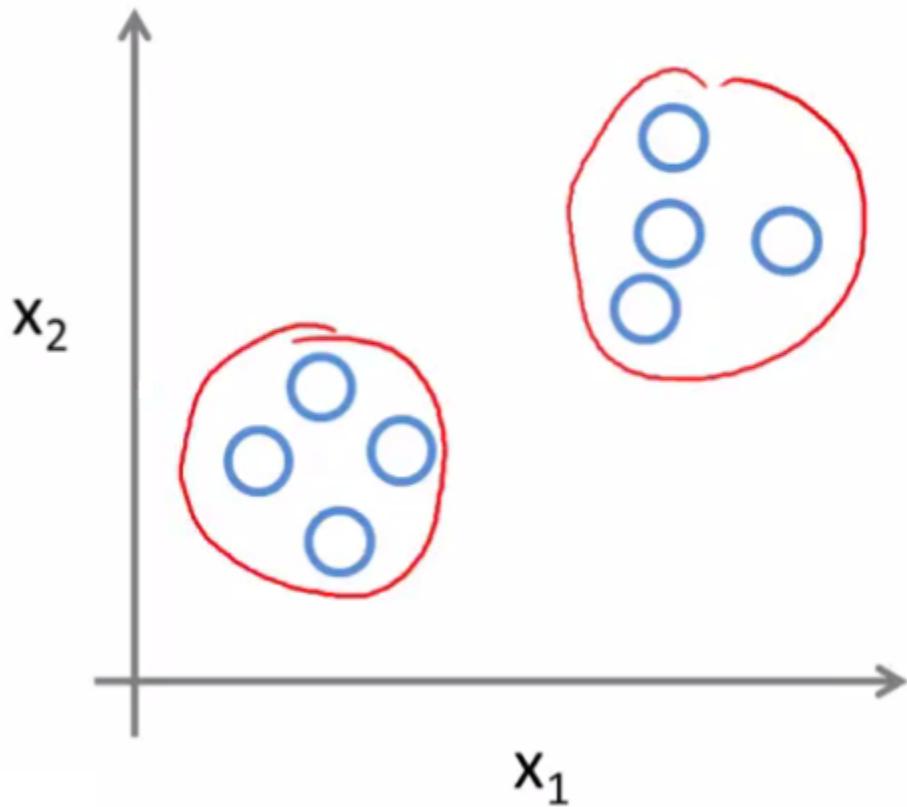
# Unsupervised Learning



So we're given the data set and we're not told what to do with it and we're not told what each data point is. Instead we're just told, here is a data set. Can you find some structure in the data?

Given this data set, an Unsupervised Learning algorithm might decide that the data lives in two different clusters.

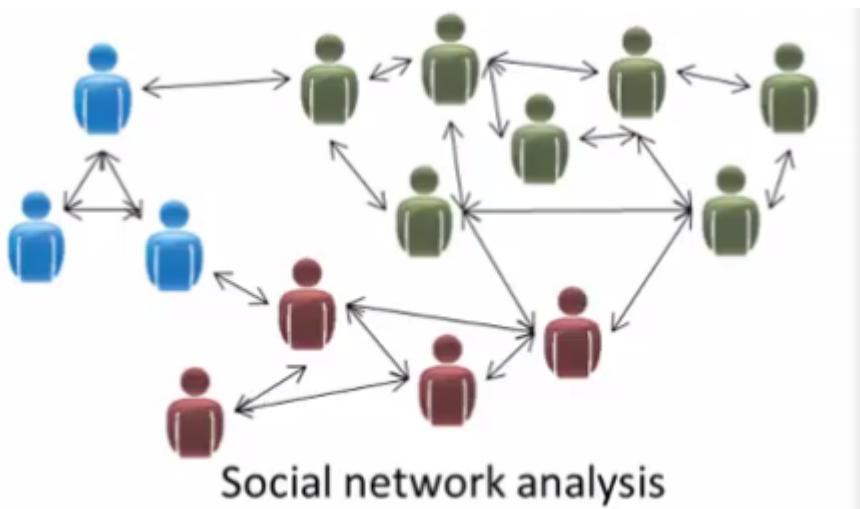
# Unsupervised Learning



This is called a **clustering** algorithm.

Here are two examples where Unsupervised Learning or clustering is used.

Social network analysis:



So given knowledge about which friends you email the most or given your Facebook friends or your Google+ circles, can we automatically identify which are cohesive groups of friends, also which are groups of people that all know each other?

Market segmentation:



## Market segmentation

Many companies have huge databases of customer information. So, can you look at this customer data set and automatically discover market segments and automatically group your customers into different market segments so that you can automatically and more efficiently sell or market your different market segments together?

This is Unsupervised Learning because we have all this customer data, but we don't know in advance what are the market segments and for the customers in our data set, we don't know in advance who is in market segment one, who is in market segment two, and so on. But we have to let the algorithm discover all this just from the data.

# **Part I**

# **Supervised Learning**



# Regression



# Chapter 1

## Linear Regression

### 1.1 Notation

In general, we will let  $x_{ij}$  represent the value of the  $j$ th variable for the  $i$ th observation, where  $i = 1, 2, \dots, n$  and  $j = 1, 2, \dots, p$ . We will use  $i$  to index the samples or observations (from 1 to  $n$ ) and  $j$  will be used to index the variables (or features) (from 1 to  $p$ ). We let  $\mathbf{X}$  denote a  $n \times p$  matrix whose  $(i, j)$ th element is  $x_{ij}$ . That is,

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{np} \end{pmatrix}$$

Note that it is useful to visualize  $\mathbf{X}$  as a spreadsheet of numbers with  $n$  rows and  $p$  columns. We will write the rows of  $\mathbf{X}$  as  $x_1, x_2, \dots, x_n$ . Here  $x_i$  is a vector of length  $p$ , containing the  $p$  variable measurements for the  $i$ th observation. That is,

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}$$

(Vectors are by default represented as columns.)

We will write the columns of  $\mathbf{X}$  as  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$ . Each is a vector of length  $n$ . That is,

$$\mathbf{x}_j = \begin{pmatrix} \mathbf{x}_{1j} \\ \mathbf{x}_{2j} \\ \vdots \\ \mathbf{x}_{nj} \end{pmatrix}$$

Using this notation, the matrix  $\mathbf{X}$  can be written as

$$\mathbf{X} = (\mathbf{x}_1 \mathbf{x}_2 \dots \mathbf{x}_p)$$

or

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}$$

The  $T$  notation denotes the transpose of a matrix or vector.

We use  $y_i$  to denote the  $i$ th observation of the variable on which we wish to make predictions. We write the set of all  $n$  observations in vector form as

$$\mathbf{y} = \begin{pmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{pmatrix}$$

Then the observed data consists of  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , where each  $x_i$  is a vector of length  $p$ . (If  $p = 1$ , then  $x_i$  is simply a scalar).

## 1.2 Model Representation

Let's consider the example about predicting housing prices. We're going to use this data set as an example,

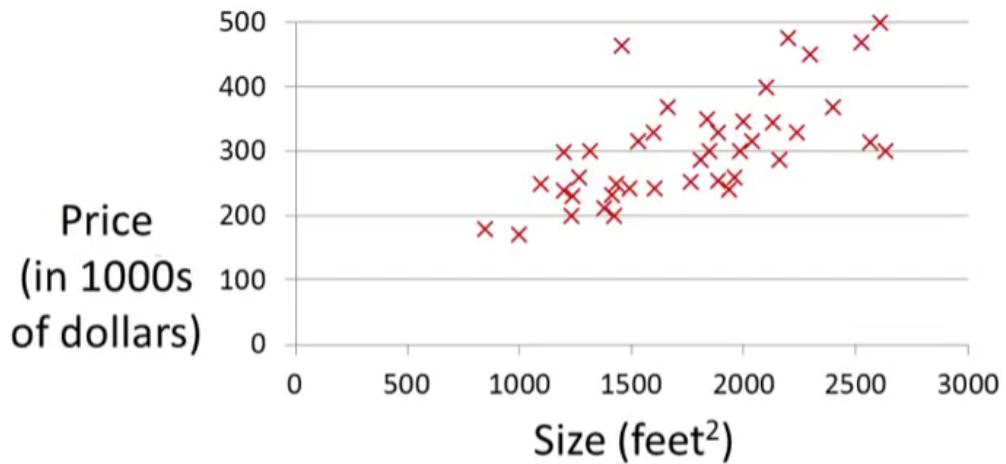


Figure 1.1:

Suppose that there is a person trying to sell a house of size 1250 square feet and he wants to know how much he might be able to sell the house for. One thing we could do is fit a model. Maybe fit a straight line to this data. Looks something like this,

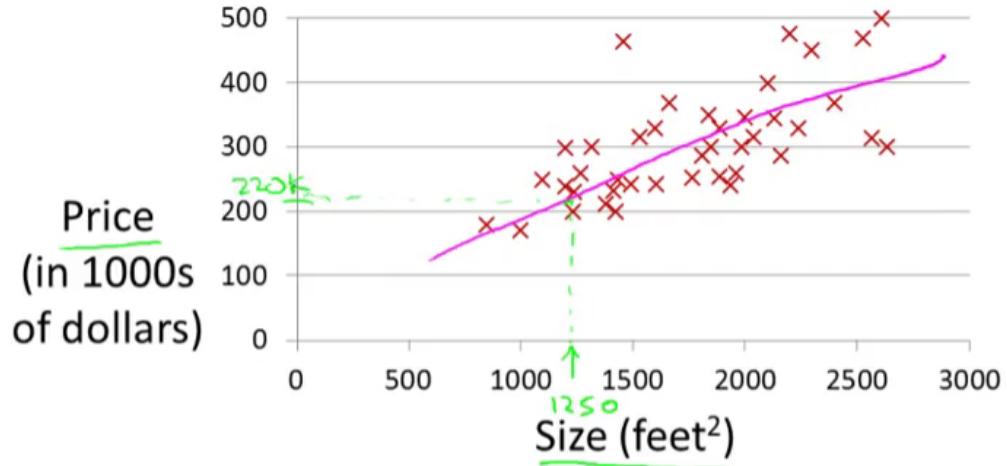


Figure 1.2:

and based on that, maybe he can sell the house for around \$220,000. Recall that this is an example of a supervised learning algorithm. And it's supervised learning because we're given the "right answer" for each of our examples. More precisely, this is an example of a regression problem where the term regression refers to the fact that we are predicting a real-valued output namely the price.

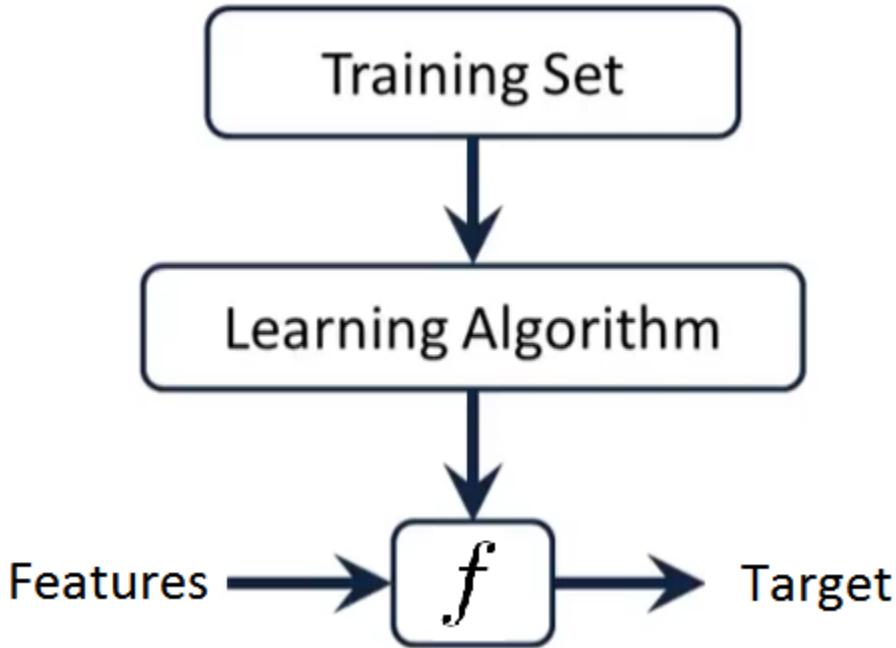
More formally, in supervised learning, we have a data set and this data set is called a **training set**. So for housing prices example, we have a training set of different housing prices and our job is to learn from this data how to predict prices of the houses.

Let's define some notation from this data set:

- The size of the house is the input variable.
- The house price is the output variable.
- The input variables are typically denoted using the variable symbol  $X$ ,
- The inputs go by different names, such as *predictors*, *independent variables*, *features*, or sometimes just *variables*.
- The output variable is often called the *response*, *dependent variable* or *target*, and is typically denoted using the symbol  $Y$ .
- $(x_i, y_i)$  is the  $i$ th training example.
- The set of  $\{(x_i, y_i)\}$  is the training set.
- $n$  is the number of training examples.

So here's how this supervised learning algorithm works. Suppose that we observe a quantitative response  $Y$  and  $p$  different predictors,  $X_1, X_2, \dots, X_p$ . We assume that there is some relationship between  $Y$  and  $X = (X_1, X_2, \dots, X_p)$ , which can be written in the very general form

$$Y = f(X) + \epsilon$$



Here  $f$  is some fixed but unknown function of  $X_1, X_2, \dots, X_p$ , and  $\epsilon$  is a random error term, which is independent of  $X$  and has mean zero. The  $f$  function is also called *hypothesis* in Machine Learning. In general, the function  $f$  may involve more than one input variable. In essence, Supervised Learning refers to a set of approaches for estimating  $f$ .

### 1.3 Why Estimate $f$ ?

There are two main reasons that we may wish to estimate  $f$ : *prediction* and *inference*.

#### Prediction

In many situations, a set of inputs  $X$  are readily available, but the output  $Y$  cannot be easily obtained. In this setting, since the error term averages to zero, we can predict  $Y$  using

$$\hat{Y} = \hat{f}(X)$$

where  $\hat{f}$  represents our estimate for  $f$ , and  $\hat{Y}$  represents the resulting prediction for  $Y$ . Like in the example above about predicting housing prices.

We can measure the accuracy of  $\hat{Y}$  by using a **cost function**. In the regression models, the most commonly-used measure is the *mean squared error* (MSE), given by

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

## Inference

We are often interested in understanding the way that  $Y$  is affected as  $X_1, X_2, \dots, X_p$  change. In this situation we wish to estimate  $f$ , but our goal is not necessarily to make predictions for  $Y$ . We instead want to understand the relationship between  $X$  and  $Y$ , or more specifically, to understand how  $Y$  changes as a function of  $X_1, X_2, \dots, X_p$ . In this case, one may be interested in answering the following questions:

- Which predictors are associated with the response?
- What is the relationship between the response and each predictor?
- Can the relationship between  $Y$  and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

## 1.4 Simple Linear Regression Model

*Simple linear regression* is a very straightforward approach for predicting a quantitative response  $Y$  on the basis of a single predictor variable  $X$ . It assumes that there is approximately a linear relationship between  $X$  and  $Y$ . Mathematically, we can write this linear relationship as

$$\begin{aligned} Y &= \beta_0 + \beta_1 X + \epsilon \\ Y &\approx \beta_0 + \beta_1 X \end{aligned}$$

where  $\beta_0$  and  $\beta_1$  are two unknown constants that represent the *intercept* and *slope*, also known as *coefficients* or *parameters*, and  $\epsilon$  is the error term.

Given some estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$  for the model coefficients, we predict future inputs  $x$  using

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

where  $\hat{y}$  indicates a prediction of  $Y$  on the basis of  $X = x$ . The *hat* symbol,  $\hat{\cdot}$ , denotes an estimated value.

## 1.5 Estimating the Coefficients

Let  $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$  be the prediction for  $Y$  based on the  $i$ th value of  $X$ . Then  $e_i = y_i - \hat{y}_i$  represents the  $i$ th *residual*.

We define the *Residual Sum of Squares* (RSS)<sup>1</sup> as

$$\begin{aligned} RSS &= e_1^2 + e_2^2 + \dots + e_n^2 \\ &= \sum_{i=1}^n e_i^2 \end{aligned}$$

or equivalently as

$$\begin{aligned} RSS &= (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \dots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2 \\ &= \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 \end{aligned}$$

---

<sup>1</sup>Also known as **SSE**: Sum of Squared Errors.

The *least squares* approach chooses  $\hat{\beta}_0$  and  $\hat{\beta}_1$  to minimize the RSS. The minimizing values can be shown to be<sup>2</sup>

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{s_{xy}}{s_x^2} = (s_x^2)^{-1} s_{xy}$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where:

- $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$  is the *sample mean*.
- $s_x^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$  is the *sample variance*. The sample standard deviation is  $s_x = \sqrt{s_x^2}$ .
- $s_{xy} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$  is the *sample covariance*. It measures the degree of linear association between  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$ . Once scaled by  $s_x s_y$ , it gives the *sample correlation coefficient*,  $r_{xy} = \frac{s_{xy}}{s_x s_y}$ .



1- To find the optimal estimates for  $\beta_0$  and  $\beta_1$  we need a choice-criterion. In the case of the *least squares* approach (more precisely, the *ordinary least squares OLS*) this criterion is the residual sum of squares *RSS*: we calculate  $\beta_0$  and  $\beta_1$  that minimise the *RSS*.

2- Minimizing the *RSS* function requires to calculate the first order derivatives with respect to  $\beta_0$  and  $\beta_1$  and set them to zero.

3- Click here and watch the video to understand more about the residuals and least squares.

4- Click here to see the influence of the distance employed in the sum of squares. Try to minimize the sum of squares for the different datasets. The choices of intercept and slope that minimize the sum of squared distances for a kind of distance are not the optimal for a different kind of distance.

## 1.6 Assessing the Accuracy of the Coefficient Estimates

The standard error of an estimator reflects how it varies under repeated sampling. We have

$$\begin{aligned} \text{SE}(\hat{\beta}_1)^2 &= \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \\ \text{SE}(\hat{\beta}_0)^2 &= \sigma^2 \left[ \frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right] \end{aligned}$$

where  $\sigma^2 = \text{Var}(\epsilon)$

In general,  $\sigma^2$  is unknown, but can be estimated from the data. The estimate of  $\sigma$  is known as the *residual standard error*, and is given by

$$\text{RSE} = \sqrt{\frac{\text{RSS}}{(n - 2)}}$$

These standard errors can be used to compute *confidence intervals*. A 95% confidence interval is defined as a range of values such that with 95% probability, the range will contain the true unknown value of the parameter. It has the form

---

<sup>2</sup>They are unique and always exist. They can be obtained by solving  $\frac{\partial}{\partial \beta_0} \text{RSS}(\beta_0, \beta_1) = 0$  and  $\frac{\partial}{\partial \beta_1} \text{RSS}(\beta_0, \beta_1) = 0$ .

$$\hat{\beta}_1 \pm 2 \cdot \text{SE}(\hat{\beta}_1)$$

That is, there is approximately a 95% chance that the interval

$$\left[ \hat{\beta}_1 - 2 \cdot \text{SE}(\hat{\beta}_1), \hat{\beta}_1 + 2 \cdot \text{SE}(\hat{\beta}_1) \right]$$

will contain the true value of  $\beta_1$ . Similarly, a confidence interval for  $\beta_0$  approximately takes the form

$$\hat{\beta}_0 \pm 2 \cdot \text{SE}(\hat{\beta}_0)$$

## Hypothesis testing

Standard errors can also be used to perform *hypothesis tests* on the coefficients. The most common hypothesis test involves testing the *null hypothesis* of

$$H_0 : \text{There is no relationship between } X \text{ and } Y$$

versus the *alternative hypothesis*

$$H_1 : \text{There is some relationship between } X \text{ and } Y$$

Mathematically, this corresponds to testing

$$H_0 : \beta_1 = 0$$

versus

$$H_1 : \beta_1 \neq 0$$

since if  $\beta_1 = 0$  then the simple linear regression model reduces to  $Y = \beta_0 + \epsilon$ , and  $X$  is not associated with  $Y$ .

To test the null hypothesis  $H_0$ , we compute a ***t-statistic***, given by

$$t = \frac{\hat{\beta}_1 - 0}{\text{SE}(\hat{\beta}_1)}$$

This will have a *t-distribution* (*Student*) with  $n - 2$  degrees of freedom, assuming  $\beta_1 = 0$ .

Using statistical software, it is easy to compute the probability of observing any value equal to  $|t|$  or larger. We call this probability the ***p-value***.

If p-value is small enough (typically under 0.01 (1% error) or 0.05 (5% error)) we reject the null hypothesis, that is we declare a relationship to exist between  $X$  and  $Y$ .

## 1.7 ANOVA and model fit

### 1.7.1 ANOVA

In this section we will see how the variance of  $Y$  is decomposed into two parts, each one corresponding to the regression and to the error, respectively. This decomposition is called the *ANalysis Of VAriance* (ANOVA).

Before explaining ANOVA, it is important to recall an interesting result: *the mean of the fitted values  $\hat{Y}_1, \dots, \hat{Y}_n$  is the mean of  $Y_1, \dots, Y_n$* . This is easily seen if we plug-in the expression of  $\hat{\beta}_0$ :

$$\frac{1}{n} \sum_{i=1}^n \hat{Y}_i = \frac{1}{n} \sum_{i=1}^n (\hat{\beta}_0 + \hat{\beta}_1 X_i) = \hat{\beta}_0 + \hat{\beta}_1 \bar{X} = (\bar{Y} - \hat{\beta}_1 \bar{X}) + \hat{\beta}_1 \bar{X} = \bar{Y}.$$

The ANOVA decomposition considers the following measures of variation related with the response:

- $SST = TSS = \sum_{i=1}^n (Y_i - \bar{Y})^2$ , the **Total Sum of Squares**. This is the *total variation* of  $Y_1, \dots, Y_n$ , since  $SST = ns_y^2$ , where  $s_y^2$  is the sample variance of  $Y_1, \dots, Y_n$ .
- $SSR = ESS = \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$ , the **Regression Sum of Squares** or **Explained Sum of Squares**<sup>3</sup>. This is the variation explained by the regression line, that is, *the variation from  $\bar{Y}$  that is explained by the estimated conditional mean  $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$* .  $SSR = ns_{\hat{y}}^2$ , where  $s_{\hat{y}}^2$  is the sample variance of  $\hat{Y}_1, \dots, \hat{Y}_n$ .
- $SSE = RSS = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$ , the **Sum of Squared Errors** or **Residual Sum of Squares**<sup>4</sup>. Is the variation around the conditional mean. Recall that  $SSE = \sum_{i=1}^n \hat{\varepsilon}_i^2 = (n-2)\hat{\sigma}^2$ , where  $\hat{\sigma}^2$  is the sample variance of  $\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_n$ .

The ANOVA decomposition is

$$\underbrace{SST}_{\text{Variation of } Y'_i s} = \underbrace{SSR}_{\text{Variation of } \hat{Y}'_i s} + \underbrace{SSE}_{\text{Variation of } \hat{\varepsilon}'_i s}$$

The graphical interpretation of this equation is shown in the following figures.

---

<sup>3</sup>Recall that SSR is different from RSS (Residual) Sum of Squares

<sup>4</sup>Recall that SSE and RSS (for  $(\hat{\beta}_0, \hat{\beta}_1)$ ) are just different names for referring to the same quantity:  $SSE = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2 = RSS(\hat{\beta}_0, \hat{\beta}_1)$ .

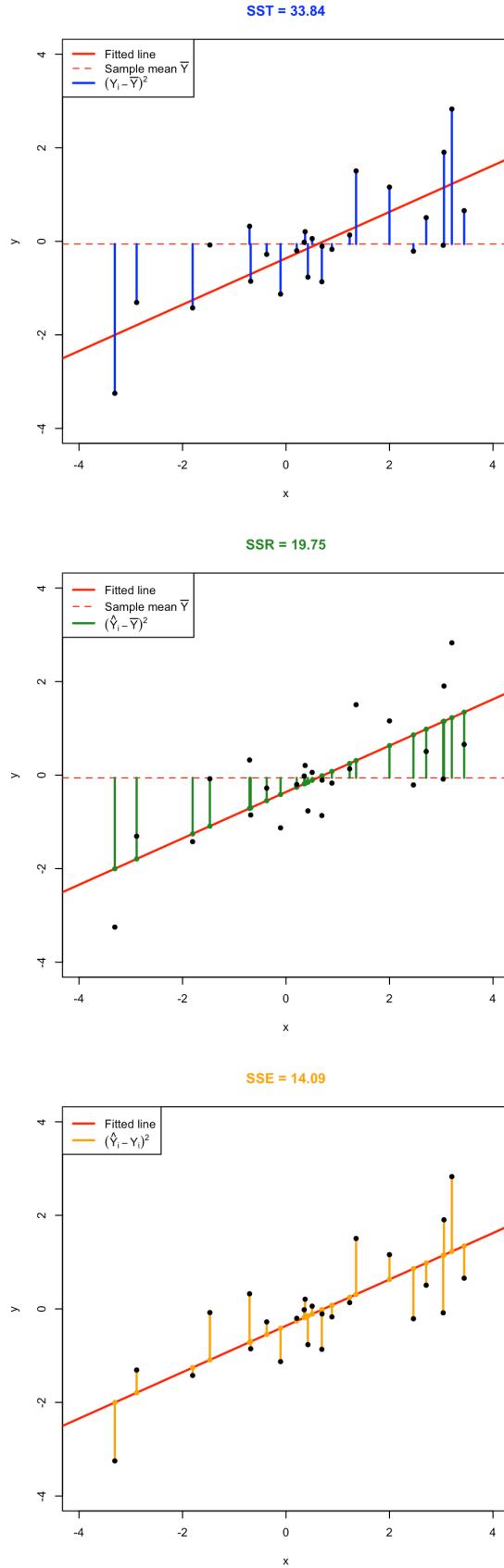


Figure 1.3: Visualization of the ANOVA decomposition. SST measures the variation of  $Y_1, \dots, Y_n$  with respect to  $\bar{Y}$ . SST measures the variation with respect to the conditional means,  $\hat{\beta}_0 + \hat{\beta}_1 X_i$ . SSE collects the variation of the residuals.



Below the ANOVA decomposition and its dependence on  $\sigma^2$  and  $\hat{\sigma}^2$ . Application is also available here.

Note that the **animation** will not be displayed the first time it is browsed (The reason is because it is hosted at [https](https://) websites with auto-signed SSL certificates). **To see it**, click on the link above. You will get a warning from your browser saying that “*Your connection is not private*”. Click in “*Advanced*” and **allow an exception** in your browser. The next time the animation will show up correctly.

The ANOVA table summarizes the decomposition of the variance. Here is given in the layout employed by R.

Degrees of freedom	Sum Squares	Mean Squares	F-value	p-value
Predictor	$SSR = \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$	$MSR = \frac{SSR}{1}$	$\frac{SSR/1}{SSE/(n-2)}$	$p$
Residuals 2	$SSE = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$	$MSE = \frac{SSE}{n-2}$		
Total $n - 1$	$SST = \sum_{i=1}^n (Y_i - \bar{Y})^2$			

The `anova` function in R takes a model as an input and returns the ANOVA table.

The “F-value” of the ANOVA table represents the value of the F-statistic  $\frac{SSR/1}{SSE/(n-2)}$ . This statistic is employed to test

$$H_0 : \beta_1 = 0 \quad \text{vs.} \quad H_1 : \beta_1 \neq 0,$$

that is, the hypothesis of no linear dependence of  $Y$  on  $X$ . The result of this test is completely equivalent to the  $t$ -test for  $\beta_1$  that we saw previously in the Hypothesis testing (this is something *specific for simple linear regression* – the F-test will not be equivalent to the  $t$ -test for  $\beta_1$  in the Multiple Linear Regression).

It happens that

$$F = \frac{SSR/1}{SSE/(n-2)} \stackrel{H_0}{\sim} F_{1,n-2},$$

where  $F_{1,n-2}$  is the *Snedecor's F distribution*<sup>5</sup> with 1 and  $n - 2$  degrees of freedom.

If  $H_0$  is true, then  $F$  is expected to be *small* since SSR will be close to zero. The p-value of this test is the same as the p-value of the  $t$ -test for  $H_0 : \beta_1 = 0$ .

## 1.7.2 The $R^2$ Statistic

To calculate  $R^2$ , we use the formula

<sup>5</sup>The  $F_{n,m}$  distribution arises as the quotient of two independent random variables  $\chi_n^2$  and  $\chi_m^2$ ,  $\frac{\chi_n^2/n}{\chi_m^2/m}$ .

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

where  $\text{TSS} = \sum(y_i - \bar{y})^2$  is the *total sum of squares*.

$R^2$  measures the *proportion of variability in  $Y$  that can be explained using  $X$* . An  $R^2$  statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression. A number near 0 indicates that the regression did not explain much of the variability in the response; this might occur because the linear model is wrong, or the inherent error  $\sigma^2$  is high, or both.

It can be shown that in this simple linear linear regression setting that  $R^2 = r^2$ , where  $r$  is the correlation between  $X$  and  $Y$ :

$$r = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$



$R^2$  does not measure the correctness of a linear model but its **usefulness** (for prediction, for *explaining the variance of  $Y$* ), assuming the model is correct.

Trusting blindly the  $R^2$  can lead to catastrophic conclusions, since the model may not be correct.

So remember:



A large  $R^2$  means *nothing* if the **assumptions of the model do not hold**.  $R^2$  is the proportion of variance of  $Y$  explained by  $X$ , but, of course, *only when the linear model is correct*.



# PW 1

## 1.8 Some R basics

### 1.8.1 Basic Commands

R uses functions to perform operations. To run a function called `funcname`, we type `funcname(input1, input2)`, where the inputs (or arguments) `input1` and `input2` tell R how to run the function. A function can have any number of inputs. For example, to create a vector of numbers, we use the function `c()` (for *concatenate*).

```
x <- c(1,3,2,5)
x
#ans> [1] 1 3 2 5
```

Note that the `>` is not part of the command; rather, it is printed by R to indicate that it is ready for another command to be entered. We can also save things using `=` rather than `<-`. Note that the answer in the code above is followed by `#ans>` while in the R console it is not.

```
x = c(1,6,2)
x
#ans> [1] 1 6 2
y = c(1,4,3)
length(x)
#ans> [1] 3
length(y)
#ans> [1] 3
x+y
#ans> [1] 2 10 5
```

Hitting the *up arrow* multiple times will display the previous commands, which can then be edited. This is useful since one often wishes to repeat a similar command.

The `ls()` function allows us to look at a list of all of the objects, such as `ls()` as data and functions, that we have saved so far. The `rm()` function can be used to delete any object that we don't want.

```
ls()
#ans> [1] "x" "y"
rm(x)
ls()
#ans> [1] "y"
```

### 1.8.2 Vectors

```
# A handy way of creating sequences is the operator :
# Sequence from 1 to 5
1:5
#ans> [1] 1 2 3 4 5

# Storing some vectors
vec <- c(-4.12, 0, 1.1, 1, 3, 4)
vec
#ans> [1] -4.12 0.00 1.10 1.00 3.00 4.00

# Entry-wise operations
vec + 1
#ans> [1] -3.12 1.00 2.10 2.00 4.00 5.00
vec^2
#ans> [1] 16.97 0.00 1.21 1.00 9.00 16.00

# If you want to access a position of a vector, use [position]
vec[6]
#ans> [1] 4

# You also can change elements
vec[2] <- -1
vec
#ans> [1] -4.12 -1.00 1.10 1.00 3.00 4.00

# If you want to access all the elements except a position, use [-position]
vec[-2]
#ans> [1] -4.12 1.10 1.00 3.00 4.00

# Also with vectors as indexes
vec[1:2]
#ans> [1] -4.12 -1.00

# And also
vec[-c(1, 2)]
#ans> [1] 1.1 1.0 3.0 4.0
```



Do the following:

- Create the vector  $x = (1, 7, 3, 4)$ .
- Create the vector  $y = (100, 99, 98, \dots, 2, 1)$ .
- Compute  $x_3 + y_4$  and  $\cos(x_3) + \sin(x_2)e^{-y_2}$ . (Answers: 100,  $-0.9899925$ )
- Set  $x_3 = 0$  and  $y_2 = -1$ . Recompute the previous expressions. (Answers: 97, 2.785875)
- Index  $y$  by  $x + 1$  and store it as  $z$ . What is the output? (Answer:  $z$  is  $c(-1, 93, 100, 96)$ )

### 1.8.3 Matrices, data frames and lists

```
# A matrix is an array of vectors
A <- matrix(1:4, nrow = 2, ncol = 2)
```

```

A
#ans>      [,1] [,2]
#ans> [1,]    1    3
#ans> [2,]    2    4

# Another matrix
B <- matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE)
B
#ans>      [,1] [,2]
#ans> [1,]    1    2
#ans> [2,]    3    4

# Binding by rows or columns
rbind(1:3, 4:6)
#ans>      [,1] [,2] [,3]
#ans> [1,]    1    2    3
#ans> [2,]    4    5    6
cbind(1:3, 4:6)
#ans>      [,1] [,2]
#ans> [1,]    1    4
#ans> [2,]    2    5
#ans> [3,]    3    6

# Entry-wise operations
A + 1
#ans>      [,1] [,2]
#ans> [1,]    2    4
#ans> [2,]    3    5
A * B
#ans>      [,1] [,2]
#ans> [1,]    1    6
#ans> [2,]    6   16

# Accessing elements
A[2, 1] # Element (2, 1)
#ans> [1] 2
A[1, ] # First row
#ans> [1] 1 3
A[, 2] # Second column
#ans> [1] 3 4

# A data frame is a matrix with column names
# Useful when you have multiple variables
myDf <- data.frame(var1 = 1:2, var2 = 3:4)
myDf
#ans>   var1 var2
#ans> 1     1     3
#ans> 2     2     4

# You can change names
names(myDf) <- c("newname1", "newname2")
myDf
#ans> newname1 newname2

```

```

#ans> 1      1      3
#ans> 2      2      4

# The nice thing is that you can access variables by its name with the $ operator
myDf$newname1
#ans> [1] 1 2

# And create new variables also (it has to be of the same
# length as the rest of variables)
myDf$myNewVariable <- c(0, 1)
myDf
#ans>  newname1 newname2 myNewVariable
#ans> 1      1      3          0
#ans> 2      2      4          1

# A list is a collection of arbitrary variables
myList <- list(vec = vec, A = A, myDf = myDf)

# Access elements by names
myList$vec
#ans> [1] -4.12 -1.00  1.10  1.00  3.00  4.00
myList$A
#ans> [,1] [,2]
#ans> [1,]   1   3
#ans> [2,]   2   4
myList$myDf
#ans>  newname1 newname2 myNewVariable
#ans> 1      1      3          0
#ans> 2      2      4          1

# Reveal the structure of an object
str(myList)
#ans> List of 3
#ans> $ vec : num [1:6] -4.12 -1 1.1 1 3 4
#ans> $ A   : int [1:2, 1:2] 1 2 3 4
#ans> $ myDf:'data.frame': 2 obs. of  3 variables:
#ans>   ..$ newname1    : int [1:2] 1 2
#ans>   ..$ newname2    : int [1:2] 3 4
#ans>   ..$ myNewVariable: num [1:2] 0 1
str(myDf)
#ans> 'data.frame': 2 obs. of  3 variables:
#ans> $ newname1    : int 1 2
#ans> $ newname2    : int 3 4
#ans> $ myNewVariable: num 0 1

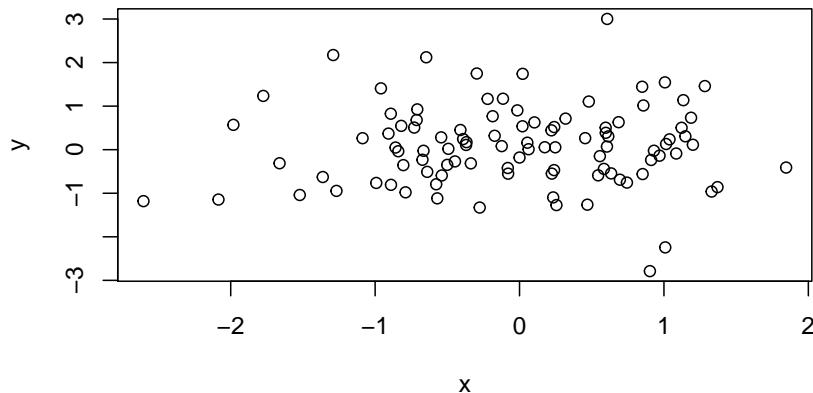
# A less lengthy output
names(myList)
#ans> [1] "vec"   "A"     "myDf"

```

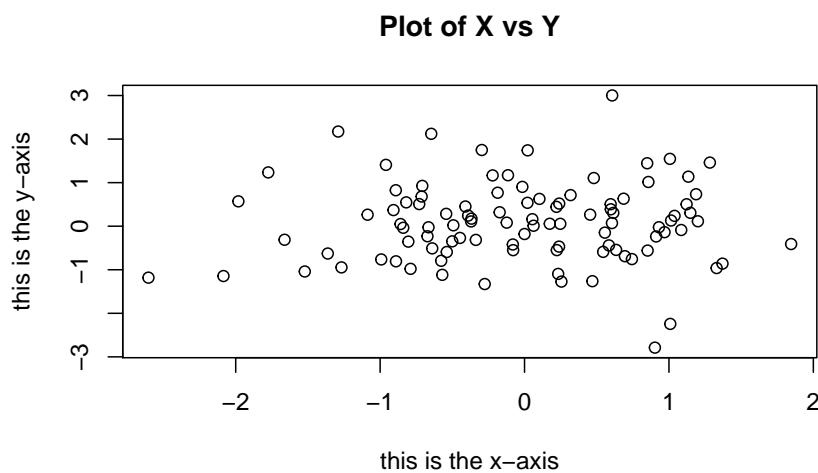
### 1.8.4 Graphics

The `plot()` function is the primary way to plot data in R . For instance, `plot(x,y)` produces a scatterplot of the numbers in `x` versus the numbers in `y`. There are many additional options that can be passed in to the `plot()` function. For example, passing in the argument `xlab` will result in a label on the `x-axis`. To find out more information about the `plot()` function, type `?plot`.

```
x=rnorm(100)
# The rnorm() function generates a vector of random normal variables,
# rnorm() with first argument n the sample size. Each time we call this
# function, we will get a different answer.
y=rnorm(100)
plot(x,y)
```



```
# with titles
plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis",
main="Plot of X vs Y")
```



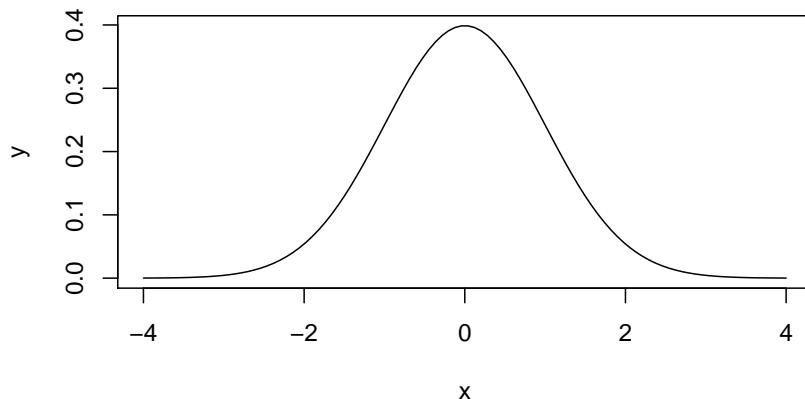
### 1.8.5 Distributions

```
# R allows to sample [r], compute density/probability mass [d],
# compute distribution function [p] and compute quantiles [q] for several
# continuous and discrete distributions. The format employed is [rdpq]name,
# where name stands for:
# - norm -> Normal
# - unif -> Uniform
# - exp -> Exponential
# - t -> Student's t
# - f -> Snedecor's F (Fisher)
# - chisq -> Chi squared
# - pois -> Poisson
# - binom -> Binomial
# More distributions: ?Distributions

# Sampling from a Normal - 100 random points from a N(0, 1)
rnorm(n = 10, mean = 0, sd = 1)
#ans> [1] -0.02861 -0.35848  0.89207  0.00277  0.21199 -1.73837 -0.32862
#ans> [8] -1.45277  0.32805  0.30720

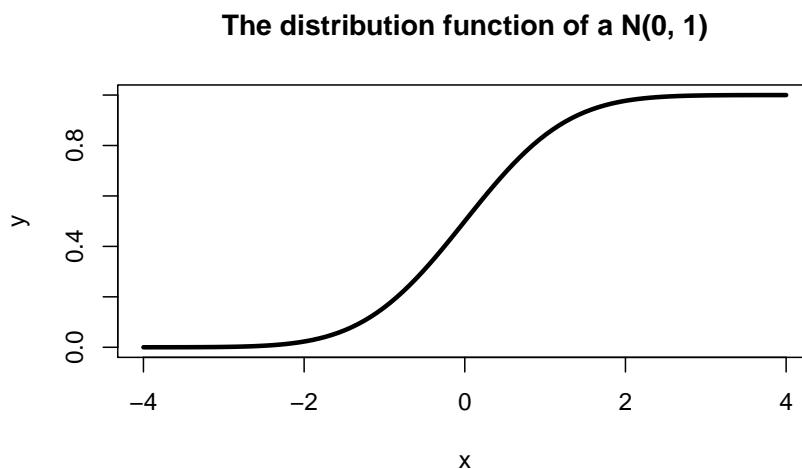
# If you want to have always the same result, set the seed of the random number
# generator
set.seed(45678)
rnorm(n = 10, mean = 0, sd = 1)
#ans> [1]  1.440 -0.720  0.671 -0.422  0.378 -1.667 -0.508  0.443 -1.799 -0.618

# Plotting the density of a N(0, 1) - the Gauss bell
x <- seq(-4, 4, l = 100)
y <- dnorm(x = x, mean = 0, sd = 1)
plot(x, y, type = "l")
```



```
# Plotting the distribution function of a N(0, 1)
x <- seq(-4, 4, l = 100)
y <- pnorm(q = x, mean = 0, sd = 1)
```

```
plot(x, y, type = "l", lwd = 3, main="The distribution function of a N(0, 1)")
```

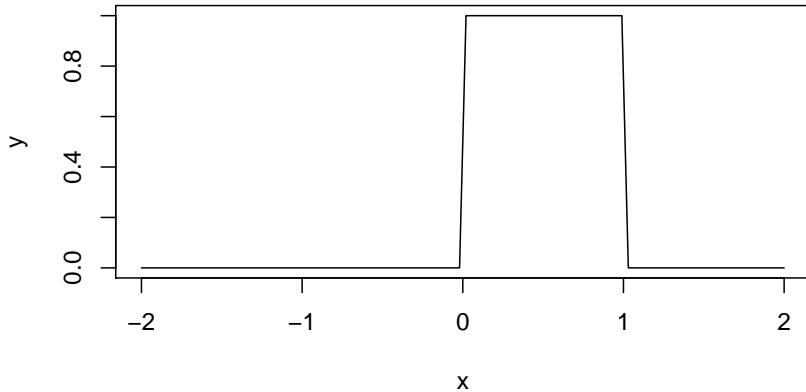


```
# Computing the 95% quantile for a N(0, 1)
qnorm(p = 0.95, mean = 0, sd = 1)
#ans> [1] 1.64

# All distributions have the same syntax: rname(n,...), dname(x,...), dname(p,...)
# and qname(p,...), but the parameters in ... change. Look them in ?Distributions
# For example, here is que same for the uniform distribution

# Sampling from a U(0, 1)
set.seed(45678)
runif(n = 10, min = 0, max = 1)
#ans> [1] 0.9251 0.3340 0.2359 0.3366 0.7489 0.9327 0.3365 0.2246 0.6474 0.0808

# Plotting the density of a U(0, 1)
x <- seq(-2, 2, l = 100)
y <- dunif(x = x, min = 0, max = 1)
plot(x, y, type = "l")
```



```
# Computing the 95% quantile for a U(0, 1)
qunif(p = 0.95, min = 0, max = 1)
#ans> [1] 0.95
```

- Do the following:
- Compute the 90%, 95% and 99% quantiles of a  $F$  distribution with  $\text{df1} = 1$  and  $\text{df2} = 5$ .  
(Answer: `c(4.060420, 6.607891, 16.258177)`)
  - Sample 100 points from a Poisson with  $\lambda = 5$ .
  - Plot the density of a  $t$  distribution with  $\text{df} = 1$  (use a sequence spanning from  $-4$  to  $4$ ). Add lines of different colors with the densities for  $\text{df} = 5$ ,  $\text{df} = 10$ ,  $\text{df} = 50$  and  $\text{df} = 100$ .

### 1.8.6 Working directory

Your *working directory* is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory.

To set your working directory within RStudio you can go to **Tools / Set working directory**, or use the command `setwd()`, we put the complete path of the directory between the brackets, do not forget to put the path into quotation marks "".

To know the actual working directory we use `getwd()`.

### 1.8.7 Loading Data

The `read.table()` function is one of the primary ways to import a data set into R. The help file `?read.table()` contains details about how to use this function. We can use the function `write.table()` to export data.

Next we will show how to load the data set `Auto.data` (Download it from [here](#)).

```
Auto=read.table("Auto.data",header=T,na.strings ="?")
# For this file we needed to tell R that the first row is the
# names of the variables.
```

```
# na.strings tells R that any time it sees a particular character
# or set of characters (such as a question mark), it should be
# treated as a missing element of the data matrix.
```



- If the file is of csv format, we use `read.csv`.
- Always try to look to the file before importing it to R (Open it in a text editor. See for example if the first row contains the variables names, if the columns are separated by , or ; or ..)
- For text editors, I suggest Sublime Text or Atom.

```
dim(Auto) # To see the dimensions of the data set
#ans> [1] 397 9
nrow(Auto) # To see the number of rows
#ans> [1] 397
ncol(Auto) # To see the number of columns
#ans> [1] 9
Auto[1:4,] # The first 4 rows of the data set
#ans>   mpg cylinders displacement horsepower weight acceleration year origin
#ans> 1 18          8            307        130    3504       12.0    70      1
#ans> 2 15          8            350        165    3693       11.5    70      1
#ans> 3 18          8            318        150    3436       11.0    70      1
#ans> 4 16          8            304        150    3433       12.0    70      1
#ans>           name
#ans> 1 chevrolet chevelle malibu
#ans> 2 buick skylark 320
#ans> 3 plymouth satellite
#ans> 4 amc rebel sst
```

```
# Once the data are loaded correctly, we can use names()
# to check the variable names.
names(Auto)
#ans> [1] "mpg"           "cylinders"     "displacement" "horsepower"
#ans> [5] "weight"        "acceleration"  "year"         "origin"
#ans> [9] "name"
```



Take a look at this (very) short introduction to R. It can be useful.

## 1.9 Regression

### 1.9.1 The `lm` function

We are going to employ the EU dataset. The EU dataset contains 28 rows with the member states of the European Union (Country), the number of seats assigned under different years (Seats2011, Seats2014), the Cambridge Compromise apportionment (CamCom2011), and the countries population (Population2010,Population2013). Click [here](#) to download the EU dataset.

```
# Load the dataset, when we load an .RData using load()
# function we do not attribute it to a name like we did
# when we used read.table() or when we use read.csv()
```

```
load("EU.RData")
```



There are two ways to tell R where is the file you want to load/use/import or where to save a file when you write/export/save :

1. write the complete path of the files.
2. set a working directory and put the files in it.

```
# lm (for linear model) has the syntax:
# lm(formula = response ~ predictor, data = data)
# The response is the y in the model. The predictor is x.
# For example (after loading the EU dataset)
mod <- lm(formula = Seats2011 ~ Population2010, data = EU)

# We have saved the linear model into mod, which now contains all the output of lm
# You can see it by typing
mod
#ans>
#ans> Call:
#ans> lm(formula = Seats2011 ~ Population2010, data = EU)
#ans>
#ans> Coefficients:
#ans>   (Intercept) Population2010
#ans>       7.91e+00      1.08e-06

# mod is indeed a list of objects whose names are
names(mod)
#ans> [1] "coefficients"    "residuals"        "effects"          "rank"
#ans> [5] "fitted.values"   "assign"           "qr"               "df.residual"
#ans> [9] "na.action"        "xlevels"          "call"              "terms"
#ans> [13] "model"

# We can access these elements by $
# For example
mod$coefficients
#ans>   (Intercept) Population2010
#ans>       7.91e+00      1.08e-06

# The residuals
mod$residuals
#ans>      Germany     France United Kingdom      Italy      Spain
#ans>      2.8675    -3.7031     -1.7847  0.0139   -3.5084
#ans>      Poland      Romania     Netherlands  Greece  Belgium
#ans>      1.9272     1.9434      0.2142  1.8977   2.3994
#ans>      Portugal   Czech Republic    Hungary  Sweden  Austria
#ans>      2.6175     2.7587      3.2898  2.0163   2.0575
#ans>      Bulgaria    Denmark    Slovakia  Finland Ireland
#ans>      1.9328    -0.8790     -0.7606 -0.6813  -0.7284
#ans>      Lithuania   Latvia    Slovenia Estonia  Cyprus
#ans>      0.4998    -1.3347     -2.1175 -3.3552  -2.7761
#ans>      Luxembourg  Malta      -2.3553
```

```

# The fitted values
mod$fitted.values
#ans>      Germany      France United Kingdom      Italy      Spain
#ans> 96.13        77.70       74.78      72.99      57.51
#ans> Poland      Romania     Netherlands      Greece      Belgium
#ans> 49.07        31.06       25.79      20.10      19.60
#ans> Portugal Czech Republic      Hungary      Sweden      Austria
#ans> 19.38        19.24       18.71      17.98      16.94
#ans> Bulgaria     Denmark     Slovakia      Finland      Ireland
#ans> 16.07        13.88       13.76      13.68      12.73
#ans> Lithuania    Latvia      Slovenia      Estonia      Cyprus
#ans> 11.50        10.33       10.12      9.36       8.78
#ans> Luxembourg   Malta
#ans> 8.45         8.36

# Summary of the model
sumMod <- summary(mod)
sumMod
#ans>
#ans> Call:
#ans> lm(formula = Seats2011 ~ Population2010, data = EU)
#ans>
#ans> Residuals:
#ans>   Min     1Q Median     3Q    Max
#ans> -3.703 -1.951  0.014  1.980  3.290
#ans>
#ans> Coefficients:
#ans>             Estimate Std. Error t value Pr(>|t|)    
#ans> (Intercept) 7.91e+00  5.66e-01   14.0  2.6e-13 ***
#ans> Population2010 1.08e-06  1.92e-08   56.3 < 2e-16 ***
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans>
#ans> Residual standard error: 2.29 on 25 degrees of freedom
#ans>   (1 observation deleted due to missingness)
#ans> Multiple R-squared:  0.992,  Adjusted R-squared:  0.992 
#ans> F-statistic: 3.17e+03 on 1 and 25 DF,  p-value: <2e-16

```

The following table contains a handy cheat sheet of equivalences between R code and some of the statistical concepts associated to linear regression.

R	Statistical concept
x	Predictor $X_1, \dots, X_n$
y	Response $Y_1, \dots, Y_n$
data <- data.frame(x = x, y = y)	Sample $(X_1, Y_1), \dots, (X_n, Y_n)$
model <- lm(y ~ x, data = data)	Fitted linear model
model\$coefficients	Fitted coefficients $\hat{\beta}_0, \hat{\beta}_1$
model\$residuals	Fitted residuals $\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_n$
model\$fitted.values	Fitted values $\hat{Y}_1, \dots, \hat{Y}_n$
model\$df.residual	Degrees of freedom $n - 2$
summaryModel <- summary(model)	Summary of the fitted linear model
summaryModel\$sigma	Fitted residual standard deviation $\hat{\sigma}$
summaryModel\$r.squared	Coefficient of determination $R^2$

R	Statistical concept
<code>summaryModel\$fstatistic</code>	$F$ -test
<code>anova(model)</code>	ANOVA table

 Do the following:

- Download The ‘EU’ dataset from [here](#) as an `.RData` file and load it using the function `load`.
- Compute the regression of `CamCom2011` into `Population2010`. Save that model as the variable `myModel`.
- Access the objects `residuals` and `coefficients` of `myModel`.
- Compute the summary of `myModel` and store it as the variable `summaryMyModel`.
- Access the object `sigma` of `myModel`.

### 1.9.2 Predicting House Value: Boston dataset

We are going to use a dataset called Boston which is part of the `MASS` package. It records the median value of houses for 506 neighborhoods around Boston. Our task is to predict the median house value (`medv`) using only one predictor (`lstat`: percent of households with low socioeconomic status).

```
# First, install the MASS package using the command: install.packages("MASS")

# load MASS package
library(MASS)

# Check the dimensions of the Boston dataset
dim(Boston)
#ans> [1] 506 14
```

#### STEP 1: Split the dataset

```
# Split the data by using the first 400 observations as the training
# data and the remaining as the testing data
train = 1:400
test = -train

# Specify that we are going to use only two variables (lstat and medv)
variables = which(names(Boston) == c("lstat", "medv"))
training_data = Boston[train, variables]
testing_data = Boston[test, variables]

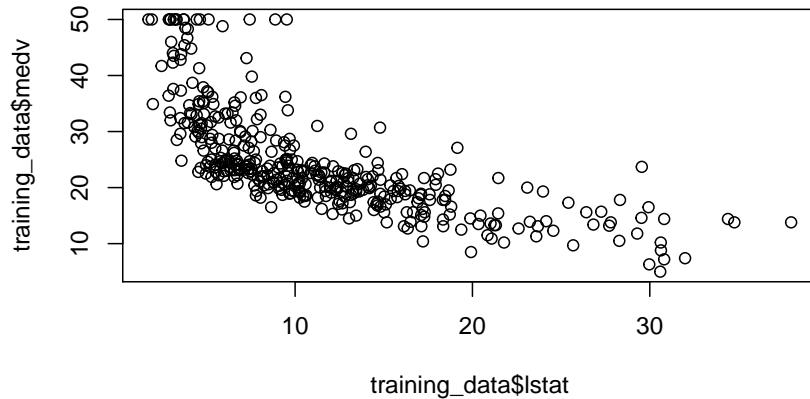
# Check the dimensions of the new dataset
dim(training_data)
#ans> [1] 400 2
```

#### STEP 2: Check for Linearity

In order to perform linear regression in R, we will use the function `lm()` to fit a simple linear regression with `medv` as the response (dependent variable) and `lstat` as the predictor or independent variable, and then save it in `model`.

But before we run our model, let's visually check if the relationship between x and y is linear.

```
# Scatterplot of lstat vs. medv
plot(training_data$lstat, training_data$medv)
```

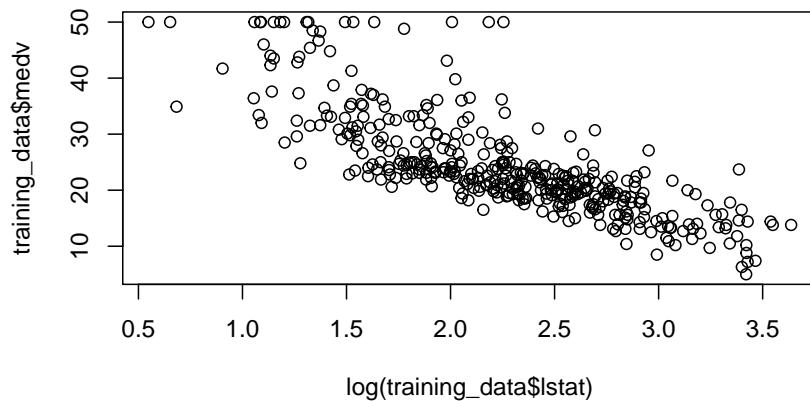


On the figure above, modify the following:

- Figure title.
- Axis titles.
- Shape of observations and their colors.
- Sizes of the chosen shape.

According to the plot, we see that the relationship is not linear. Let's try a transformation of our explanatory variable `lstat`.

```
# Scatterplot of log(lstat) vs. medv
plot(log(training_data$lstat), training_data$medv)
```



Look at the plot, it is more linear, so we can proceed and perform `lm()`:

**STEP 3: Run the linear regression model**

```
model = lm(medv ~ log(lstat), data = training_data)
model
#ans>
#ans> Call:
#ans> lm(formula = medv ~ log(lstat), data = training_data)
#ans>
#ans> Coefficients:
#ans> (Intercept)  log(lstat)
#ans>      51.8       -12.2
```

Notice that basic information when we print `model`. This only give us the slope ( $-12.2$ ) and the intercept ( $51.8$ ) of the linear model. Note that here we are looking at `log(lstat)` and not `lstat` anymore. So for every one unit increase in `lstat`, the median value of the house will decrease by  $e^{12.2}$ . For more detailed information, we can use the `summary()` function:

```
summary(model)
#ans>
#ans> Call:
#ans> lm(formula = medv ~ log(lstat), data = training_data)
#ans>
#ans> Residuals:
#ans>     Min      1Q  Median      3Q     Max
#ans> -11.385 -3.908 -0.779  2.245 25.728
#ans>
#ans> Coefficients:
#ans>             Estimate Std. Error t value Pr(>|t|)
#ans> (Intercept) 51.783     1.097   47.2 <2e-16 ***
#ans> log(lstat) -12.203     0.472  -25.9 <2e-16 ***
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans>
#ans> Residual standard error: 5.6 on 398 degrees of freedom
#ans> Multiple R-squared:  0.627, Adjusted R-squared:  0.626
#ans> F-statistic: 669 on 1 and 398 DF, p-value: <2e-16
```

Now, we have access to p-values and standard errors for the coefficients, as well as the  $R^2$ .

- The output states that the slope is statistically significant and different from 0 and with a t-value=  $-25.9$  (p-value  $< 0.05$ ), which means that there is a significant relationship between the percentage of households with low socioeconomic income and the median house value.
- This relationship is negative. That is as the percentage of household with low socioeconomic income increases, the median house value decreases.
- Looking at  $R^2$ , we can deduce that 62.7% of the model variation is being explained by the predictor `log(lstat)`. This is probably low, but indeed it would increase if we had more independent (explanatory) variables. We can use the `names()` function to see what other pieces of information are stored in our linear model (`model`).

```
names(model)
#ans> [1] "coefficients"    "residuals"      "effects"        "rank"
#ans> [5] "fitted.values"   "assign"         "qr"            "df.residual"
#ans> [9] "xlevels"         "call"          "terms"         "model"

model$coefficients
#ans> (Intercept)  log(lstat)
#ans>      51.8       -12.2
```

To obtain the confidence interval for the linear model (`model`), we can use the `confint()` function:

```
confint(model, level = 0.95)
#ans>           2.5 % 97.5 %
#ans> (Intercept) 49.6   53.9
#ans> log(lstat) -13.1   -11.3
```

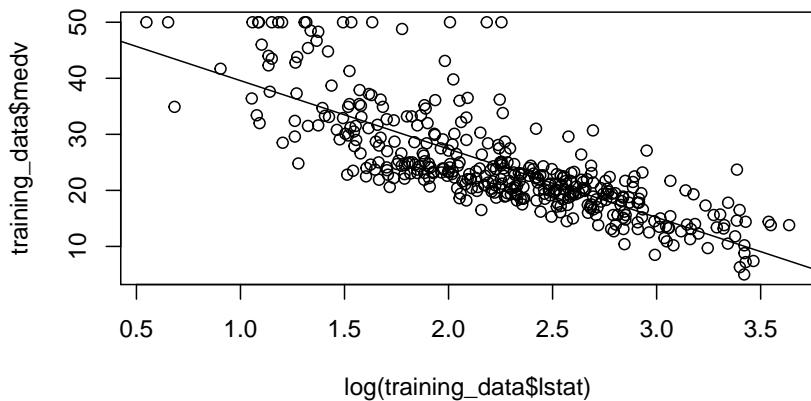
So, a 95% confidence interval for the slope of `log(lstat)` is  $(-13.13, -11.28)$ . Notice that this confidence interval gives us the same result as the hypothesis test performed earlier, by stating that we are 95% confident that the slope of `lstat` is not zero (in fact it is less than zero, which means that the relationship is negative.)

#### STEP 4: Plot the regression model

Now, let's plot our regression line on top of our data.

```
# Scatterplot of lstat vs. medv
plot(log(training_data$lstat), training_data$medv)

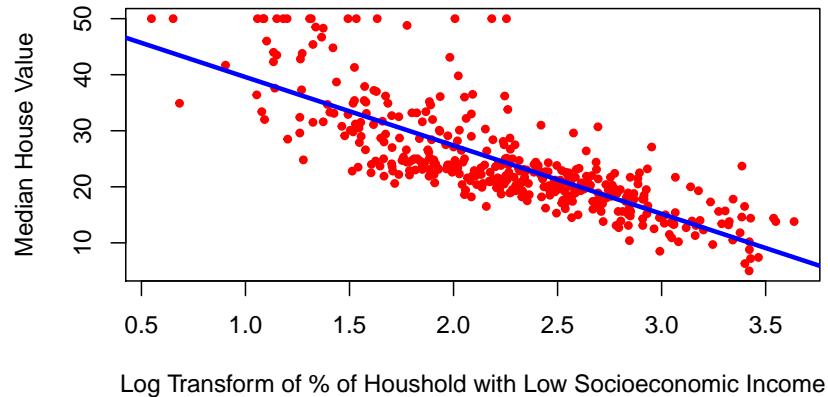
# Add the regression line to the existing scatterplot
abline(model)
```



Let's play with the look of the plot, and makes it prettier!

```
# Scatterplot of lstat vs. medv
plot(log(training_data$lstat), training_data$medv,
xlab = "Log Transform of % of Household with Low Socioeconomic Income",
ylab = "Median House Value",
col = "red",
pch = 20)

# Make the line color blue, and the line's width =3 (play with the width!)
abline(model, col = "blue", lwd = 3)
```



### STEP 5: Assess the model

Final thing we will do is to predict using our fitted model. We can use the `predict()` function for this purpose:

```
# Predict what is the median value of the house with lstat= 5%
predict(model, data.frame(lstat = c(5)))
#ans> 1
#ans> 32.1

# Predict what is the median values of houses with lstat= 5%, 10%, and 15%
predict(model, data.frame(lstat = c(5,10,15), interval = "prediction"))
#ans> 1 2 3
#ans> 32.1 23.7 18.7
```

Now let's assess our model, by computing the mean squared error (MSE). To assess the model we created, then we will be using the test data!

```
# Save the testing median values for houses (testing y) in y
y = testing_data$medv

# Compute the predicted value for this y (y hat)
y_hat = predict(model, data.frame(lstat = testing_data$lstat))

# Now we have both y and y_hat for our testing data.
# let's find the mean square error
error = y-y_hat
error_squared = error^2
MSE = mean(error_squared)
MSE
#ans> [1] 17.7
```

# Chapter 2

## Multiple Linear Regression

Simple linear regression is a useful approach for predicting a response on the basis of a single predictor variable. However, in practice we often have more than one predictor. In the previous chapter, we took for example the prediction of housing prices considering we had the size of each house. We had a single feature  $X$ , the size of the house. But now imagine if we had not only the size of the house as a feature but we also knew the number of bedrooms, the number of floors and the age of the house in years. It seems like this would give us a lot more information with which to predict the price.

### 2.1 The Model

In general, suppose that we have  $p$  distinct predictors. Then the multiple linear regression model takes the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

where  $X_j$  represents the  $j$ th predictor and  $\beta_j$  quantifies the association between that variable and the response. We interpret  $\beta_j$  as the average effect on  $Y$  of a one unit increase in  $X_j$ , *holding all other predictors fixed*.

In matrix terms, supposing we have  $n$  observations and  $p$  variables, we need to define the following matrices:

$$\mathbf{Y}_{n \times 1} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \quad \mathbf{X}_{n \times (p+1)} = \begin{pmatrix} 1 & X_{11} & X_{12} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{np} \end{pmatrix} \quad (2.1)$$

$$\beta_{(p+1) \times 1} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \quad \epsilon_{n \times 1} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \quad (2.2)$$

In matrix terms, the general linear regression model is

$$\mathbf{Y}_{n \times 1} = \mathbf{X}_{n \times (p+1)} \beta_{(p+1) \times 1} + \epsilon_{n \times 1}$$

where,

- $\mathbf{Y}$  is a vector of responses.
- $\beta$  is a vector of parameters.
- $\mathbf{X}$  is a matrix of constants.
- $\epsilon$  is a vector of independent *normal* (Gaussian) random variables.

## 2.2 Estimating the Regression Coefficients

As was the case in the simple linear regression setting, the regression coefficients  $\beta_0, \beta_1, \dots, \beta_p$  are unknown, and must be estimated. Given estimates  $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ , we can make predictions using the formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_p x_p$$

We choose  $\beta_0, \beta_1, \dots, \beta_p$  to minimize the residual sum of squares

$$\begin{aligned} RSS &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 \hat{x}_{i1} - \hat{\beta}_2 \hat{x}_{i2} - \dots - \hat{\beta}_p \hat{x}_{ip})^2 \end{aligned}$$

The values  $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$  that minimize the RSS are the multiple least squares regression coefficient estimates, they are calculated using this formula (in matrix terms):

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

To obtain  $\hat{\beta}$ , we can write the residual sum of squares as

$$RSS = (\mathbf{Y} - \mathbf{X}\beta)^T (\mathbf{Y} - \mathbf{X}\beta)$$

This is a quadratic function in the  $p+1$  parameters. Differentiating with respect to  $\beta$  we obtain

$$\begin{aligned} \frac{\partial RSS}{\partial \beta} &= -2\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\beta) \\ \frac{\partial^2 RSS}{\partial \beta \partial \beta^T} &= 2\mathbf{X}^T \mathbf{X}. \end{aligned}$$

Assuming (for the moment) that  $\mathbf{X}$  has full column rank, and hence  $\mathbf{X}^T \mathbf{X}$  is positive definite<sup>1</sup>, we set the first derivative to zero

$$\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\beta) = 0$$

to obtain the unique solution

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Note 1:

---

<sup>1</sup>Important to be sure that  $\hat{\beta}$  is minimising RSS.



It is a remarkable property of matrix algebra that the results for the general linear regression model in matrix notation appear exactly as those for the simple linear regression model. Only the degrees of freedom and other constants related to the number of  $X$  variables and the dimensions of some matrices are different. Which means there are some similarities between  $\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$  and  $\hat{\beta}_1 = (s_x^2)^{-1} s_{xy}$  from the simple linear model: both are related to the covariance between  $\mathbf{X}$  and  $\mathbf{Y}$  weighted by the variance of  $\mathbf{X}$ .

Note 2:



If  $\mathbf{X}^T \mathbf{X}$  is noninvertible, the common causes might be having:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g.  $p \geq n$ ). In this case, we delete some features or we use “regularization” (to be, maybe, explained in a later lesson).

## 2.3 Some important questions

When we perform multiple linear regression, we usually are interested in answering a few important questions.

1. Is at least one of the predictors  $X_1, X_2, \dots, X_p$  useful in predicting the response?
2. Do all the predictors help to explain  $Y$ , or is only a subset of the predictors useful?
3. How well does the model fit the data?
4. Given a set of predictor values, what response value should we predict, and how accurate is our prediction?

### *Relationship Between the Response and Predictors?*

#### **F-Statistic**

Recall that in the simple linear regression setting, in order to determine whether there is a relationship between the response and the predictor we can simply check whether  $\beta_1 = 0$ . In the multiple regression setting with  $p$  predictors, we need to ask whether all of the regression coefficients are zero, i.e. whether  $\beta_1 = \beta_2 = \dots = \beta_p = 0$ . As in the simple linear regression setting, we use a hypothesis test to answer this question. We test the null hypothesis,

$$H_0 : \beta_1 = \beta_2 = \dots = \beta_p = 0$$

versus the alternative hypothesis

$$H_1 : \text{at least one } \beta_j \text{ is non-zero}$$

This hypothesis test is performed by computing the *F*-statistic (*Fisher*):

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)} \sim F_{p, n-p-1}$$

where, as with simple linear regression,  $TSS = \sum(y_i - \bar{y})^2$  and  $RSS = \sum(y_i - \hat{y}_i)^2$ .

Note that  $F_{p,n-p-1}$  represents the *Fisher-Snedecor's F distribution* with  $p$  and  $n - p - 1$  degrees of freedom. If  $H_0$  is true, then  $F$  is expected to be *small* since  $\text{ESS}^2$  will be close to zero (little variation is explained by the regression model since  $\hat{\beta} \approx \mathbf{0}$ ).

So the question we ask here: *Is the whole regression explaining anything at all?* The answer comes from the  $F$ -test in the ANOVA (ANalysis Of VAriance) table. This is what we get in an ANOVA table:

Source	df	SS	MS	F	p-value
Factor (Explained)	$p$	$\text{ESS} = \text{SSR}$	$\text{MSR} = \text{ESS}/(p)$	$F = \text{MSR}/\text{MSE}$	p-value
Error (Unexplained)	$n - p - 1$	$\text{RSS} = \text{SSE}$	$\text{MSE} = \text{RSS}/(n - p - 1)$		
Total	$n - 1$	$\text{TSS} = \text{SST}$			

The ANOVA table has many pieces of information. What we care about is the  $F$  Ratio and the corresponding p-value. We compare the  $F$  Ratio with  $F_{(p,n-p-1)}$  and a corresponding  $\alpha$  value (error).



The “ANOVA table” is a broad concept in statistics, with different variants. Here we are only covering the basic ANOVA table from the relation  $\text{SST} = \text{SSR} + \text{SSE}$ . However, further sophistications are possible when  $\text{SSR}$  is decomposed into the variations contributed by *each* predictor. In particular, for multiple linear regression R’s `anova` implements a *sequential (type I) ANOVA table*, which is **not** the previous table!

The `anova` function in R takes a model as an input and returns the following *sequential* ANOVA table<sup>3</sup>:

	Degrees of freedom	Sum Squares	Mean Squares	F-value	p-value
Predictor 1	1	$\text{ESS}_1$	$\frac{\text{ESS}_1}{1}$	$\frac{\text{ESS}_1/1}{\text{RSS}/(n-p-1)}$	$p_1$
Predictor 2	1	$\text{ESS}_2$	$\frac{\text{ESS}_2}{1}$	$\frac{\text{ESS}_2/1}{\text{RSS}/(n-p-1)}$	$p_2$
:	:	:	:	:	:
Predictor $p$	1	$\text{ESS}_p$	$\frac{\text{ESS}_p}{1}$	$\frac{\text{ESS}_p/1}{\text{RSS}/(n-p-1)}$	$p_p$
Residuals	$n - p - 1$	$\text{RSS}$	$\frac{\text{RSS}}{n-p-1}$		

Here the  $\text{ESS}_j$  represents the explained sum of squares (same as regression sum of squares) associated to the inclusion of  $X_j$  in the model with predictors  $X_1, \dots, X_{j-1}$ , this is:

$$\text{ESS}_j = \text{ESS}(X_1, \dots, X_j) - \text{ESS}(X_1, \dots, X_{j-1}).$$

The  $p$ -values  $p_1, \dots, p_p$  correspond to the testing of the hypotheses

$$H_0 : \beta_j = 0 \quad \text{vs.} \quad H_1 : \beta_j \neq 0,$$

carried out *inside the linear model*  $Y = \beta_0 + \beta_1 X_1 + \dots + \beta_j X_j + \varepsilon$ . This is like the  $t$ -test for  $\beta_j$  for the model with predictors  $X_1, \dots, X_j$ . Recall that there is no  $F$ -test in this version of the ANOVA table.

### p-values

The p-values provide information about whether each individual predictor is related to the response, after adjusting for the other predictors. Let’s look at the following table we obtain in general using a statistical software for example

<sup>2</sup>Recall that ESS is the explained sum of squares,  $\text{ESS} = \text{TSS} - \text{RSS}$ .

<sup>3</sup>More complex – included here just for clarification of the `anova`’s output.

	Coefficient	Std. error	t-statistic	p-value
Constant	2.939	0.3119	9.42	<0.0001
$X_1$	0.046	0.0014	32.81	<0.0001
$X_2$	0.189	0.0086	21.89	<0.0001
$X_3$	-0.001	0.0059	-0.18	0.8599

In this table we have the following model

$$Y = 2.939 + 0.046X_1 + 0.189X_2 - 0.001X_3$$

Note that for each individual predictor a *t*-statistic and a p-value were reported. These p-values indicate that  $X_1$  and  $X_2$  are related to  $Y$ , but that there is no evidence that  $X_3$  is associated with  $Y$ , in the presence of these two.

### Deciding on Important Variables

The most direct approach is called *all subsets* or *best subsets* regression: we compute the least squares fit for all possible subsets and then choose between them based on some criterion that balances training error with model size.

However we often can't examine all possible models, since they are  $2^p$  of them; for example when  $p = 40$  there are over a billion models! Instead we need an automated approach that searches through a subset of them. Here are two commonly use approaches:

#### Forward selection:

- Begin with the *null model* — a model that contains an intercept (constant) but no predictors.
- Fit  $p$  simple linear regressions and add to the null model the variable that results in the lowest RSS.
- Add to that model the variable that results in the lowest RSS amongst all two-variable models.
- Continue until some stopping rule is satisfied, for example when all remaining variables have a p-value above some threshold.

#### Backward selection:

- Start with all variables in the model.
- Remove the variable with the largest p-value — that is, the variable that is the least statistically significant.
- The new ( $p-1$ )-variable model is fit, and the variable with the largest p-value is removed.
- Continue until a stopping rule is reached. For instance, we may stop when all remaining variables have a significant p-value defined by some significance threshold.



There are more systematic criteria for choosing an “optimal” member in the path of models produced by forward or backward stepwise selection. These include *Mallow's  $C_p$* , *Akaike information criterion (AIC)*, *Bayesian information criterion (BIC)*, *adjusted  $R^2$*  and *Cross-validation (CV)*.

### Model Fit

Two of the most common numerical measures of model fit are the RSE and  $R^2$ , the fraction of variance explained. These quantities are computed and interpreted in the same fashion as for simple linear regression. Recall that in simple regression,  $R^2$  is the square of the correlation of the response and the variable. In multiple linear regression, it turns out that it equals  $\text{Cor}(Y, \hat{Y})^2$ , the square of the correlation between the response and the fitted linear model; in fact one property of the fitted linear model is that it maximizes this correlation among all possible linear models. An  $R^2$  value close to 1 indicates that the model explains a large portion of the variance in the response variable.

In general RSE is defined as

$$\text{RSE} = \sqrt{\frac{1}{n-p-1} \text{RSS}}$$

### 2.3.1 Other Considerations in Regression Model

#### Qualitative Predictors

- If we have a categorial (qualitative) variable (feature), how do we fit into a regression equation?
- For example, if  $X_1$  is the gender (male or female).
- We can code, for example, male = 0 and female = 1.
- Suppose  $X_2$  is a quantitative variable, the regression equation becomes:

$$Y_i \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 = \begin{cases} \beta_0 + \beta_2 X_2 & \text{if male} \\ \beta_0 + \beta_1 X_1 + \beta_2 X_2 & \text{if female} \end{cases}$$

- Another possible coding scheme is to let male = -1 and female = 1, the regression equation is then:

$$Y_i \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 = \begin{cases} \beta_0 - \beta_1 X_1 + \beta_2 X_2 & \text{if male} \\ \beta_0 + \beta_1 X_1 + \beta_2 X_2 & \text{if female} \end{cases}$$

#### Interaction Terms

- When the effect on  $Y$  of increasing  $X_1$  depends on another  $X_2$ .
- We may in this case try the model

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$$

- $X_1 X_2$  is the Interaction term.

## 2.4 How to select the best performing model

After trying different linear models, you need to make a choice which model you want to use. More specifically, the questions that one can ask: “How to determine which model suits best to my data? Do I just look at the R square, SSE, etc.?” and “As the interpretation of that model (quadratic, root, etc.) will be very different, won’t it be an issue?”

The second question can be answered easily. First, find a model that best suits to your data and then interpret its results. It is good if you have ideas how your data might be explained. However, interpret the best model, only. Now we will address the first question. Note that there are multiple ways to select a best model. In addition, this approach only applies to univariate models (simple models) with just one input variable.

Use the following interactive application and play around with different datasets and models. Notice how parameters change and become more confident with assessing simple linear models.

### Use the Adjusted $R^2_{adj}$ for univariate models

If you only use one input variable, the adjusted  $R^2_{adj}$  value gives you a good indication of how well your model performs. It illustrates how much variation is explained by your model.

In contrast to the simple  $R^2$ <sup>4</sup>, the adjusted adjusted  $R_{adj}^2$ <sup>5</sup> takes the number of input factors into account. It penalizes too many input factors and favors parsimonious models.

The adjusted  $R_{adj}^2$  is sensitive to the amount of noise in the data. As such, only compare this indicator of models for the same dataset than comparing it across different datasets.

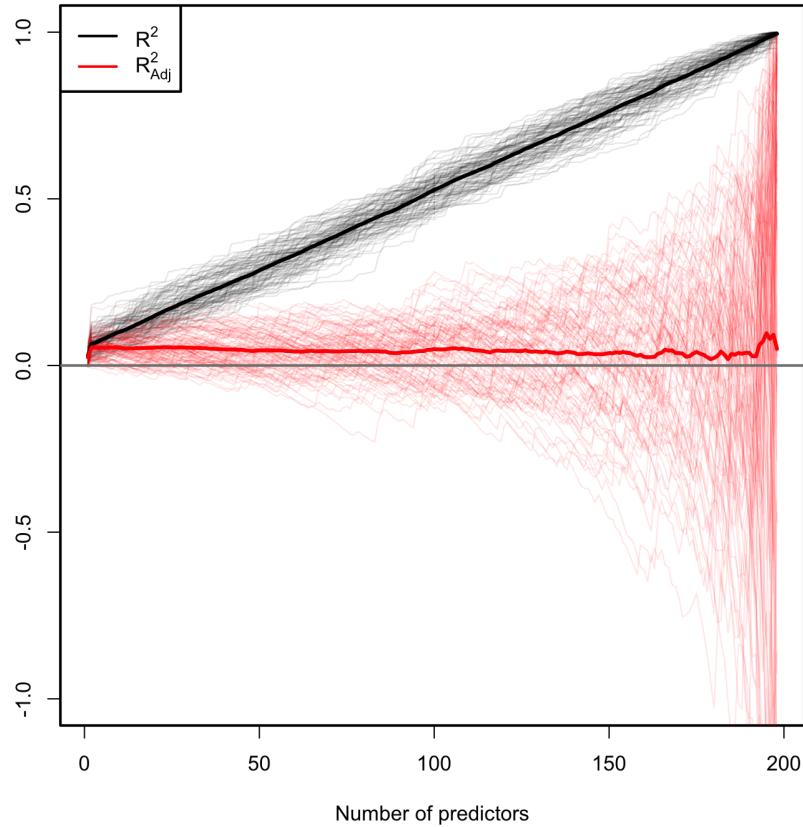


Figure 2.1: Comparison of  $R^2$  and  $R_{adj}^2$  for  $n = 200$  and  $p$  ranging from 1 to 198.  $M = 100$  datasets were simulated with **only the first two** predictors being significant. The thicker curves are the mean of each color's curves.

Figure 2.1 contains the results of an experiment where 100 datasets were simulated with **only the first two** predictors being significant. As you can see  $R^2$  increases linearly with the number of predictors considered, although only the first two ones were important! On the contrary,  $R_{adj}^2$  only increases in the first two variables and then is flat on average, but it has a huge variability when  $p$  approaches  $n - 2$ . The experiment evidences that  $R_{adj}^2$  is **more adequate than the  $R^2$  for evaluating the fit of a multiple linear regression**.

### Have a look at the residuals or error terms

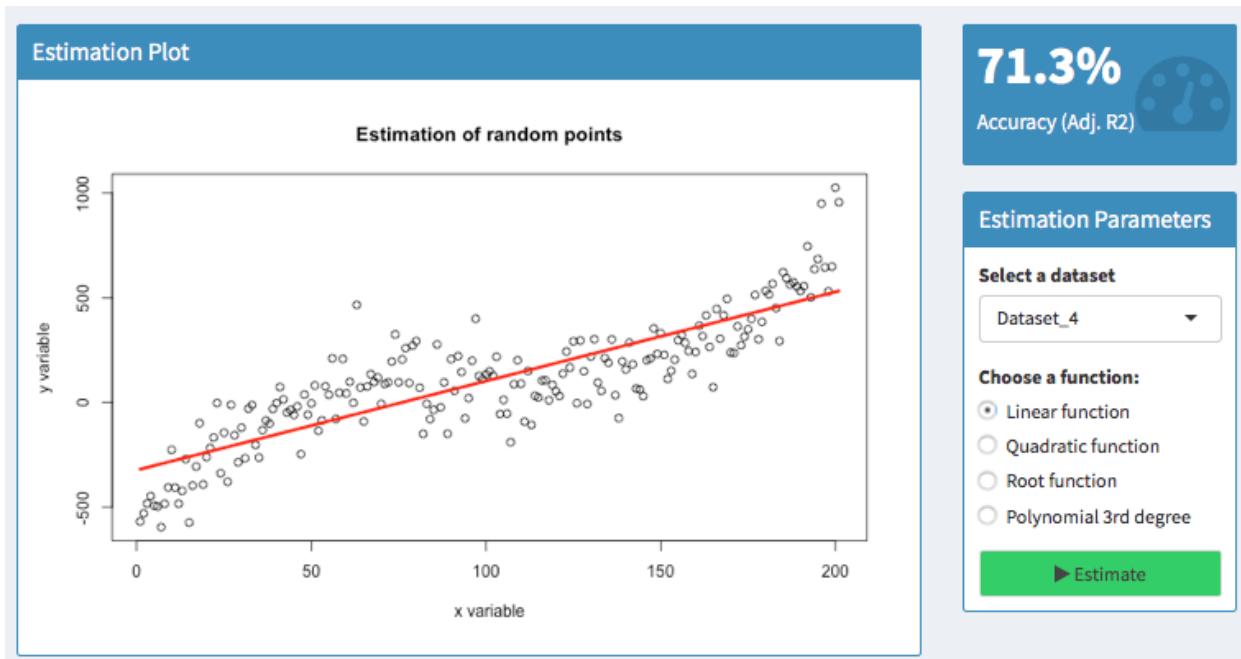
What is often ignored are error terms or so-called residuals. They often tell you more than what you might think. The residuals are the difference between your predicted values and the actual values. Their benefit is that they can show both the magnitude as well as the direction of the errors.

Let's have a look at an example:

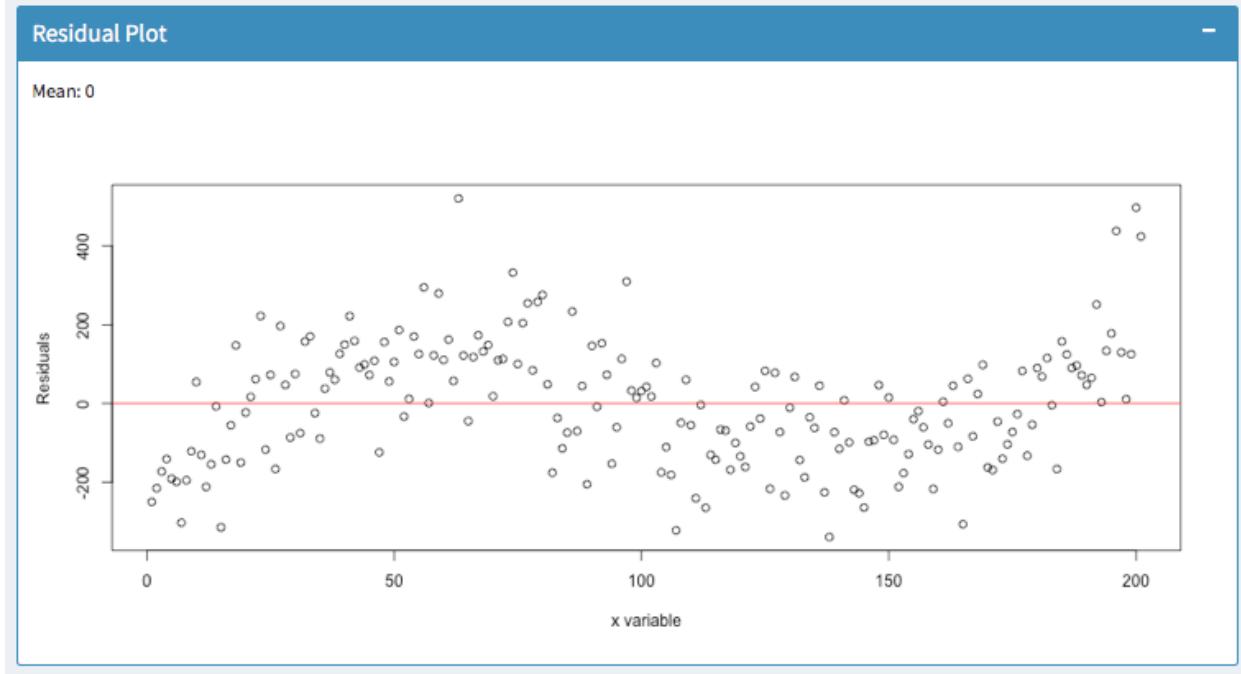
---

<sup>4</sup>Recall that  $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$

<sup>5</sup>It is defined as  $R_{adj}^2 = 1 - \frac{\text{RSS}/(n-p-1)}{\text{TSS}/(n-1)} = 1 - \frac{\text{RSS}}{\text{TSS}} \times \frac{n-1}{n-p-1}$



### Analysis of Residuals



Here, we try to predict a polynomial dataset with a linear function. Analyzing the residuals shows that there are areas where the model has an upward or downward bias.

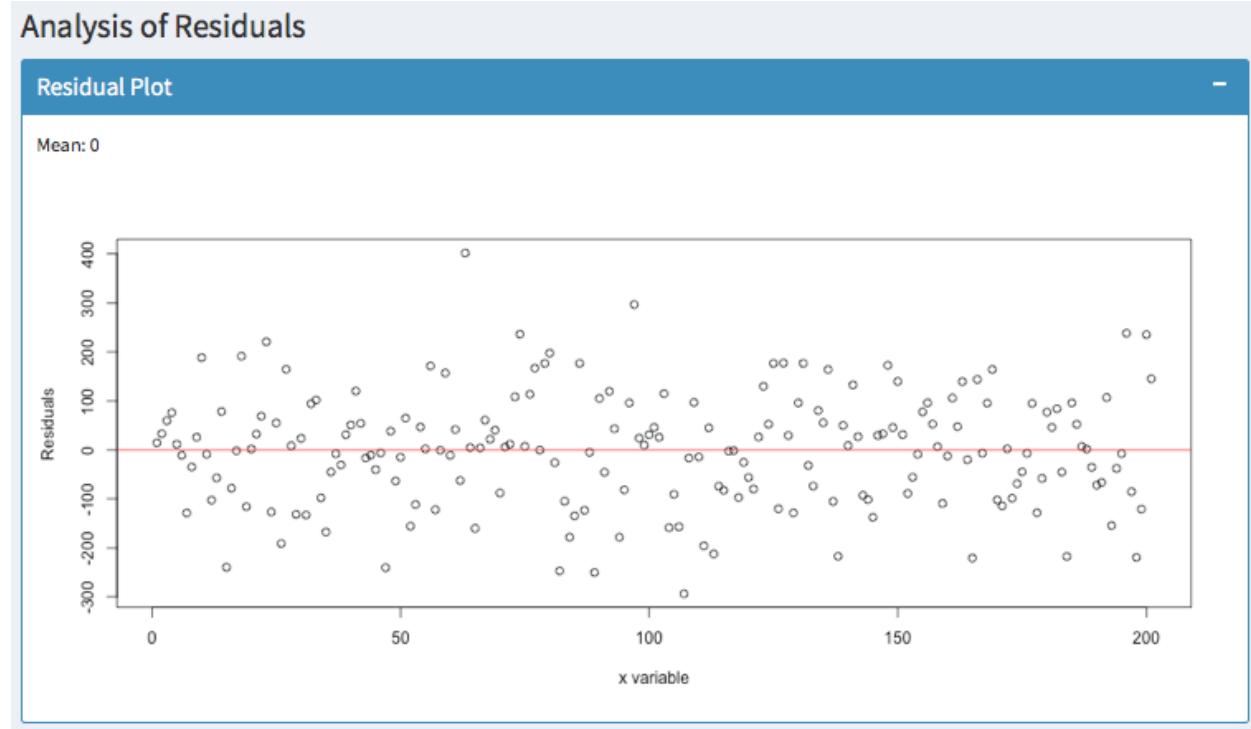
For  $50 < x < 100$ , the residuals are above zero. So in this area, the actual values have been higher than the predicted values—our model has a downward bias.

For  $100 < x < 150$ , however, the residuals are below zero. Thus, the actual values have been lower than the predicted values—the model has an upward bias.

It is always good to know, whether your model suggests too high or too low values. But you usually do not

want to have patterns like this.

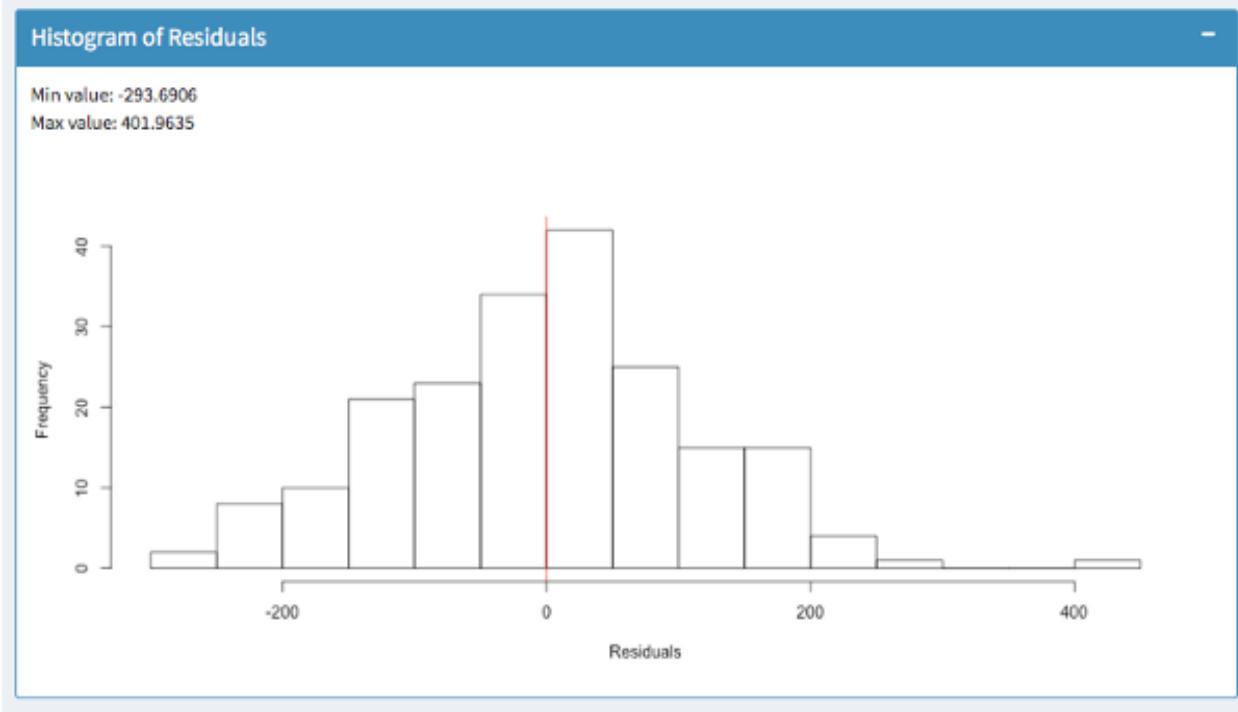
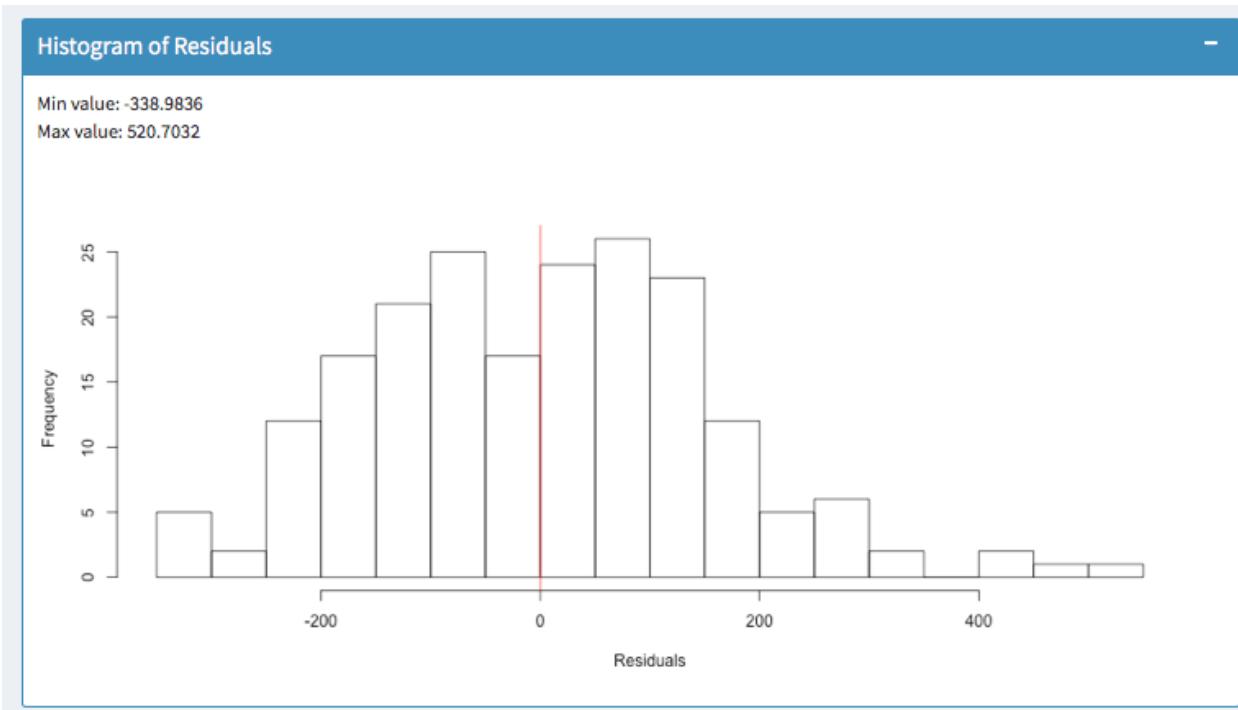
The residuals should be zero on average (as indicated by the mean) and they should be equally distributed. Predicting the same dataset with a polynomial function of 3 degrees suggests a much better fit:



In addition, you can observe whether the variance of your errors increases. In statistics, this is called Heteroscedasticity. You can fix this easily with robust standard errors. Otherwise, your hypothesis tests are likely to be wrong.

### Histogram of residuals

Finally, the histogram summarizes the magnitude of your error terms. It provides information about the bandwidth of errors and indicates how often which errors occurred.



The above screenshots show two models for the same dataset. In the first histogram, errors occur within a range of -338 and 520. In the second histogram, errors occur within -293 and 401. So the outliers are much lower. Furthermore, most errors in the model of the second histogram are closer to zero. So we would favor the second model.

# PW 2

## Multiple Linear Regression

In this practical work, we will continue the analysis of the Boston data set that we started last week (section 1.9.2). Recall that this dataset records the median value of houses for 506 neighborhoods around Boston. Our task is to predict the median house value (`medv`).

1. Load the Boston dataset from `MASS` package.
2. Split the dataset into training set and testing set. (keep all the variables of the Boston data set)
3. Check if there is a linear relationship between the variables `medv` and `age`. (use `cor()` function).
4. Fit a model of housing prices in function of age and plot the observations and the regression line.
5. Train a regression model using both `lstat` and `age` as predictors of median house value. (Remember that we transformed `lstat`, use the same transformation here). What is the obtained model?
6. Print the summary of the obtained regression model.
7. Are the predictors significant ?
8. Is the model as a whole significant? Answer on this question must be detailed.
9. Train a new model using all the variables of the dataset. (We can use `.` as a short cut instead of writing down all the variables names)
10. When using all the variables as predictors, we didn't transform `lstat`. Re train the model using `log(lstat)` instead of `lstat`.
11. Did  $R^2$  improve ?
12. To see if there is correlated variables print the correlation matrix using the `cor()` function (round the correlations with 2 digits).
13. Visualize the correlations using the `corrplot` package. To do so, install the `corrplot` package, load it, then use the function `corrplot.mixed()`. See this link for examples and to understand how to use it.
14. What is the correlation between `tax` and `rad`?
15. Run the model again without `tax`. What happens to the  $R^2$  ? and for the F-statistic?



Of course  $R^2$  should go a little lower because we deleted one of the variables. But check for the model significance (F-statistic) gets higher, which means the p-values get lower and thus the model is more significant without `rad`.

16. Calculate the mean squared error (MSE) for the last model.

### Anova

Next we will apply an analysis of variances (ANOVA) in order to test if there is a significant difference of means between two groups  $i$  and  $j$  (Consider group  $i$  is the suburbs bounding the river and  $j$  the suburbs which not). The hypotheses are

$$H_0 : \mu_i = \mu_j$$

$$H_1 : \mu_i \neq \mu_j$$

Where  $\mu_i$  is the mean of `medv` in group  $i$ .

17. In the Boston data set there is a categorical variable `chas` which corresponds to Charles River (= 1 if a suburb bounds the river; 0 otherwise). Use command `str()` to see how this variable is present in the dataset. How many of the suburbs in this data set bound the Charles river?
18. Create Boxplots of the median value of houses with respect to the variable `chas`. Do we observe some difference between the median value of houses with respect to the neighborhood to Charles River?
19. Calculate  $\mu_i$  and  $\mu_j$  (in one line using the function `aggregate()`).
20. Apply an ANOVA test of `medv` whith respect to `chas` (use the function `aov()`). Print the result and the summary of it. what do you conclude ?

### Qualitative predictors

 Before starting the next question, please read section 2.3.1 and Appendix C about using qualitative predictors in regression.

We are going to use the categorical variable `chas` which corresponds to Charles River (= 1 if a suburb bounds the river; 0 otherwise). Using the `str()` command you can notice that this variable is not codified as a factor, but it has values 0 or 1, so it is already dummyfied.

21. Fit a new model where the predictors are the Charles River and the Crime Rate. Interpret the coefficients of this model and conclude if the presence of the river adds a valuable information for explaining the house price.

22. Is `chas` is significant as well in the presence of more predictors?

### Interaction terms

As you saw in section 2.3.1 we may sometimes try models with intercation terms. Let's say we have two predictors  $X_1$  and  $X_2$ , the way of adding these interactions in R is through `:` and `*`. The operator `:` only adds the term  $X_1X_2$  and `*` adds  $X_1$ ,  $X_2$ , and  $X_1X_2$ .

23. Fit a model whith first order interaction term where predictors are `lstat` and `age`. Print its summary.
24. Fit a model with all the first order interaction terms.

## Reporting

In R there is some packages to make it **easy to create reproducible web-based reports**. To do so, click on `File -> Knit document` or `File -> Compile report...`. The output is a html report containing the results of your R codes. If your file is named `report.R`, your report is named `report.html`.

25. Compile a report based on your script.



- Make sure to have the latest version of `Rstudio`.
- If you have problems with compiling (problem in installing packages, etc..) close your `Rstudio` and reopen it with administrative tools and retry.
- Be ready to submit your report (your `.html` file) at the end of each class.
- Your report must be named: `YouLastName_YourFirstName_WeekNumber.html`



# Classification



# Chapter 3

## Logistic Regression

### 3.1 Introduction

In the previous chapters we discussed the linear regression model, which assumes that the response variable  $Y$  is quantitative. But in many situations, the response variable is instead qualitative (categorical). For example, eye color is qualitative, taking on values blue, brown, or green.

The process for predicting qualitative responses is known as *classification*.

Given a feature vector  $X$  and a qualitative response  $Y$  taking values in the set  $\mathcal{C}$ , the classification task is to build a function  $C(X)$  that takes as input the feature vector  $X$  and predicts its value for  $Y$ ; i.e.  $C(X) \in \mathcal{C}$ . We are often more interested in estimating the probabilities that  $X$  belongs to each category in  $\mathcal{C}$ .

If  $c$  is a category ( $c \in \mathcal{C}$ ), by the probability that  $X$  belongs to  $c$  we mean  $p(X \in c) = \mathbb{P}(Y = c|X)$ .

In the binomial or binary logistic regression, the outcome can have only two possible types of values (e.g. “Yes” or “No”, “Success” or “Failure”). Multinomial logistic refers to cases where the outcome can have three or more possible types of values (e.g., “good” vs. “very good” vs. “best”). Generally outcome is coded as “0” and “1” in binary logistic regression.

### 3.2 Logistic Regression

Consider a data set where the response falls into one of two categories, Yes or No. Rather than modeling the response  $Y$  directly, logistic regression models the *probability* that  $Y$  belongs to a particular category.

#### 3.2.1 The Logistic Model

Let us suppose the response has two categories and we use the generic 0/1 coding for the response. How should we model the relationship between  $p(X) = \mathbb{P}(Y = 1|X)$  and  $X$ ?

The simplest situation is when  $Y$  is *binary*: it can only take two values, codified for convenience as 1 (success) and 0 (failure).

More formally, a binary variable is known as a *Bernoulli variable*, which is the simplest non-trivial random variable. We say that  $Y \sim \text{Ber}(p)$ ,  $0 \leq p \leq 1$ , if

$$Y = \begin{cases} 1, & \text{with probability } p, \\ 0, & \text{with probability } 1 - p, \end{cases}$$

or, equivalently, if  $\mathbb{P}[Y = 1] = p$  and  $\mathbb{P}[Y = 0] = 1 - p$ , which can be written compactly as

$$\mathbb{P}[Y = y] = p^y(1 - p)^{1-y}, \quad y = 0, 1.$$

Recall that a *binomial variable with size n and probability p*,  $\text{Bi}(n, p)$ , was obtained by adding  $n$  independent  $\text{Ber}(p)$  (so  $\text{Ber}(p)$  is the same as  $\text{Bi}(1, p)$ ).



A Bernoulli variable  $Y$  is completely determined by  $p$ . **So its mean and variance:**

- $\mathbb{E}[Y] = p \times 1 + (1 - p) \times 0 = p$
- $\text{Var}[Y] = p(1 - p)$ .

In particular, recall that  $\mathbb{P}[Y = 1] = \mathbb{E}[Y] = p$ .

Assume then that  $Y$  is a binary/Bernoulli variable and that  $X$  are predictors associated to them (no particular assumptions on them). The purpose in *logistic regression* is to estimate

$$p(x) = \mathbb{P}[Y = 1|X = x] = \mathbb{E}[Y|X = x],$$

this is, how the probability of  $Y = 1$  is changing according to particular values, denoted by  $x$ , of the random variables  $X$ .

*Why not linear regression?* A tempting possibility is to consider the model

$$p(x) = \beta_0 + \beta_1 x.$$

However, such a model will run into problems inevitably: negative probabilities and probabilities larger than one ( $p(x) < 0$  for some values of  $X$  and  $p(X) > 1$  for others). To avoid this problem, the solution is to consider a function to encapsulate the value of  $z = \beta_0 + \beta_1 x$ , in  $\mathbb{R}$ , and map it to  $[0, 1]$ . There are several alternatives to do so, based on distribution functions  $F : \mathbb{R} \longrightarrow [0, 1]$  that deliver  $y = F(z) \in [0, 1]$ . Many functions meet this description. In logistic regression, we use the *logistic function*,

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$



- No matter what values  $\beta_0$ ,  $\beta_1$  or  $X$  take,  $p(X)$  will have values between 0 and 1.
- The logistic function will always produce an *S-shaped* curve.
- The logistic *distribution* function is:

$$F(z) = \text{logistic}(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}.$$

After a bit of manipulation of the previous equation, we find that

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X}$$



The quantity  $p(X)/[1 - p(X)]$  is called the *odds*, and can take on any value between 0 and  $\infty$ .

By taking the logarithm of both sides of the equation, we arrive at

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X$$



The left-hand side is called the *log-odds* or *logit*. We see that the logistic regression model has a logit that is linear in X.

### 3.2.2 Estimating the Regression Coefficients

We estimate  $\beta_0$  and  $\beta_1$  using the *Maximum Likelihood Estimation* method (MLE). The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for  $\beta_0$  and  $\beta_1$  such that the predicted probability  $\hat{p}(x_i)$  of the response for each individual, corresponds as closely as possible to the individual's observed response status (recall that the response  $Y$  is categorical). The *likelihood function* is

$$l(\beta_0, \beta_1) = \prod_{i=1}^n p(x_i)^{Y_i} (1 - p(x_i))^{1-Y_i}.$$

This likelihood is **the probability of the data based on the model**. It gives the probability of the observed zeros and ones in the data. The estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are chosen to *maximize* this likelihood function. The interpretation of the likelihood function is the following:

- $\prod_{i=1}^n$  appears because the sample elements are assumed to be independent and we are computing the probability of observing the whole sample  $(x_1, y_1), \dots, (x_n, y_n)$ . This probability is equal to the product of the *probabilities of observing each*  $(x_i, y_i)$ .
- $p(x_i)^{Y_i} (1 - p(x_i))^{1-Y_i}$  is the probability of observing  $(x_i, Y_i)$ .



In the linear regression setting, the least squares approach is a special case of maximum likelihood.

We will not give mathematical details about the maximum likelihood and how to estimate the parameters. We will use R to fit the logistic regression models (using `glm` function).

Use the following application (also available here) to see how the log-likelihood changes with respect to the values for  $(\beta_0, \beta_1)$  in three data patterns. The logistic regression fit and its dependence on  $\beta_0$  (horizontal displacement) and  $\beta_1$  (steepness of the curve). Recall the effect of the sign of  $\beta_1$  in the curve: if positive, the logistic curve has an *s* form; if negative, the form is a reflected *s*.

```
#ans> PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pl
```

Note that the **animation** will not be displayed the first time it is browsed (The reason is because it is hosted at `https` websites with auto-signed SSL certificates). **To see it**, click on the link above. You will get a warning from your browser saying that “Your connection is not private”. Click in “Advanced” and **allow an exception** in your browser. The next time the animation will show up correctly.

### 3.2.3 Prediction

#### Example

	Coefficient	Std. error	Z-statistic	p-value
Constant	-10.6513	0.3612	-29.5	<0.0001
X	0.0055	0.0002	24.9	<0.0001

In this example,  $\hat{\beta}_0 = -10.6513$  and  $\hat{\beta}_1 = 0.0055$ . It produces the blue curve that separates that data in the

following figure,

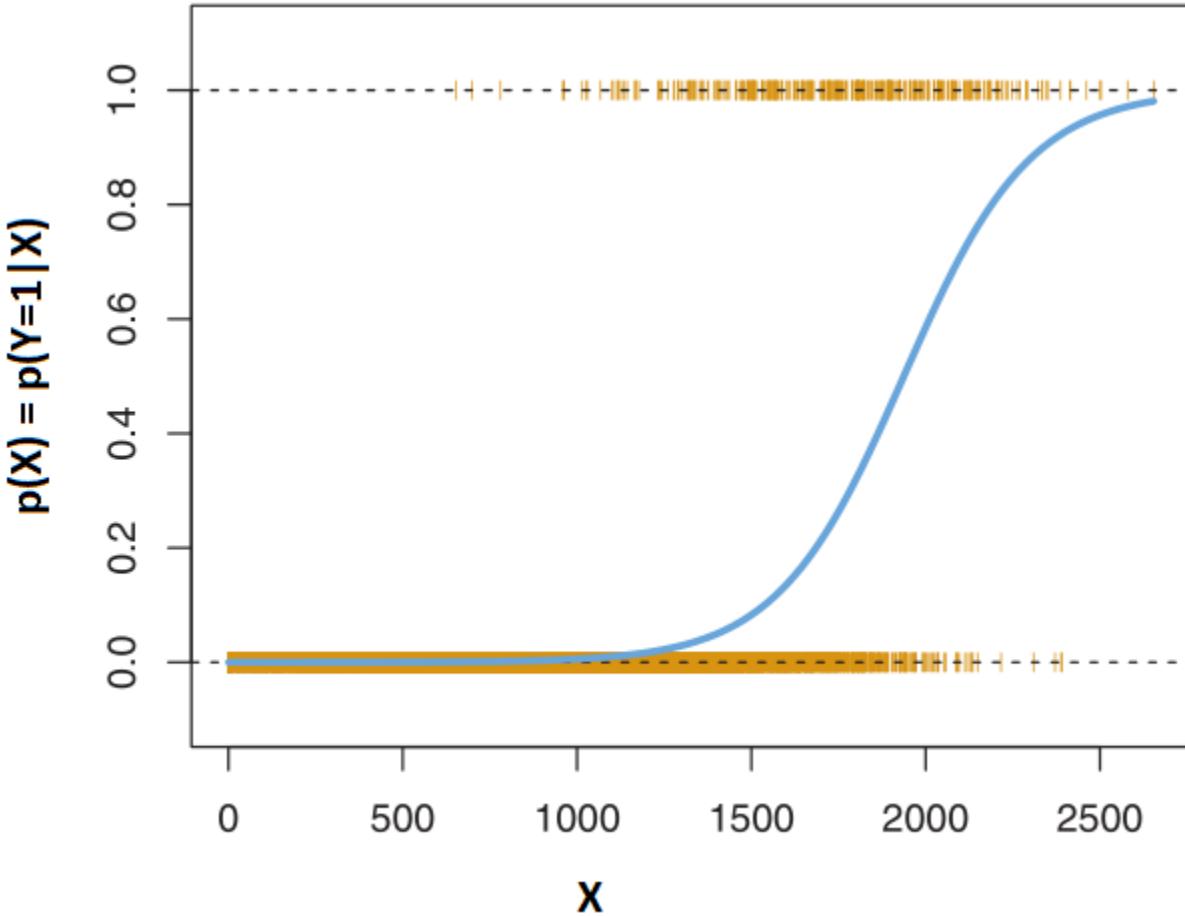


Figure 3.1:

As for prediction, we use the model built with the estimated parameters to predict probabilities. For example, If  $X = 1000$ ,

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}} = \frac{e^{-10.6513 + 0.0055 \times 1000}}{1 + e^{-10.6513 + 0.0055 \times 1000}} = 0.006$$

If  $X = 2000$ ,

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}} = \frac{e^{-10.6513 + 0.0055 \times 2000}}{1 + e^{-10.6513 + 0.0055 \times 2000}} = 0.586$$

### 3.3 Multiple Logistic Regression

We now consider the problem of predicting a binary response using multiple predictors. By analogy with the extension from simple to multiple linear regression in the previous chapters, we can generalize the simple

logistic regression equation as follows:

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

where  $X = (X_1, \dots, X_p)$  are  $p$  predictors. The equation above can be rewritten as

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}$$

Just as in the simple logistic regression we use the maximum likelihood method to estimate  $\beta_0, \beta_1, \dots, \beta_p$ .

## 3.4 Example

### 3.4.1 Case study: *The Challenger disaster*

The *Challenger* disaster occurred on the 28th January of 1986, when the NASA Space Shuttle orbiter *Challenger* broke apart and disintegrated at 73 seconds into its flight, leading to the deaths of its seven crew members. The accident deeply shocked the US society, in part due to the attention the mission had received because of the presence of Christa McAuliffe, who would have been the first astronaut-teacher. Because of this, NASA TV broadcasted live the launch to US public schools, which resulted in millions of school children witnessing the accident. The accident had serious consequences for the NASA credibility and resulted in an interruption of 32 months in the shuttle program. The Presidential *Rogers Commission* (formed by astronaut Neil A. Armstrong and Nobel laureate Richard P. Feynman, among others) was created to investigate the disaster.

Challenger launch and posterior explosion, as broadcasted live by NBC in 28/01/1986.

The Rogers Commission elaborated a report (Presidential Commission on the Space Shuttle Challenger Accident, 1986) with all the findings. The commission determined that the disintegration began with the **failure of an O-ring seal in the solid rocket motor due to the unusual cold temperatures (-0.6 Celsius degrees)** during the launch. This failure produced a breach of burning gas through the solid rocket motor that compromised the whole shuttle structure, resulting in its disintegration due to the extreme aerodynamic forces. The **problematic with O-rings was something known**: the night before the launch, there was a three-hour teleconference between motor engineers and NASA management, discussing the effect of low temperature forecasted for the launch on the O-ring performance. The conclusion, influenced by Figure 3.2a, was:

**“Temperature data [are] not conclusive on predicting primary O-ring blowby.”**

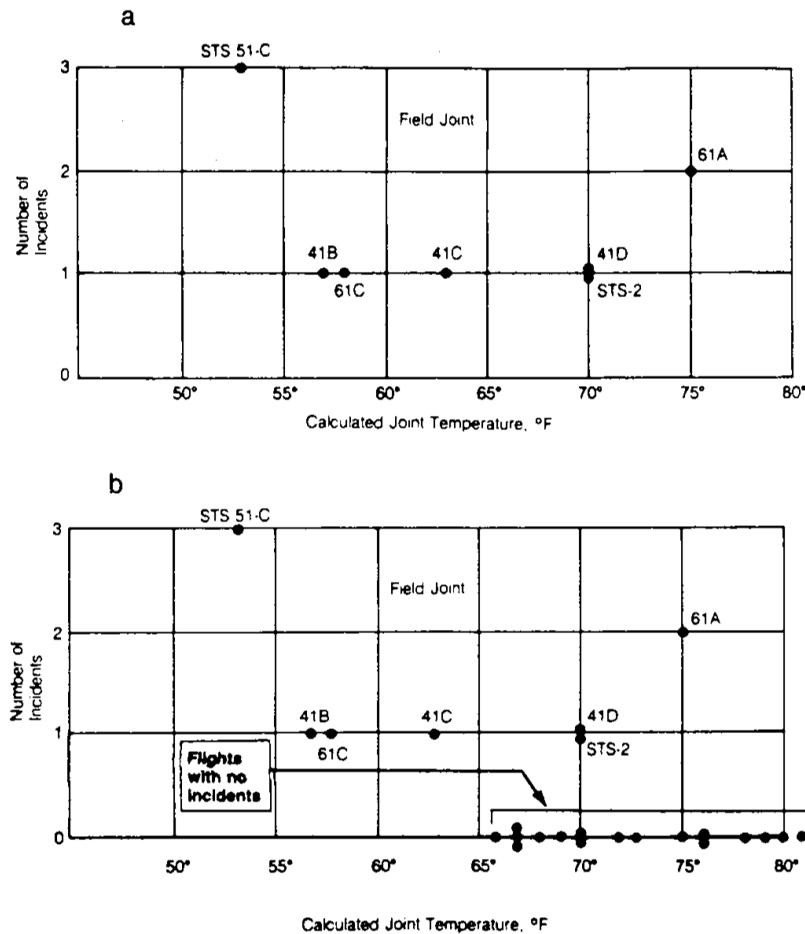


Figure 3.2: Number of incidents in the O-rings (field joints) versus temperatures. Panel *a* includes only flights with incidents. Panel *b* contains all flights (with and without incidents).

The Rogers Commission noted a major flaw in Figure 3.2a: the **flights with zero incidents were excluded** from the plot because *it was felt that these flights did not contribute any information about the temperature effect* (Figure 3.2b). The Rogers Commission concluded:

**“A careful analysis of the flight history of O-ring performance would have revealed the correlation of O-ring damage in low temperature”.**

The purpose of this case study, inspired by Dalal et al. (1989), is to quantify what was the influence of the temperature in the probability of having at least one incident related with the O-rings. Specifically, we want to address the following questions:

- Q1. *Is the temperature associated with O-ring incidents?*
- Q2. *In which way was the temperature affecting the probability of O-ring incidents?*
- Q3. *What was the predicted probability of an incident in an O-ring for the temperature of the launch day?*

To try to answer these questions we have the **challenger** ( ) dataset. The dataset contains (as shown in the table below) information regarding the state of the solid rocket boosters after launch<sup>1</sup> for 23 flights. Each row has, among others, the following variables:

<sup>1</sup> After the shuttle exits the atmosphere, the solid rocket boosters separate and descend to land using a parachute where they are carefully analyzed.

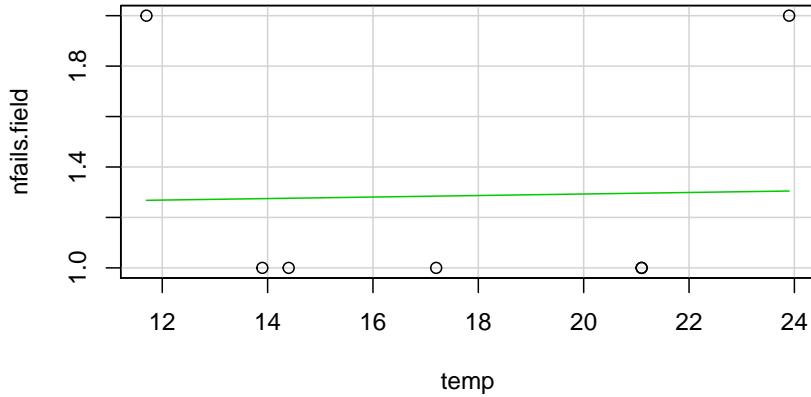
- `fail.field`, `fail.nozzle`: binary variables indicating whether there was an incident with the O-rings in the field joints or in the nozzles of the solid rocket boosters. 1 codifies an incident and 0 its absence. On the analysis, we focus on the O-rings of the field joint as being the most determinants for the accident.
- `temp`: temperature in the day of launch. Measured in Celsius degrees.
- `pres.field`, `pres.nozzle`: leak-check pressure tests of the O-rings. These tests assured that the rings would seal the joint.

Table 3.2: The `challenger` dataset.

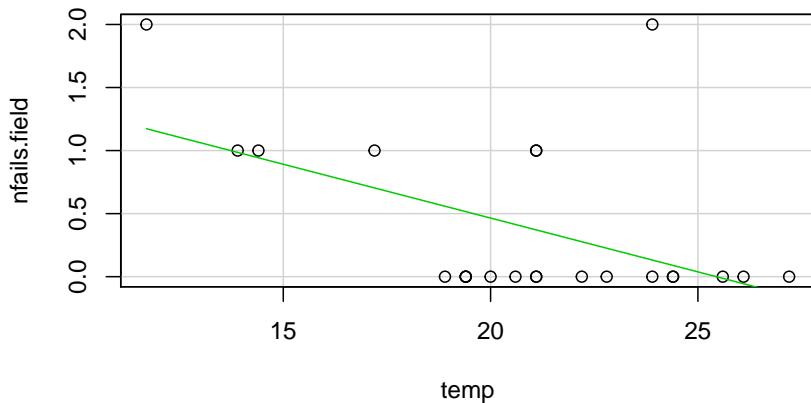
flight	date	fail.field	fail.nozzle	temp
1	12/04/81	0	0	18.9
2	12/11/81	1	0	21.1
3	22/03/82	0	0	20.6
5	11/11/82	0	0	20.0
6	04/04/83	0	1	19.4
7	18/06/83	0	0	22.2
8	30/08/83	0	0	22.8
9	28/11/83	0	0	21.1
41-B	03/02/84	1	1	13.9
41-C	06/04/84	1	1	17.2
41-D	30/08/84	1	1	21.1
41-G	05/10/84	0	0	25.6
51-A	08/11/84	0	0	19.4
51-C	24/01/85	1	1	11.7
51-D	12/04/85	0	1	19.4
51-B	29/04/85	0	1	23.9
51-G	17/06/85	0	1	21.1
51-F	29/07/85	0	0	27.2
51-I	27/08/85	0	0	24.4
51-J	03/10/85	0	0	26.1
61-A	30/10/85	1	0	23.9
61-B	26/11/85	0	1	24.4
61-C	12/01/86	1	1	14.4

Let's begin the analysis by replicating Figures 3.2a and 3.2b and checking that linear regression is not the right tool for answering Q1–Q3. For that, we make two scatterplots of `nfails.field` (number of total incidents in the field joints) in function of `temp`, the first one excluding the launches without incidents (`subset = nfails.field > 0`) and the second one for all the data.

```
require(car)
#ans> Loading required package: car
scatterplot(nfails.field ~ temp, reg.line = lm, smooth = FALSE, spread = FALSE,
           boxplots = FALSE, data = challenger, subset = nfails.field > 0)
```

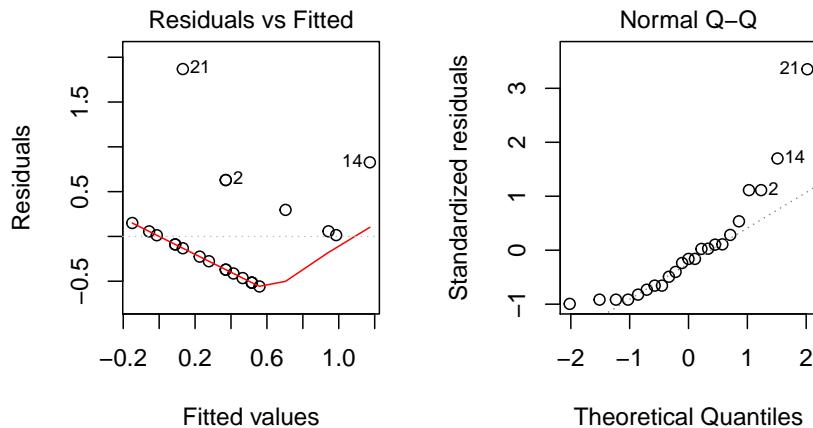


```
scatterplot(nfails.field ~ temp, reg.line = lm, smooth = FALSE, spread = FALSE,  
           boxplots = FALSE, data = challenger)
```



There is a fundamental problem in using linear regression for this data: **the response is not continuous**. As a consequence, there is no linearity and the errors around the mean are not normal (indeed, they are strongly non normal). We can check this with the corresponding diagnostic plots:

```
mod <- lm(nfails.field ~ temp, data = challenger)
par(mfrow = 1:2)
plot(mod, 1)
plot(mod, 2)
```



Although linear regression is not the adequate tool for this data, it is able to detect the obvious difference between the two plots:

1. **The trend for launches with incidents is flat, hence suggesting there is no dependence on the temperature** (Figure 3.2a). This was one of the arguments behind NASA's decision of launching the rocket at a temperature of -0.6 degrees.
2. However, **the trend for all launches indicates a clear negative dependence between temperature and number of incidents!** (Figure 3.2b). Think about it in this way: the minimum temperature for a launch without incidents ever recorded was above 18 degrees, and the Challenger was launched at -0.6 without clearly knowing the effects of such low temperatures.

Instead of trying to predict the number of incidents, we will concentrate on modeling the *probability of expecting at least one incident given the temperature*, a simpler but also revealing approach. In other words, we look to estimate the following curve:

$$p(x) = \mathbb{P}(\text{incident} = 1 | \text{temperature} = x)$$

from `fail.field` and `temp`. This probability can not be properly modeled as a linear function like  $\beta_0 + \beta_1 x$ , since inevitably will fall outside  $[0, 1]$  for some value of  $x$  (some will have negative probabilities or probabilities larger than one). The technique that solves this problem is the **logistic regression**. The logistic model in this case is

$$\mathbb{P}(\text{incident} = 1 | \text{temperature} = x) = \text{logistic}(\beta_0 + \beta_1 x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}},$$

with  $\beta_0$  and  $\beta_1$  unknown.

Let's fit the model to the data by estimating  $\beta_0$  and  $\beta_1$ .

```
nasa <- glm(fail.field ~ temp, family = "binomial", data = challenger)
summary(nasa)

#ans>
#ans> Call:
#ans> glm(formula = fail.field ~ temp, family = "binomial", data = challenger)
#ans>
#ans> Deviance Residuals:
#ans>      Min        1Q     Median        3Q       Max
#ans> -1.057   -0.757   -0.382    0.457    2.220
#ans>
#ans> Coefficients:
#ans>             Estimate Std. Error z value Pr(>|z|)
```

```
#ans> (Intercept)    7.584      3.915     1.94     0.053 .
#ans> temp          -0.417      0.194    -2.15     0.032 *
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans>
#ans> (Dispersion parameter for binomial family taken to be 1)
#ans>
#ans> Null deviance: 28.267 on 22 degrees of freedom
#ans> Residual deviance: 20.335 on 21 degrees of freedom
#ans> AIC: 24.33
#ans>
#ans> Number of Fisher Scoring iterations: 5
exp(coef(nasa)) # Exponentiated coefficients ("odds ratios")
#ans> (Intercept)           temp
#ans> 1965.974            0.659
```



The `glm()` function fits generalized linear models, a class of models that includes logistic regression. The syntax of the `glm()` function is similar to that of `lm()`, except that we must pass in the argument `family=binomial` in order to tell R to run a logistic regression rather than some other type of generalized linear model.

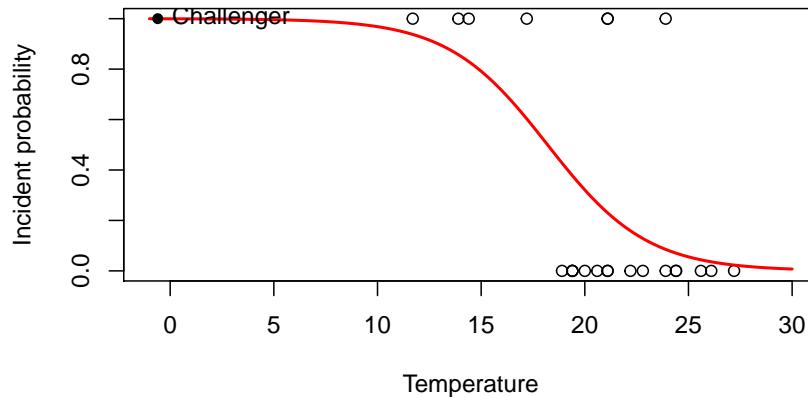
The summary of the logistic model is notably different from the linear regression, as the methodology behind is quite different. Nevertheless, we have tests for the significance of each coefficient. Here we obtain that `temp` is significantly different from zero, at least at a level  $\alpha = 0.05$ . Therefore we can conclude that **the temperature is indeed affecting the probability of an incident with the O-rings** (answers Q1).

The coefficient of `temp`,  $\hat{\beta}_1$ , can be regarded the “correlation between the temperature and the probability of having at least one incident”. This correlation, as evidenced by the sign of  $\hat{\beta}_1$ , is negative. Let’s plot the fitted logistic curve to see that indeed the probability of incident and temperature are negatively correlated:

```
# Plot data
plot(challenger$temp, challenger$fail.field, xlim = c(-1, 30), xlab = "Temperature",
      ylab = "Incident probability")

# Draw the fitted logistic curve
x <- seq(-1, 30, l = 200)
y <- exp(-(nasa$coefficients[1] + nasa$coefficients[2] * x))
y <- 1 / (1 + y)
lines(x, y, col = 2, lwd = 2)

# The Challenger
points(-0.6, 1, pch = 16)
text(-0.6, 1, labels = "Challenger", pos = 4)
```



At the sight of this curve and the summary of the model we can conclude that **the temperature was increasing the probability of an O-ring incident** (Q2). Indeed, the confidence intervals for the coefficients show a significative negative correlation at level  $\alpha = 0.05$ :

```
confint(nasa, level = 0.95)
#ans>              2.5 % 97.5 %
#ans> (Intercept)  1.336 17.783
#ans> temp        -0.924 -0.109
```

Finally, **the probability of having at least one incident with the O-rings in the launch day was 0.9996 according to the fitted logistic model** (Q3). This is easily obtained:

```
predict(nasa, newdata = data.frame(temp = -0.6), type = "response")
#ans> 1
#ans> 1
```

Be aware that `type = "response"` has a different meaning in logistic regression. In linear models it returns a CI for the prediction. But, `type = "response"` means that the *probability* should be returned, instead of the value of the link function, which is returned with `type = "link"` (the default).

Recall that there is a serious problem of **extrapolation** in the prediction, which makes it less precise (or more variable). But this extrapolation, together with the evidences raised by a simple analysis like we did, should have been strong arguments for postponing the launch.



# PW 3

## Social Networks Ads

In this PW we are going to analyse the `Social_Network_Ads` dataset . This dataset contains informations of users of a social network. The social network has several business clients and its business clients put ads on the social network for marketing compaigns purposes. For this dataset, a company has put ads for one of its new products and the social network gathered some informations about wich users responded positively to the ad by buying the product and those who responded negatively by not buying the product.

1. Download the `Social_Network_Ads` dataset and import it into R.
2. **Explore** and **Describe** the dataset (you can use `str()` and `summary()` functions, you can calculate and visualize the correlations, show some histograms, scatterplots, pie charts, etc..).



We will consider the variables `Age` and `EstimatedSalary` as input variables (features) to see the correlations between them and the decision of the user to buy (or not) the product.

3. Now we are going to split the dataset into training set and test set. Last week we did it manually. From now on we will split it randomly, you can use this code (after understanding it of course):

```
library(caTools) # install it first in the console
set.seed(123)
# we use the function set.seed() with the same seed number
# to randomly generate the same values, you already know that right?
#and you know why we want to generate the same values, am I wrong?
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
# here we chose the SplitRatio to 75% of the dataset,
# and 25% for the test set.
training_set = subset(dataset, split == TRUE)
# we use subset to split the dataset
test_set = subset(dataset, split == FALSE)
```

4. Scale the input variables in both training set and test set. Do you know what is scaling? Explain it one sentence.
5. Now fit a **simple** logistic regression model of `Purchased` in function of `Age`.
6. As you saw in the Logistic Regression chapter and in the previous question, we choose argument `family` to be `binomial` when we use the function `glm`. Explain why!
7. Write in a equation the model you obtained in question 5? (Read the following note concerning this question).



Note that there are three different ways to express the equation of a Logistic Regression model.

8. Is the feature `Age` significant to the model? Justify your answer.

9. What is the value of AIC of the model?



The **AIC** is the **Akaike Information Criterion**. You will use this while comparing multiple models. The model with lower value of AIC is better. Suppose that we have a statistical model of some data. Let  $\hat{L}$  be the maximum value of the likelihood function for the model; let  $k$  be the number of estimated parameters in the model. Then the AIC value of the model is the following.

$$\text{AIC} = 2k - 2 \ln(\hat{L})$$

where

- $\hat{L}$  = the maximized value of the likelihood function of the model  $M$ , i.e.  $\hat{L} = p(x|\hat{\beta}, M)$ , where  $\hat{\beta}$  are the parameter values that maximize the likelihood function.
- $x$  = the observed data.
- $k$  = the number of free parameters to be estimated. If the model under consideration is a linear regression,  $k$  is the number of regressors, including the intercept.

10. Plot `Purchased` in function of `Age` and add the curve of the obtained logistic regression model.

(**Hints:** One way to do it is to first plot the observations, then use the `curve()` function with option `add=TRUE` in order to add the curve to the plot. Pay attention that the argument “type” of the function `predict()` must be `response`)

You must obtain something like

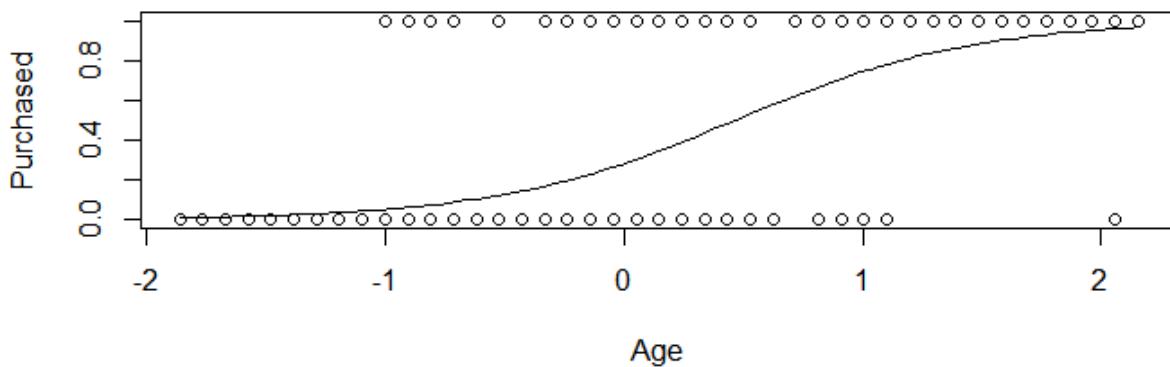


Figure 3.3:



**Extra:** A great R package for visualization if `ggplot2`. Take a look on this link for some examples. With this library we obtain the following plot for our model,

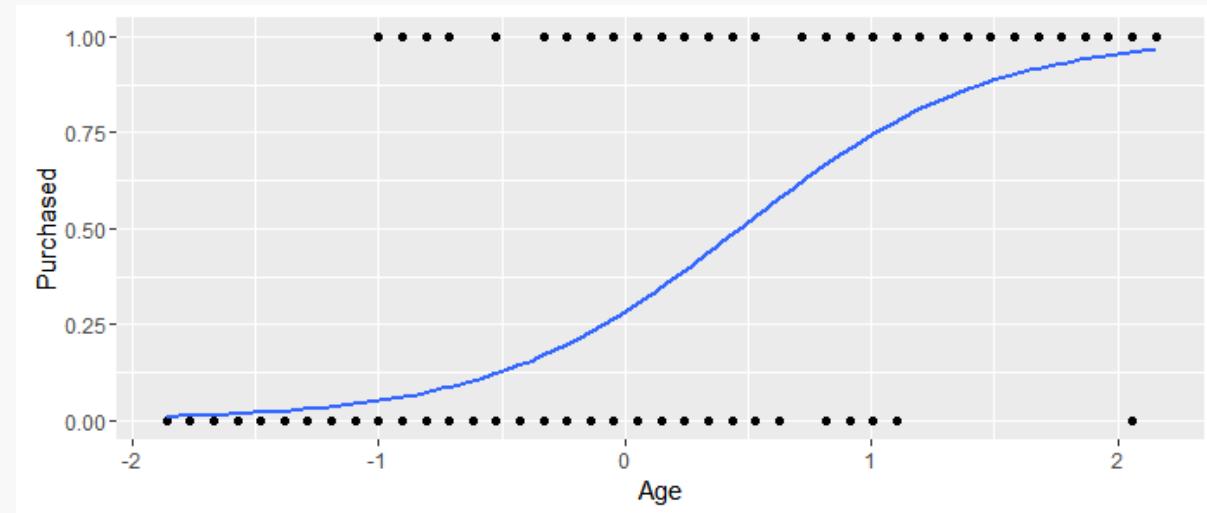


Figure 3.4:

I obtained the last figure with these lines of code,

```
library(ggplot2)
ggplot(training_set, aes(x=Age, y=Purchased)) +
  geom_point() +
  stat_smooth(method="glm", method.args=list(family="binomial"), se=FALSE)
```

**11.** Now let us take another feature into account in the model. Fit a logistic regression model of purchasing the product in function of the age of a user and its salary.

**12.** Are the predictors significant to the new model?

**13.** Do we obtain a better model by adding the estimated salary?

**14. Predictions:** On the test set, predict the probability of purchasing the product by the users using the obtained model.

**15.** Take a look on your predicted values for the variable `Purchased`. We predicted the probability that the user will purchase the product right? Now in order to compare your results with the real answers, transform the predicted values to 0 or 1 (1 if  $>0.5$ ).

**Hint:** You can easily do it with the `ifelse()` function.

Now read the following about the evaluation of a classifier (of a classification model).



**Confusion matrix:** is a tabular representation of Actual vs Predicted values. This helps us to find the accuracy of the model. The different results from a binary classifier are true positives, true negatives, false positives, and false negatives. This is how the confusion matrix looks like:

		True condition		Prevalence $= \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Positive predictive value (PPV), Precision $= \frac{\sum \text{True positive}}{\sum \text{Test outcome positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Test outcome positive}}$
		Total population	Condition positive			
Predicted condition	Predicted condition positive	True positive	False positive (Type I error)	False omission rate (FOR) $= \frac{\sum \text{False negative}}{\sum \text{Test outcome negative}}$	Negative predictive value (NPV) $= \frac{\sum \text{True negative}}{\sum \text{Test outcome negative}}$	
	Predicted condition negative	False negative (Type II error)	True negative	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR}^+}{\text{LR}^-}$	
	Accuracy (ACC) = $= \frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$	True positive rate (TPR), Sensitivity, Recall $= \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$	

(Image source: Wikipedia)

You can calculate the **accuracy** of your model with:

## True Positive + True Negatives

## True Positive + True Negatives + False Positives + False Negatives

Figure 3.5:

**Accuracy** is a key measure of performance, and is more specifically the rate at which the model is able to predict the correct value (classification or regression) for a given data point or observation. In other words, accuracy is the proportion of correct predictions out of all predictions made.

The other two metrics from the confusion matrix worth discussing are **Precision** and **Recall**.

Precision (positive predictive value) is the ratio of true positives to the total amount of positive predictions made (i.e., true or false). Said another way, precision measures the proportion of accurate positive predictions out of all positive predictions made.

Recall on the other hand, or true positive rate, is the ratio of true positives to the total amount of actual positives, whether predicted correctly or not. So in other words, recall measures the proportion of accurate positive predictions out of all actual positive observations.

A metric that is associated with precision and recall is called the F-score (also called F1 score), which combines them mathematically, and somewhat like a weighted average, in order to produce a single measure of performance based on the simultaneous values of both. Its values range from 0 (worst) to 1 (best).

Another important concept to know about is the **Receiver Operating Characteristic**, which when plotted, results in what's known as an **ROC** curve.

**ROC Curve:** An ROC curve is a two-dimensional plot of *sensitivity* (recall, or true positive rate) vs *1 - specificity* (false positive rate). The area under the curve is referred to as the **AUC**, and is a numeric metric used to represent the quality and performance of the classifier (model).

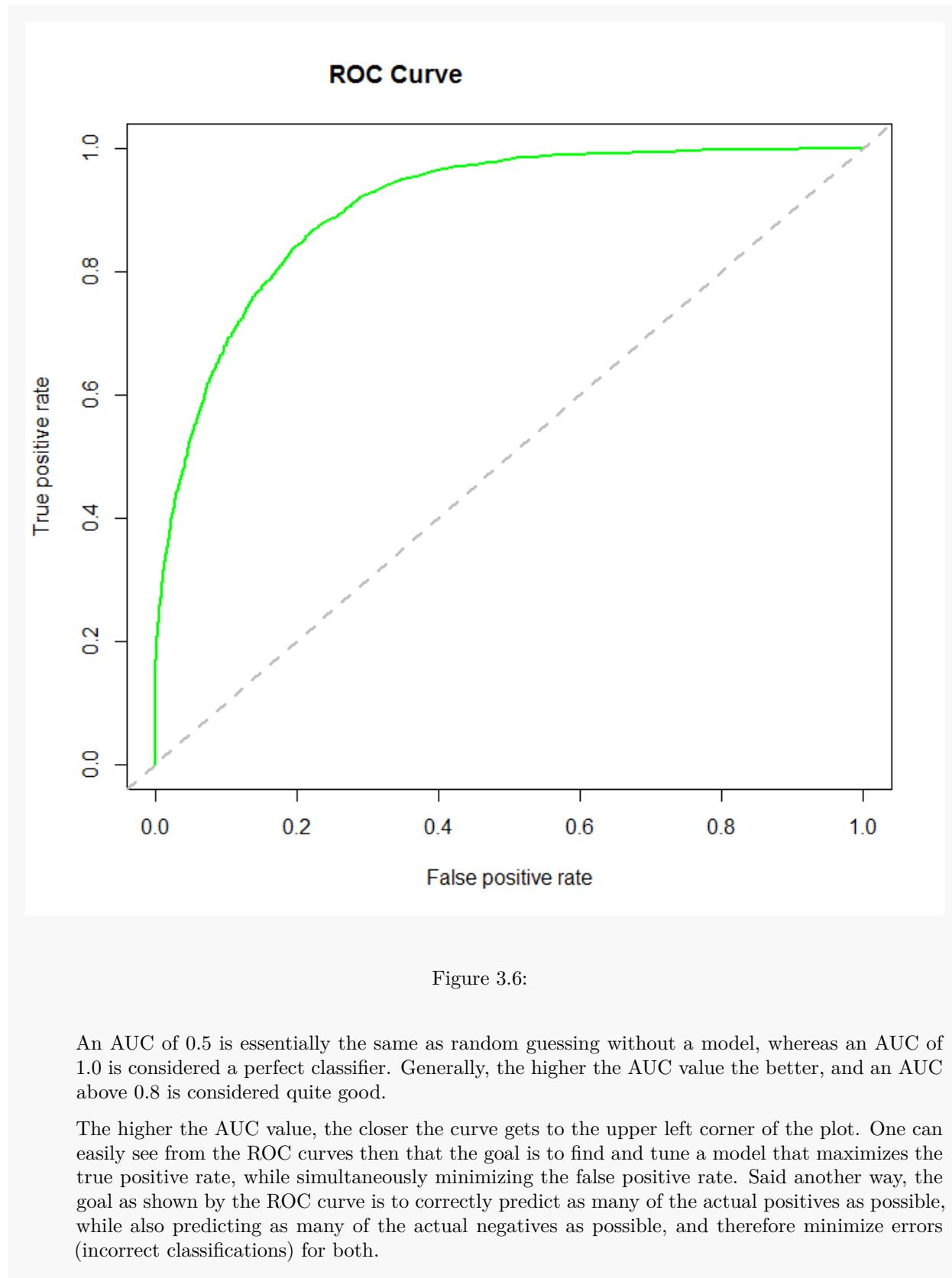


Figure 3.6:

An AUC of 0.5 is essentially the same as random guessing without a model, whereas an AUC of 1.0 is considered a perfect classifier. Generally, the higher the AUC value the better, and an AUC above 0.8 is considered quite good.

The higher the AUC value, the closer the curve gets to the upper left corner of the plot. One can easily see from the ROC curves then that the goal is to find and tune a model that maximizes the true positive rate, while simultaneously minimizing the false positive rate. Said another way, the goal as shown by the ROC curve is to correctly predict as many of the actual positives as possible, while also predicting as many of the actual negatives as possible, and therefore minimize errors (incorrect classifications) for both.

16. Now in order to evaluate the model and its predictions, compute the confusion matrix. What do you

obtain ?

(**Hint:** you can use the `table()` function).

**17.** Calculate the accuracy, specificity, sensitivity and the precision of the model.

(**Note:** You can create a function that takes a confusion matrix in input and returns the needed metrics)

**18.** Plot the ROC curve and calculate AUC value.



**Hints:** to plot it, install the `ROCR` package. Load and use the functions:

- `prediction()` to calculate the elements of the confusion matrix.
- `performance()` to calculate the AUC.
- `plot()` to plot the ROC curve, you can plot the performance calculated before.
- `abline()` to plot a line of equation  $y=x$ .

**19.** Compare the AUC of the two models you fitted (one with only age and one with age and estimated salary) and plot their ROC curves in the same figure.

# Chapter 4

## Discriminant Analysis

Discriminant analysis is a popular method for multiple-class classification. We will start first by the *Linear Discriminant Analysis (LDA)*.

### 4.1 Introduction

As we saw in the previous chapter, Logistic regression involves directly modeling  $\mathbb{P}(Y = k|X = x)$  using the *logistic function*, for the case of two response classes. In logistic regression, we model the conditional distribution of the response  $Y$ , given the predictor(s)  $X$ . We now consider an alternative and less direct approach to estimating these probabilities. In this alternative approach, **we model the distribution** of the predictors  $X$  **separately** in each of the response classes (i.e. given  $Y$ ), and then use **Bayes' theorem** to flip these around into estimates for  $\mathbb{P}(Y = k|X = x)$ . When these distributions are assumed to be *Normal*, it turns out that the model is very similar in form to logistic regression.

**Why not logistic regression?** Why do we need another method, when we have logistic regression? There are several reasons:

- When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.
- If  $n$  is small and the distribution of the predictors  $X$  is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
- Linear discriminant analysis is popular when we have *more than two response classes*.

### 4.2 Bayes' Theorem

Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

where  $A$  and  $B$  are events and  $P(B) \neq 0$ .

- $P(A|B)$ , a conditional probability, is the probability of observing event  $A$  given that  $B$  is true. It is called the **posterior** probability.
- $P(A)$ , is called the **prior**, is the initial degree of belief in A.
- $P(B)$  is the **likelihood**.



The posterior probability can be written in the memorable form as :

Posterior probability  $\propto$  Likelihood  $\times$  Prior probability.

### Extended form:

Suppose we have a partition  $\{A_i\}$  of the sample space, the even space is given or conceptualized in terms of  $P(A_j)$  and  $P(B|A_j)$ . It is then useful to compute  $P(B)$  using the law of total probability:

$$P(B) = \sum_j P(B|A_j)P(A_j)$$

$$\Rightarrow P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$$

### Bayes' Theorem for Classification:

Suppose that we wish to classify an observation into one of  $K$  classes, where  $K \geq 2$ . In other words, the qualitative response variable  $Y$  can take on  $K$  possible distinct and unordered values.

Let  $\pi_k$  represent the overall or *prior* probability that a randomly chosen observation comes from the  $k$ -th class; this is the probability that a given observation is associated with the  $k$ -th category of the response variable  $Y$ .

Let  $f_k(X) \equiv P(X = x|Y = k)$  denote the *density function* of  $X$  for an observation that comes from the  $k$ -th class. In other words,  $f_k(x)$  is relatively large if there is a high probability that an observation in the  $k$ -th class has  $X \approx x$ , and  $f_k(x)$  is small if it is very unlikely that an observation in the  $k$ -th class has  $X \approx x$ . Then *Bayes' theorem* states that

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{c=1}^K \pi_c f_c(x)} \quad (4.1)$$

As we did in the last chapter, we will use the abbreviation  $p_k(X) = P(Y = k|X)$ .

The equation above stated by *Bayes' theorem* suggests that instead of directly computing  $p_k(X)$  as we did in the logistic regression, we can simply plug in estimates of  $\pi_k$  and  $f_k(X)$  into the equation. In general, estimating  $\pi_k$  is easy (the fraction of the training observations that belong to the  $k$ -th class). But estimating  $f_k(X)$  tends to be more challenging.

Recall that  $p_k(x)$  is the *posterior* probability that an observation  $X = x$  belongs to  $k$ -th class.

If we can find a way to estimate  $f_k(X)$ , we can develop a classifier with the lowest possible error rate out of all classifiers.

## 4.3 LDA for $p = 1$

Assume that  $p = 1$ , which mean we have only one predictor. We would like to obtain an estimate for  $f_k(x)$  that we can plug into the Equation (4.1) in order to estimate  $p_k(x)$ . **We will then classify an observation to the class for which  $p_k(x)$  is greatest.**

In order to estimate  $f_k(x)$ , we will first make some assumptions about its form.

Suppose we assume that  $f_k(x)$  is *normal (Gaussian)*. In the one-dimensional setting, the normal density take the form

$$f_k(x) = \frac{1}{\sigma_k \sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right) \quad (4.2)$$

where  $\mu_k$  and  $\sigma_k^2$  are the mean and variance parameters for  $k$ -th class. Let us assume that  $\sigma_1^2 = \dots = \sigma_K^2 = \sigma$  (which means there is a shared variance term across all  $K$  classes). Plugging Eq. (4.2) into the Bayes formula in Eq. (4.1) we get,

$$p_k(x) = \frac{\pi_k \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{c=1}^K \pi_c \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_c}{\sigma}\right)^2}} \quad (4.3)$$



Note that  $\pi_k$  and  $\pi_c$  denote the prior probabilities. And  $\pi$  is the mathematical constant  $\pi \approx 3.14159$ .

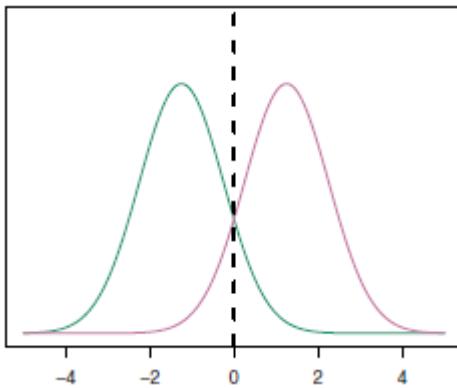
To classify at the value  $X = x$ , we need to see which of the  $p_k(x)$  is largest. Taking logs, and discarding terms that do not depend on  $k$ , we see that this is equivalent to assigning  $x$  to the class with the largest **discriminant score**:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \quad (4.4)$$

Note that  $\delta_k(x)$  is a *linear* function of  $x$ .



- The decision surfaces for a linear discriminant classifiers are defined by the linear equations  $\delta_k(x) = \delta_c(x)$ .
- Example: If  $K = 2$  and  $\pi_1 = \pi_2$ , then the **desicion boundary** is at  $x = \frac{\mu_1 + \mu_2}{2}$  (**Prove it!**).
- An example where  $\mu_1 = -1.5$ ,  $\mu_2 = 1.5$ ,  $\pi_1 = \pi_2 = 0.5$  and  $\sigma^2 = 1$  is shown in this following figure



- See this [video](#) to understand more about *decision boundary* (Applied on logistic regression).
- As we classify a new point according to which density is highest, when the priors are different we take them into account as well, and compare  $\pi_k f_k(x)$ . On the right of the following figure, we favor the pink class (remark that the decision boundary has shifted to the left).



## 4.4 Estimating the parameters

Typically we don't know these parameters; we just have the training data. In that case we simply estimate the parameters and plug them into the rule.

Let  $n$  the total number of training observations, and  $n_k$  the number of training observations in the  $k$ -th class. The following estimates are used:

$$\begin{aligned}\hat{\pi}_k &= \frac{n_k}{n} \\ \hat{\mu}_k &= \frac{1}{n_k} \sum_{i:y_i=k} x_i \\ \hat{\sigma}^2 &= \frac{1}{n-K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2 \\ &= \sum_{k=1}^K \frac{n_k - 1}{n-K} \cdot \hat{\sigma}_k^2\end{aligned}$$

where  $\hat{\sigma}_k^2 = \frac{1}{n_k - 1} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$  is the usual formula for the estimated variance in the  $-k$ -th class.

The linear discriminant analysis (LDA) classifier plugs these estimates in Eq. (4.4) and assigns an observation  $X = x$  to the class for which

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k) \quad (4.5)$$

is largest.

The *discriminant functions* in Eq. (4.5) are linear functions of  $x$ .

Recall that we assumed that the observations come from a normal distribution with a common variance  $\sigma^2$ .

## 4.5 LDA for $p > 1$

Let us now suppose that we have multiple predictors. We assume that  $X = (X_1, X_2, \dots, X_p)$  is drawn from *multivariate Gaussian* distribution (assuming they have a common covariance matrix, e.g. same variances as in the case of  $p = 1$ ). The multivariate Gaussian distribution assumes that each individual predictor follows a one-dimensional normal distribution as in Eq. (4.2), with some correlation between each pair of predictors.

To indicate that a  $p$ -dimensional random variable  $X$  has a multivariate Gaussian distribution, we write  $X \sim \mathcal{N}(\mu, \Sigma)$ . Where

$$\mu = E(X) = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{pmatrix}$$

and,

$$\Sigma = Cov(X) = \begin{pmatrix} \sigma_1^2 & Cov[X_1, X_2] & \dots & Cov[X_1, X_p] \\ Cov[X_2, X_1] & \sigma_2^2 & \dots & Cov[X_2, X_p] \\ \vdots & \vdots & \ddots & \vdots \\ Cov[X_p, X_1] & Cov[X_p, X_2] & \dots & \sigma_p^2 \end{pmatrix}$$

$\Sigma$  is the  $p \times p$  covariance matrix of  $X$ .

Formally, the multivariate Gaussian density is defined as

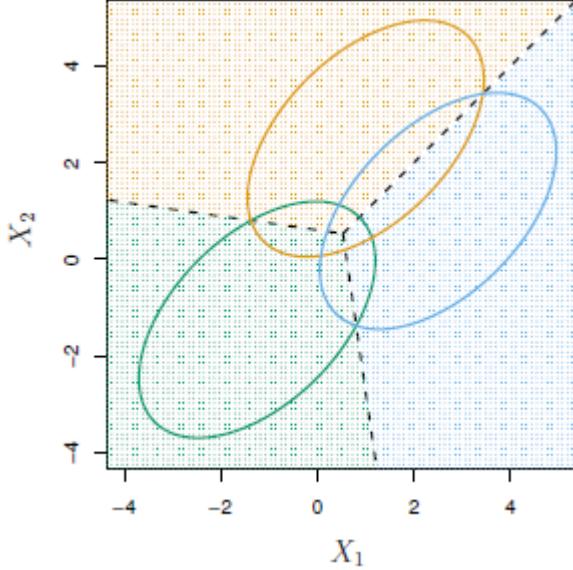
$$f(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

Plugging the density function for the  $k$ -th class,  $f_k(X = x)$ , into Eq. (4.1) reveals that the Bayes classifier assigns an observation  $X = x$  to the class for which

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \quad (4.6)$$

is largest. This is the vector/matrix version of (4.4).

An example is shown in the following figure. Three equally-sized Gaussian classes are shown with class-specific mean vectors and a common covariance matrix ( $\pi_1 = \pi_2 = \pi_3 = 1/3$ ). The three ellipses represent regions that contain 95% of the probability for each of the three classes. The dashed lines are the Bayes decision boundaries.



Recall that the decision boundaries represent the set of values  $x$  for which  $\delta_k(x) = \delta_c(x)$ ; i.e. for  $k \neq c$

$$x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k = x^T \Sigma^{-1} \mu_c - \frac{1}{2} \mu_c^T \Sigma^{-1} \mu_c$$

Note that there are three lines representing the Bayes decision boundaries because there are three pairs of classes among the three classes. That is, one Bayes decision boundary separates class 1 from class 2, one separates class 1 from class 3, and one separates class 2 from class 3. These three Bayes decision boundaries divide the predictor space into three regions. The Bayes classifier will classify an observation according to the region in which it is located.

Once again, we need to estimate the unknown parameters  $\mu_1, \dots, \mu_k$ , and  $\pi_1, \dots, \pi_k$ , and  $\Sigma$ ; the formulas are similar to those used in the one-dimensional case. To assign a new observation  $X = x$ , LDA plugs these estimates into Eq. (4.6) and classifies to the class for which  $\delta_k(x)$  is largest.

Note that in Eq. (4.6)  $\delta_k(x)$  is a *linear* function of  $x$ ; that is, the LDA decision rule depends on  $x$  only through a linear combination of its elements. This is the reason for the word linear in LDA.

## 4.6 Making predictions

Once we have estimates  $\hat{\delta}_k(x)$ , we can turn these into estimates for class probabilities:

$$\hat{P}(Y = k | X = x) = \frac{e^{\hat{\delta}_k(x)}}{\sum_{c=1}^K e^{\hat{\delta}_c(x)}}$$

So classifying to the largest  $\hat{\delta}_k(x)$  amounts to classifying to the class for which  $\hat{P}(Y = k | X = x)$  is largest.

When  $K = 2$ , we classify to class 2 if  $\hat{P}(Y = 2 | X = x) \geq 0.5$ , else to class 1.

## 4.7 Other forms of Discriminant Analysis

$$P(Y = k | X = x) = \frac{\pi_k f_k(x)}{\sum_{c=1}^K \pi_c f_c(x)}$$

We saw before that when  $f_k(x)$  are Gaussian densities, with the same covariance matrix  $\Sigma$  in each class, this leads to Linear Discriminant Analysis (LDA).

By altering the forms for  $f_k(x)$ , we get different classifiers.

- With Gaussians but different  $\Sigma_k$  in each class, we get *Quadratic Discriminant Analysis (QDA)*.
- With  $f_k(x) = \prod_{j=1}^p f_{jk}(x_j)$  (conditional independence model) in each class we get *Naive Bayes*. (For Gaussian, this means the  $\Sigma_k$  are diagonal, e.g.  $Cov(X_i, X_j) = 0 \forall 1 \leq i, j \leq p$ ).
- Many other forms by proposing specific density models for  $f_k(x)$ , including *nonparametric approaches*.

### 4.7.1 Quadratic Discriminant Analysis (QDA)

Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a Gaussian distribution, and plugging estimates for the parameters into Bayes' theorem in order to perform prediction.

However, unlike LDA, QDA assumes that each class has its own covariance matrix. Under this assumption, the Bayes classifier assigns an observation  $X = x$  to the class for which

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x - \mu)^T \Sigma_k^{-1}(x - \mu) - \frac{1}{2} \log |\Sigma_k| + \log \pi_k \\ &= -\frac{1}{2}x^T \Sigma_k^{-1}x + \frac{1}{2}x^T \Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma_k^{-1}\mu_k - \frac{1}{2} \log |\Sigma_k| + \log \pi_k\end{aligned}$$

is largest.

Unlike in LDA, the quantity  $x$  appears as a quadratic function in QDA. This is where QDA gets its name.



The decision boundary in QDA is non-linear. It is quadratic (a curve).

### 4.7.2 Naive Bayes

We use Naive Bayes classifier if the features are independent in each class. It is useful when  $p$  is large (unlike LDA and QDA).

Naive Bayes assumes that each  $\Sigma_k$  is diagonal, so

$$\begin{aligned}\delta_k(x) &\propto \log \left[ \pi_k \prod_{j=1}^p f_{kj}(x_j) \right] \\ &= -\frac{1}{2} \sum_{j=1}^p \frac{(x_j - \mu_{kj})^2}{\sigma_{kj}^2} + \log \pi_k\end{aligned}$$

It can be used for mixed feature vectors (qualitative and quantitative). If  $X_j$  is qualitative, we replace  $f_{kj}(x_j)$  by probability mass function (histogram) over discrete categories.

## 4.8 LDA vs Logistic Regression

the logistic regression and LDA methods are closely connected. Consider the two-class setting with  $p = 1$  predictor, and let  $p_1(x)$  and  $p_2(x) = 1 - p_1(x)$  be the probabilities that the observation  $X = x$  belongs to class

1 and class 2, respectively. In the LDA framework, we can see from Eq. (4.4) (and a bit of simple algebra) that the log odds is given by

$$\log \left( \frac{p_1(x)}{1 - p_1(x)} \right) = \log \left( \frac{p_1(x)}{p_2(x)} \right) = c_0 + c_1 x$$

where  $c_0$  and  $c_1$  are functions of  $\mu_1, \mu_2$ , and  $\sigma^2$ .

On the other hand, we know that in logistic regression

$$\log \left( \frac{p_1}{1 - p_1} \right) = \beta_0 + \beta_1 x$$

Both of the equations above are linear functions of  $x$ . Hence both logistic regression and LDA produce linear decision boundaries. The only difference between the two approaches lies in the fact that  $\beta_0$  and  $\beta_1$  are estimated using *maximum likelihood*, whereas  $c_0$  and  $c_1$  are computed using the estimated mean and variance from a *normal distribution*. This same connection between LDA and logistic regression also holds for multidimensional data with  $p > 1$ .



- Logistic regression uses the conditional likelihood based on  $P(Y|X)$  (known as *discriminative learning*).
- LDA uses the full likelihood based on  $P(X, Y)$  (known as *generative learning*).
- Despite these differences, in practice the results are often very similar.

**Remark:** Logistic regression can also

fit quadratic boundaries like QDA, by explicitly including quadratic terms in the model.

# PW 4

This week we are going to continue the analysis of the `Social_Network_Ads` dataset . Recall that this dataset contains informations of users of a social network and if they bought a specified product. Last week we built a Logistic Regression model for the variable `Purchased` in function of `Age` and `EstimatedSalary`. We will consider the same variables this week but we will fit different models using methods such as LDA, QDA, and Naive Bayes.

## Logistic Regression

1. First, let's do the pre-processing steps you were asked to do during the last session and fit a **logistic regression** model. Please read and understand very well the following code (read the comments!). Then copy what is necessary for today's session to your report (but remove my comments!).

If you lost the dataset, you can download it from [here](#).

```
# Loading the dataset.. I have putted it into a folder called "datasets"
dataset <- read.csv('http://www.mghassany.com/MLcourse/datasets/Social_Network_Ads.csv')

# Describing and Exploring the dataset

str(dataset) # to show the structure of the dataset.
summary(dataset) # will show some statistics of every column.
# Remark what it shows when the column is a numerical or categorical variable.
# Remark that it has no sense for the variable User.ID

boxplot(Age ~ Purchased, data=dataset, col = "blue", main="Boxplot Age ~ Purchased")
# You know what is a boxplot right? I will let you interpret it.
boxplot(EstimatedSalary ~ Purchased, data=dataset,col = "red", main="Boxplot EstimatedSalary ~ Purchased")
# Another boxplot

aov(EstimatedSalary ~Purchased, data=dataset)
# Anova test, but we need to show the summary of
# it in order to see the p-value and to interpret.

summary(aov(EstimatedSalary ~Purchased, data=dataset))
# What do you conclude ?
# Now another anova test for the variable Age
summary(aov(Age ~Purchased, data=dataset))

# There is a categorical variable in the dataset, which is Gender.
# Of course we cannot show a boxplot of Gender and Purchased.
# But we can show a table, or a mosaic plot, both tell the same thing.
```

```



```

```

# prediction
pred.glm = predict(classifier.logreg, newdata = test_set[,-3], type="response")
# Do not forget to put type response.
# By the way, you know what you get when you do not put it, right?

# Now let's assign observations to classes with respect to the probabilities
pred.glm_0_1 = ifelse(pred.glm >= 0.5, 1,0)
# I created a new vector, because we need the probabilities later for the ROC curve.

# show some values of the vectors
head(pred.glm)
head(pred.glm_0_1)

# confusion matrix
cm = table(test_set[,3], pred.glm_0_1)
cm
# First line to store it into cm, second line to show the matrix!

# You remember my note about table() function and the order of the arguments?
cm = table(pred.glm_0_1, test_set[,3])
cm

# You can show the confusion matrix in a mosaic plot by the way
mosaicplot(cm,col=sample(1:8,2)) # colors are random between 8 colors.

# ROC
require(ROCR)
score <- prediction(pred.glm,test_set[,3]) # we use the predicted probabilities not the 0 or 1
performance(score,"auc") # y.values
plot(performance(score,"tpr","fpr"),col="green")
abline(0,1,lty=8)

```

So now we have a logistic regression model stored in `classifier.logreg`. It is a model of `Purchased` in function of `Age` and `EstimatedSalary`. We will use this model to show the decision boundary in the next part of this PW. Then we will compare this model to other models obtained by Discriminant Analysis approaches.

## Decision Boundary of Logistic Regression

Now you are going to visualize the decision boundary for logistic regression.



- Since the decision boundary of logistic regression is a linear (*you know why right?*) and the dimension of the feature space is 2 (`Age` and `EstimatedSalary`), the decision boundary in this 2-dimensional space is a line that separates the predicted classes “0” and “1” (values of the response `Purchased`).
- For logistic regression, we predict  $y = 1$  if  $\beta^T X \geq 0$  (right side of the line) and  $y = 0$  if  $\beta^T X < 0$  (left side of the line). Where

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} \text{ and } X = \begin{pmatrix} 1 \\ X_1 \\ X_2 \end{pmatrix}$$

So we predict  $y = 1$  if  $\beta_0 + \beta_1 X_1 + \beta_2 X_2 \geq 0$  which means that the equation of the decision boundary (a line here) is  $X_2 = -\frac{\beta_1}{\beta_2}X_1 - \frac{\beta_0}{\beta_2}$

**2.** Plot the decision boundary obtained with logistic regression. In order to do so, calculate the intercept and the slope of the line presenting the decision boundary, then plot `EstimatedSalary` in function of `Age` (from the `test_set`) and add the line using `abline()`.

**3.** In order to verify that your line (decision boundary) is well plotted, color the points on the last Figure with respect to the predicted response.

**Hints:**

- If your predictions are stored in `y_pred`, you can do it using `bg = ifelse(y_pred == 1, 'color1', 'color2')`, and precise the argument `pch` to be 21 (you can choose `pch` to be a value between 21 and 25, try it).
- Then, add the line using `abline()`, put the line width = 2 to make it more visible. Do not forget to title the Figure).

**4.** Now make the same plot but color the points with respect to their real labels (the variable `Purchased`). From this figure, count the number of the false positive predictions and compare it to the value obtained in the confusion matrix.

## Linear Discriminant Analysis (LDA)

Let us apply linear discriminant analysis (LDA) now. First we will make use of the `lda()` function in the package `MASS`. Second, you are going to create the model and predict the classes by yourself without using the `lda()` function. And we will visualize the decision boundary of LDA.

**5.** Fit a LDA model of `Purchased` in function of `Age` and `EstimatedSalary`. Name the model `classifier.lda`.

```
library(MASS)
classifier.lda <- lda(Purchased~Age+EstimatedSalary, data=training_set)
```

**6.** Call `classifier.lda` and see what does it compute.

**Plus:** If you enter the following you will be returned with a list of summary information concerning the computation:

```
classifier.lda$prior
classifier.lda$means
```

**7.** On the test set, predict the probability of purchasing the product by the users using the model `classifier.lda`. Remark that when we predict using LDA, we obtain a list instead of a matrix, do `str()` for your predictions to see what do you get.

**Remark:** we get the predicted class here, without being obligated to round the predictions as we did for logistic regression.

**8.** Compute the confusion matrix and compare the predictions results obtained by LDA to the ones obtained by logistic regression. What do you remark?

(**Hint:** compare the accuracy)

**9.** Now let us plot the decision boundary obtained with LDA. You saw in the course that decision boundary for LDA represent the set of values  $x$  where  $\delta_k(x) = \delta_c(x)$ . Recall that

$$\delta_k(X) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

Here in our case, we have 2 classes ( $K = 2$ ) and 2 predictors ( $p = 2$ ). So the decision boundary (which is linear in the case of LDA, and line in our case since  $p = 2$ ) will verify the equation  $\delta_0(x) = \delta_1(x)$ . Since we have two classes “0” and “1”. In the case of LDA this leads to linear boundary and is easy to be plotted. But in more complicated cases it is difficult to manually simplify the equations and plot the decision boundary. Anyway, there is a smart method to plot (but a little bit costly) the decision boundary in R using the function `contour()`, the corresponding code is the following (you must adapt it and use it to plot your decision boundary):

```
# create a grid corresponding to the scales of Age and EstimatedSalary
# and fill this grid with lot of points
X1 = seq(min(training_set[, 1]) - 1, max(training_set[, 1]) + 1, by = 0.01)
X2 = seq(min(training_set[, 2]) - 1, max(training_set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
# Adapt the variable names
colnames(grid_set) = c('Age', 'EstimatedSalary')

# plot 'Estimated Salary' ~ 'Age'
plot(test_set[, 1:2],
      main = 'Decision Boundary LDA',
      xlab = 'Age', ylab = 'Estimated Salary',
      xlim = range(X1), ylim = range(X2))

# color the plotted points with their real label (class)
points(test_set[1:2], pch = 21, bg = ifelse(test_set[, 3] == 1, 'green4', 'red3'))

# Make predictions on the points of the grid, this will take some time
pred_grid = predict(classifier.lda, newdata = grid_set)$class

# Separate the predictions by a contour
contour(X1, X2, matrix(as.numeric(pred_grid), length(X1), length(X2)), add = TRUE)
```

**10.** Now let us build a LDA model for our data set without using the `lda()` function. You are free to do it by creating a function or without creating one. Go back to question **6** and see what did you obtain by using `lda()`. It computes the prior probability of group membership and the estimated group means for each of the two groups. Additional information that is not provided, but may be important, is the single covariance matrix that is being used for the various groupings.



In LDA, we compute for every observation  $x$  its discriminant score  $\delta_k(x)$ . Then we attribute  $x$  to the class that has the highest  $\delta$ . Recall that

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

So to compute  $\delta_k(x)$  we need to estimate  $\pi_k$ ,  $\mu_k$  and  $\Sigma$ .

Note that

$$x = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

and here  $X_1=\text{Age}$  and  $X_2=\text{EstimatedSalary}$ .

So let us do it step by step, first we will do the estimates:

**10.1** Subset the training set into two sets: `class0` where `Purchased = 0` and `class1` where `Purchased = 1`.

**10.2** Compute  $\pi_0$  and  $\pi_1$ .

$$\pi_i = N_i/N, \text{ where } N_i \text{ is the number of data points in group } i$$

**10.3** Compute  $\mu_0$  and  $\mu_1$ .

$$\mu_0 = \begin{pmatrix} \mu_0(X_1) \\ \mu_0(X_2) \end{pmatrix} \text{ and } \mu_1 = \begin{pmatrix} \mu_1(X_1) \\ \mu_1(X_2) \end{pmatrix}$$

where, for example,  $\mu_0(X_1)$  is the mean of the variable  $X_1$  in the group 0 (the subset `class0`).

**10.4** Compute  $\Sigma$ . In the case of two classes like here, it is computed by calculating the following:

$$\Sigma = \frac{(N_0 - 1)\Sigma_0 + (N_1 - 1)\Sigma_1}{N_0 + N_1 - 2}$$

where  $\Sigma_i$  is the estimated covariance matrix for specific group  $i$ .

**Remark:** Recall that in LDA we use the same  $\Sigma$ . But in QDA we do not.

**10.5.** Now that we have computed all the needed estimates, we can calculate  $\delta_0(x)$  and  $\delta_1(x)$  for any observation  $x$ . And we will attribute  $x$  to the class with the highest  $\delta$ . First, try it for  $x$  where  $x^T = (1, 1.5)$ , what is class prediction for this specific  $x$ ?

**10.6.** Compute the discriminant scores  $\delta$  for the test set (a matrix  $100 \times 2$ ), predict the classes and compare your results with the results obtained with the `lda()` function.

## Quadratic Discriminant Analysis (QDA)

Training and assessing a QDA model in R is very similar in syntax to training and assessing a LDA model. The only difference is in the function name `qda()`

**11.** Fit a QDA model of `Purchased` in function of `Age` and `EstimatedSalary`. Name the model `classifier.qda`.

```
# qda() is a function of library(MASS)
classifier.qda <- qda(Purchased~, data = training_set)
```

**12.** Make predictions on the `test_set` using the QDA model `classifier.qda`. Show the confusion matrix and compare the results with the predictions obtained using the LDA model `classifier.lda`.

**13.** Plot the decision boundary obtained with QDA. Color the points with the real labels.

## Comparison

**14.** In order to compare the methods we used, plot on the same Figure the ROC curve for each classifier we fitted and compare the correspondant AUC. What was the best model for this dataset? Can you justify it?

**Remark:** If you use the `ROCR` package:

- For Logistic regression, use the predicted probabilities in the `prediction()` (and not the round values “0” or “1”).
- For LDA and QDA, put `pred.lda$posterior[,2]` in the `prediction()` function (those are the posterior probabilities that observations belong to class “1”).

# Chapter 5

## Special Session

This session is special. You are given a dataset on Devinci Online, go download it and analyze it. You must submit a report (html) of your analysis at the end of the session. This report is the most important one for your evaluation.

Pay attention to the objective of the dataset, your analysis depends on your dataset's understanding. The data analysis must include all the following steps: importing the dataset, exploring the dataset and describing it, data preprocessing, data splitting, building models and assessing models.

Here are somes rules for this special session:

- The dataset will be available on Devinci Online during the session.
- When you use the function `set.seed()` to split the dataset, use your **student's number**.
- The professor will not give you any hint. You can ask questions but the professor will not answer all of your questions. It depends on the question.
- Of course every student must work individually and no discussions are allowed.
- You have the right to access internet, the course's website, and even googling some R tips for your codes. But you are not allowed to open any Social Network website neither your email. If you open one of these you will get directly zero.
- The professor has the right to verify by himself if you are opening any Social network/email website.
- The report must be written in english of course.
- There won't be a quiz in this special session in order to give you the complete 3 hours to analyze the dataset.
- Your report must be clear. Every step or decision you make must be explained. Your code must be commented and your results must be interpreted.

Good Luck!



## **Part II**

# **Dimensionality Reduction**



# Chapter 6

# Principal Components Analysis

## 6.1 Introduction

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the **variation** present in the data set. This is achieved by *transforming* to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables.

Suppose that we wish to visualize  $n$  observations with measurements on a set of  $p$  features,  $X_1, X_2, \dots, X_p$ , as part of an exploratory data analysis. We could do this by examining two-dimensional scatterplots of the data, each of which contains the  $n$  observations' measurements on two of the features. However, there are  $C_p^2 = p(p+1)/2$  such scatterplots. For example, with  $p = 10$  there are 45 plots! If  $p$  is large, then it will certainly not be possible to look at all of them; moreover, most likely none of them will be informative since they each contain just a small fraction of the total information present in the data set. Clearly, a better method is required to visualize the  $n$  observations when  $p$  is large. In particular, we would like to find a low-dimensional representation of the data that captures as much of the information as possible. PCA provides a tool to do just this.

PCA finds a low-dimensional representation of a data set that contains as much as possible of the variation. The idea is that each of the  $n$  observations lives in  $p$ -dimensional space, but not all of these dimensions are equally interesting. PCA seeks a small number of dimensions that are as interesting as possible, where the concept of interesting is measured by the amount that the observations vary along each dimension. Each of the dimensions found by PCA is a **linear combination of the  $p$  features**. We now explain the manner in which these dimensions, or principal components, are found.

## 6.2 Principal Components

### Notations and Procedure

Suppose that we have a random vector of the features  $X$ .

$$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{pmatrix}$$

with population variance-covariance matrix

$$\text{var}(\mathbf{X}) = \Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1p} \\ \sigma_{21} & \sigma_2^2 & \dots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \dots & \sigma_p^2 \end{pmatrix}$$

Consider the linear combinations

$$\begin{aligned} Y_1 &= a_{11}X_1 + a_{12}X_2 + \dots + a_{1p}X_p \\ Y_2 &= a_{21}X_1 + a_{22}X_2 + \dots + a_{2p}X_p \\ &\vdots \\ Y_p &= a_{p1}X_1 + a_{p2}X_2 + \dots + a_{pp}X_p \end{aligned}$$

Note that  $Y_i$  is a function of our random data, and so is also random. Therefore it has a population variance

$$\text{var}(Y_i) = \sum_{k=1}^p \sum_{l=1}^p a_{ik}a_{il}\sigma_{kl} = \mathbf{a}_i^T \Sigma \mathbf{a}_i$$

Moreover,  $Y_i$  and  $Y_j$  will have a population covariance

$$\text{cov}(Y_i, Y_j) = \sum_{k=1}^p \sum_{l=1}^p a_{ik}a_{jl}\sigma_{kl} = \mathbf{a}_i^T \Sigma \mathbf{a}_j$$

and a correlation

$$\text{cor}(Y_i, Y_j) = \frac{\text{cov}(Y_i, Y_j)}{\sigma_i^2 \sigma_j^2}$$

Here the coefficients  $a_{ij}$  are collected into the vector

$$\mathbf{a}_i = \begin{pmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{ip} \end{pmatrix}$$

The coefficients  $a_{ij}$  are also called *loadings* of the principal component  $i$  and  $\mathbf{a}_i$  is a principal component loading vector.



- The total variation of  $X$  is the *trace* of the variance-covariance matrix  $\Sigma$ .
- The trace of  $\Sigma$  is the sum of the variances of the individual variables.
- $\text{trace}(\Sigma) = \sigma_1^2 + \sigma_2^2 + \dots + \sigma_p^2$

### First Principal Component ( $\text{PC}_1$ ): $Y_1$

The *first principal component* is the *normalized* linear combination of the features  $X_1, X_2, \dots, X_p$  that has maximum variance (among all linear combinations), so it accounts for as much variation in the data as possible.

Specifically we will define coefficients  $a_{11}, a_{12}, \dots, a_{1p}$  for that component in such a way that its variance is maximized, subject to the constraint that the sum of the squared coefficients is equal to one (that is what we mean by *normalized*). This constraint is required so that a unique answer may be obtained.

More formally, select  $a_{11}, a_{12}, \dots, a_{1p}$  that maximizes

$$\text{var}(Y_1) = \mathbf{a}_1^T \Sigma \mathbf{a}_1 = \sum_{k=1}^p \sum_{l=1}^p a_{1k} a_{1l} \sigma_{kl}$$

subject to the constraint that

$$\sum_{j=1}^p a_{1j}^2 = \mathbf{a}_1^T \mathbf{a}_1 = 1$$

### Second Principal Component ( $\text{PC}_2$ ): $Y_2$

The *second principal component* is the linear combination of the features  $X_1, X_2, \dots, X_p$  that accounts for as much of the remaining variation as possible, with the constraint that the correlation between the first and second component is 0. So the second principal component has maximal variance out of all linear combinations that are uncorrelated with  $Y_1$ .

To compute the coefficients of the second principal component, we select  $a_{21}, a_{22}, \dots, a_{2p}$  that maximizes the variance of this new component

$$\text{var}(Y_2) = \sum_{k=1}^p \sum_{l=1}^p a_{2k} a_{2l} \sigma_{kl} = \mathbf{a}_2^T \Sigma \mathbf{a}_2$$

subject to:

- The constraint that the sums of squared coefficients add up to one,  $\sum_{j=1}^p a_{2j}^2 = \mathbf{a}_2^T \mathbf{a}_2 = 1$ .
- Along with the additional constraint that these two components will be uncorrelated with one another:

$$\text{cov}(Y_1, Y_2) = \mathbf{a}_1^T \Sigma \mathbf{a}_2 = \sum_{k=1}^p \sum_{l=1}^p a_{1k} a_{2l} \sigma_{kl} = 0$$

All subsequent principal components have this same property: they are linear combinations that account for as much of the remaining variation as possible and they are not correlated with the other principal components.

We will do this in the same way with each additional component. For instance:

*i<sup>th</sup>* Principal Component ( $\text{PC}_i$ ):  $Y_i$ 

We select  $a_{i1}, a_{i2}, \dots, a_{ip}$  that maximizes

$$\text{var}(Y_i) = \sum_{k=1}^p \sum_{l=1}^p a_{ik} a_{il} \sigma_{kl} = \mathbf{a}_i^T \Sigma \mathbf{a}_i$$

subject to the constraint that the sums of squared coefficients add up to one, along with the additional constraint that this new component will be uncorrelated with all the previously defined components:

$$\begin{aligned} \sum_{j=1}^p a_{ij}^2 \mathbf{a}_i^T \mathbf{a}_i &= \mathbf{a}_i^T \mathbf{a}_i = 1 \\ \text{cov}(Y_1, Y_i) &= \sum_{k=1}^p \sum_{l=1}^p a_{1k} a_{il} \sigma_{kl} = \mathbf{a}_1^T \Sigma \mathbf{a}_i = 0 \\ \text{cov}(Y_2, Y_i) &= \sum_{k=1}^p \sum_{l=1}^p a_{2k} a_{il} \sigma_{kl} = \mathbf{a}_2^T \Sigma \mathbf{a}_i = 0 \\ &\vdots \\ \text{cov}(Y_{i-1}, Y_i) &= \sum_{k=1}^p \sum_{l=1}^p a_{i-1,k} a_{il} \sigma_{kl} = \mathbf{a}_{i-1}^T \Sigma \mathbf{a}_i = 0 \end{aligned}$$

Therefore all principal components are uncorrelated with one another.

### 6.3 How do we find the coefficients?

How do we find the coefficients  $a_{ij}$  for a principal component? The solution involves the **eigenvalues** and **eigenvectors** of the variance-covariance matrix  $\Sigma$ .

Let  $\lambda_1, \dots, \lambda_p$  denote the eigenvalues of the variance-covariance matrix  $\Sigma$ . These are ordered so that  $\lambda_1$  has the largest eigenvalue and  $\lambda_p$  is the smallest.

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$$

We are also going to let the vectors  $\mathbf{a}_1, \dots, \mathbf{a}_p$  denote the corresponding eigenvectors.

It turns out that the elements for these eigenvectors will be the coefficients of the principal components.



The elements for the eigenvectors of  $\Sigma$  are the coefficients of the principal components.

The variance for the  $i$ th principal component is equal to the  $i$ th eigenvalue.

$$\text{var}(Y_i) = \text{var}(a_{i1}X_1 + a_{i2}X_2 + \dots + a_{ip}X_p) = \lambda_i$$

Moreover, the principal components are uncorrelated with one another.

$$\text{cov}(Y_i, Y_j) = 0$$

The variance-covariance matrix may be written as a function of the eigenvalues and their corresponding eigenvectors. In fact, the variance-covariance matrix can be written as the sum over the  $p$  eigenvalues, multiplied by the product of the corresponding eigenvector times its transpose as shown in the following expression

$$\Sigma = \sum_{i=1}^p \lambda_i \mathbf{a}_i \mathbf{a}_i^T$$

If  $\lambda_{k+1}, \lambda_{k+2}, \dots, \lambda_p$  are small, we might approximate  $\Sigma$  by

$$\Sigma \cong \sum_{i=1}^k \lambda_i \mathbf{a}_i \mathbf{a}_i^T$$

Earlier in the chapter we defined the total variation of  $X$  as the trace of the variance-covariance matrix. This is also equal to the sum of the eigenvalues as shown below:

$$\begin{aligned} \text{trace}(\Sigma) &= \sigma_1^2 + \sigma_2^2 + \dots + \sigma_p^2 \\ &= \lambda_1 + \lambda_2 + \dots + \lambda_p \end{aligned}$$

This will give us an interpretation of the components in terms of the amount of the full variation explained by each component. The proportion of variation explained by the  $i$ th principal component is then going to be defined to be the eigenvalue for that component divided by the sum of the eigenvalues. In other words, the  $i$ th principal component explains the following proportion of the total variation:

$$\frac{\lambda_i}{\lambda_1 + \lambda_2 + \dots + \lambda_p}$$

A related quantity is the proportion of variation explained by the first  $k$  principal component. This would be the sum of the first  $k$  eigenvalues divided by its total variation.

$$\frac{\lambda_1 + \lambda_2 + \dots + \lambda_k}{\lambda_1 + \lambda_2 + \dots + \lambda_p}$$

In practice, these proportions are often expressed as percentages.

Naturally, if the proportion of variation explained by the first  $k$  principal components is large, then not much information is lost by considering only the first  $k$  principal components.

## Why It May Be Possible to Reduce Dimensions

When we have correlations (multicollinearity) between the features, the data may more or less fall on a line or plane in a lower number of dimensions. For instance, imagine a plot of two features that have a nearly perfect correlation. The data points will fall close to a straight line. That line could be used as a new (one-dimensional) axis to represent the variation among data points.



All of this is defined in terms of the population variance-covariance matrix  $\Sigma$  which is *unknown*. However, we may estimate  $\Sigma$  by the sample variance-covariance matrix which is given in the standard formula here:

$$\mathbf{S} = \frac{1}{n-1} \sum_{i=1}^n (\mathbf{X}_i - \bar{\mathbf{x}})(\mathbf{X}_i - \bar{\mathbf{x}})^T$$

## Procedure

Compute the eigenvalues  $\hat{\lambda}_1, \hat{\lambda}_2, \dots, \hat{\lambda}_p$  of the sample variance-covariance matrix  $\mathbf{S}$ , and the corresponding eigenvectors  $\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \dots, \hat{\mathbf{a}}_p$ .

Then we will define our estimated principal components using the eigenvectors as our coefficients:

$$\begin{aligned}\hat{Y}_1 &= \hat{a}_{11}X_1 + \hat{a}_{12}X_2 + \cdots + \hat{a}_{1p}X_p \\ \hat{Y}_2 &= \hat{a}_{21}X_1 + \hat{a}_{22}X_2 + \cdots + \hat{a}_{2p}X_p \\ &\vdots \\ \hat{Y}_p &= \hat{a}_{p1}X_1 + \hat{a}_{p2}X_2 + \cdots + \hat{a}_{pp}X_p\end{aligned}$$

Generally, we only retain the first  $k$  principal component. There are a number of criteria that may be used to decide how many components should be retained:

1. To obtain the simplest possible interpretation, we want  $k$  to be as small as possible. If we can explain most of the variation just by **two** principal components then this would give us a much simpler description of the data.
2. Retain the first  $k$  components which explain a “large” proportion of the total variation, say 70 – 80%.
3. Examine a scree plot. This is a plot of the eigenvalues versus the component number. The idea is to look for the “elbow” which corresponds to the point after which the eigenvalues decrease more slowly. Adding components after this point explains relatively little more of the variance. See the next figure for an example of a scree plot.

Scree plot showing eigenvalue by number of principal component.

## 6.4 Standardization of the features

If we use the raw data, the principal component analysis will tend to give more emphasis to the variables that have higher variances than to those variables that have very low variances.

In effect the results of the analysis will depend on what units of measurement are used to measure each variable. That would imply that a principal component analysis should only be used with the raw data if all variables have the same units of measure. And even in this case, only if you wish to give those variables which have higher variances more weight in the analysis.



- The results of principal component analysis depend on the scales at which the variables are measured.
- Variables with the highest sample variances will tend to be emphasized in the first few principal components.
- Principal component analysis using the covariance function should only be considered if all of the variables have the same units of measurement.

If the variables either have different units of measurement, or if we wish each variable to receive equal weight in the analysis, then the variables should be **standardized (scaled)** before a principal components analysis is carried out. Standardize the variables by subtracting its mean from that variable and dividing it by its standard deviation:

$$Z_{ij} = \frac{X_{ij} - \bar{x}_j}{\sigma_j}$$

where

- $X_{ij}$  = Data for variable  $j$  in sample unit  $i$
- $\bar{x}_j$  = Sample mean for variable  $j$
- $\sigma_j$  = Sample standard deviation for variable  $j$

**Note:**  $Z_j$  has mean = 0 and variance = 1.



The variance-covariance matrix of the standardized data is equal to the correlation matrix for the unstandardized data. Therefore, principal component analysis using the standardized data is equivalent to principal component analysis using the correlation matrix. **Remark:** You are going to prove it in the practical work session.

## 6.5 Projection of the data

### Scores

Using the coefficients (loadings) of every principal component, we can project the observations on the axis of the principal component, those projections are called *scores*. For example, the scores of the first principal component are

$$\forall 1 \leq i \leq n \quad \hat{Y}_1^i = \hat{a}_{11}X_1^i + \hat{a}_{12}X_2^i + \cdots + \hat{a}_{1p}X_p^i$$

( $X_1^i$  is the value of feature 1 for the observation  $i$ )

This can be written for all observations and all the principal components using the matrix formulation

$$\hat{\mathbf{Y}} = \hat{\mathbf{A}}\mathbf{X}$$

where  $\hat{\mathbf{A}}$  is the matrix of the coefficients  $\hat{a}_{ij}$ .

### Visualization

Once we have computed the principal components, we can plot them against each other in order to produce low-dimensional views of the data.

We can plot the score vector  $Y_1$  against  $Y_2$ ,  $Y_1$  against  $Y_3$ ,  $Y_2$  against  $Y_3$ , and so forth. Geometrically, this amounts to projecting the original data down onto the subspace spanned by  $\mathbf{a}_1$ ,  $\mathbf{a}_2$ , and  $\mathbf{a}_3$ , and plotting the projected points.

To interpret the results obtained by PCA, we plot on the same figure both the principal component scores and the loading vectors. This figure is called a *biplot*. An example is given later in this chapter.

### Extra



- You can read this [tutorial](#). In the document, there is an introduction about the mathematical concepts used in PCA. Plus a detailed example of PCA.
- You can watch these videos for a nice explanation of PCA [1](#) [2](#).

## 6.6 Case study

### Employment in European countries in the late 70s

The purpose of this case study is to reveal the structure of the job market and economy in different developed countries. The final aim is to have a meaningful and rigorous plot that is able to show the most important features of the countries in a concise form.

The dataset `eurojob` contains the data employed in this case study. It contains the percentage of workforce employed in 1979 in 9 industries for 26 European countries. The industries measured are:

- Agriculture (**Agr**)
- Mining (**Min**)
- Manufacturing (**Man**)
- Power supply industries (**Pow**)
- Construction (**Con**)
- Service industries (**Ser**)
- Finance (**Fin**)
- Social and personal services (**Soc**)
- Transport and communications (**Tra**)

If the dataset is imported into R and the case names are set as **Country** (important in order to have only numerical variables), then the data should look like this:

Table 6.1: The ‘eurojob’ dataset.

Country	Agr	Min	Man	Pow	Con	Ser	Fin	Soc	Tra
Belgium	3.3	0.9	27.6	0.9	8.2	19.1	6.2	26.6	7.2
Denmark	9.2	0.1	21.8	0.6	8.3	14.6	6.5	32.2	7.1
France	10.8	0.8	27.5	0.9	8.9	16.8	6.0	22.6	5.7
WGerm	6.7	1.3	35.8	0.9	7.3	14.4	5.0	22.3	6.1
Ireland	23.2	1.0	20.7	1.3	7.5	16.8	2.8	20.8	6.1
Italy	15.9	0.6	27.6	0.5	10.0	18.1	1.6	20.1	5.7
Luxem	7.7	3.1	30.8	0.8	9.2	18.5	4.6	19.2	6.2
Nether	6.3	0.1	22.5	1.0	9.9	18.0	6.8	28.5	6.8
UK	2.7	1.4	30.2	1.4	6.9	16.9	5.7	28.3	6.4
Austria	12.7	1.1	30.2	1.4	9.0	16.8	4.9	16.8	7.0
Finland	13.0	0.4	25.9	1.3	7.4	14.7	5.5	24.3	7.6
Greece	41.4	0.6	17.6	0.6	8.1	11.5	2.4	11.0	6.7
Norway	9.0	0.5	22.4	0.8	8.6	16.9	4.7	27.6	9.4
Portugal	27.8	0.3	24.5	0.6	8.4	13.3	2.7	16.7	5.7
Spain	22.9	0.8	28.5	0.7	11.5	9.7	8.5	11.8	5.5
Sweden	6.1	0.4	25.9	0.8	7.2	14.4	6.0	32.4	6.8
Switz	7.7	0.2	37.8	0.8	9.5	17.5	5.3	15.4	5.7
Turkey	66.8	0.7	7.9	0.1	2.8	5.2	1.1	11.9	3.2
Bulgaria	23.6	1.9	32.3	0.6	7.9	8.0	0.7	18.2	6.7
Czech	16.5	2.9	35.5	1.2	8.7	9.2	0.9	17.9	7.0

EGerm	4.2	2.9	41.2	1.3	7.6	11.2	1.2	22.1	8.4
Hungary	21.7	3.1	29.6	1.9	8.2	9.4	0.9	17.2	8.0
Poland	31.1	2.5	25.7	0.9	8.4	7.5	0.9	16.1	6.9
Romania	34.7	2.1	30.1	0.6	8.7	5.9	1.3	11.7	5.0
USSR	23.7	1.4	25.8	0.6	9.2	6.1	0.5	23.6	9.3
Yugoslavia	48.7	1.5	16.8	1.1	4.9	6.4	11.3	5.3	4.0

**Note:** To set the case names as Country, we do

```
row.names(eurojob) <- eurojob$Country
eurojob$Country <- NULL
```

So far, we know how to compute summaries for *each variable*, and how to quantify and visualize relations between variables with the correlation matrix and the scatterplot matrix. But even for a moderate number of variables like this, their results are hard to process.

```
# Summary of the data - marginal
summary(eurojob)

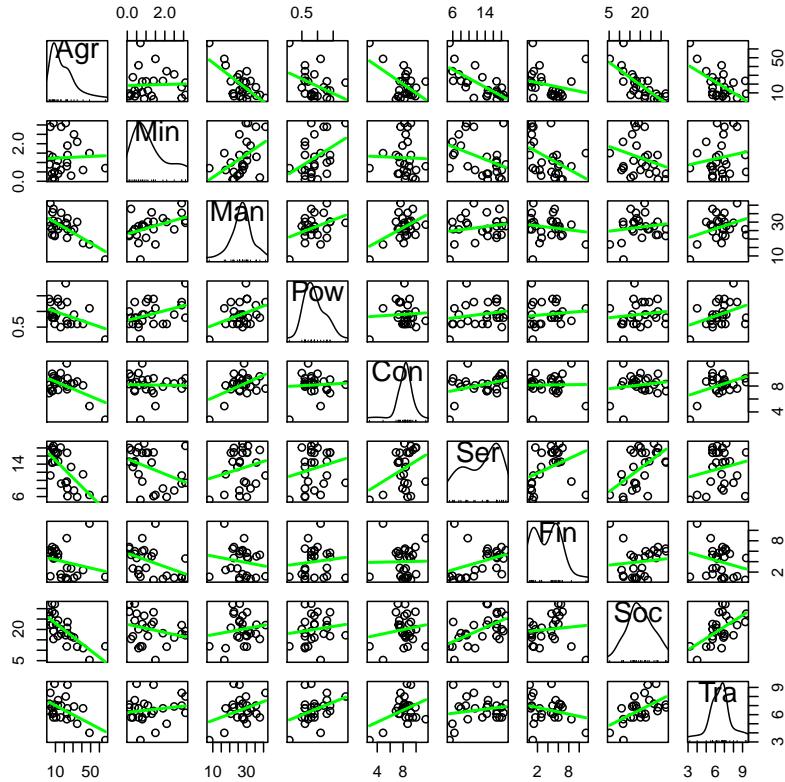
#ans>      Agr           Min          Man          Pow
#ans> Min.   : 2.7   Min.   :0.100   Min.   : 7.9   Min.   :0.100
#ans> 1st Qu.: 7.7   1st Qu.:0.525   1st Qu.:23.0   1st Qu.:0.600
#ans> Median :14.4   Median :0.950   Median :27.6   Median :0.850
#ans> Mean   :19.1   Mean   :1.254   Mean   :27.0   Mean   :0.908
#ans> 3rd Qu.:23.7   3rd Qu.:1.800   3rd Qu.:30.2   3rd Qu.:1.175
#ans> Max.   :66.8   Max.   :3.100   Max.   :41.2   Max.   :1.900
#ans>      Con           Ser          Fin          Soc
#ans> Min.   : 2.80   Min.   : 5.20   Min.   : 0.50   Min.   : 5.3
#ans> 1st Qu.: 7.53   1st Qu.: 9.25   1st Qu.: 1.23   1st Qu.:16.2
#ans> Median : 8.35   Median :14.40   Median : 4.65   Median :19.6
#ans> Mean   : 8.17   Mean   :12.96   Mean   : 4.00   Mean   :20.0
#ans> 3rd Qu.: 8.97   3rd Qu.:16.88   3rd Qu.: 5.92   3rd Qu.:24.1
#ans> Max.   :11.50   Max.   :19.10   Max.   :11.30   Max.   :32.4
#ans>      Tra
#ans> Min.   :3.20
#ans> 1st Qu.:5.70
#ans> Median :6.70
#ans> Mean   :6.55
#ans> 3rd Qu.:7.08
#ans> Max.   :9.40

# Correlation matrix
cor(eurojob)

#ans>      Agr      Min      Man      Pow      Con      Ser      Fin      Soc      Tra
#ans> Agr  1.0000  0.0358 -0.671  -0.4001 -0.5383 -0.737 -0.2198 -0.747 -0.565
#ans> Min  0.0358  1.0000  0.445   0.4055 -0.0256 -0.397 -0.4427 -0.281  0.157
#ans> Man -0.6711  0.4452  1.000   0.3853  0.4945  0.204  -0.1558  0.154  0.351
#ans> Pow -0.4001  0.4055  0.385   1.0000  0.0599  0.202   0.1099  0.132  0.375
#ans> Con -0.5383 -0.0256  0.494   0.0599  1.0000  0.356   0.0163  0.158  0.388
#ans> Ser -0.7370 -0.3966  0.204   0.2019  0.3560  1.000   0.3656  0.572  0.188
#ans> Fin -0.2198 -0.4427 -0.156   0.1099  0.0163  0.366   1.0000  0.108 -0.246
#ans> Soc -0.7468 -0.2810  0.154   0.1324  0.1582  0.572   0.1076  1.000  0.568
#ans> Tra -0.5649  0.1566  0.351   0.3752  0.3877  0.188  -0.2459  0.568  1.000

# Scatterplot matrix
require(car)
```

```
scatterplotMatrix(eurojob, regLine = list(method=lm, lty=1, lwd=2, col="green"), smooth = FALSE,
                 ellipse = FALSE, plot.points = T, col="black",
                 diagonal = TRUE)
```



We definitely need a way of visualizing and quantifying the relations between variables for a moderate to large amount of variables. PCA will be a handy way. Recall what PCA does:

1. Takes the data for the variables  $X_1, \dots, X_p$ .
2. Using this data, looks for new variables  $\text{PC}_1, \dots, \text{PC}_p$  such that:
  - $\text{PC}_j$  is a **linear combination** of  $X_1, \dots, X_k$ ,  $1 \leq j \leq p$ . This is,  $\text{PC}_j = a_{1j}X_1 + a_{2j}X_2 + \dots + a_{pj}X_p$ .
  - $\text{PC}_1, \dots, \text{PC}_p$  are **sorted decreasingly in terms of variance**. Hence  $\text{PC}_j$  has more variance than  $\text{PC}_{j+1}$ ,  $1 \leq j \leq p-1$ ,
  - $\text{PC}_{j_1}$  and  $\text{PC}_{j_2}$  are **uncorrelated**, for  $j_1 \neq j_2$ .
  - $\text{PC}_1, \dots, \text{PC}_p$  have the **same information**, measured in terms of **total variance**, as  $X_1, \dots, X_p$ .
3. Produces three key objects:
  - **Variances of the PCs.** They are sorted decreasingly and give an idea of which PCs contain most of the information of the data (the ones with more variance).
  - **Weights of the variables in the PCs.** They give the interpretation of the PCs in terms of the original variables, as they are the coefficients of the linear combination. The weights of the variables  $X_1, \dots, X_p$  on the PC $_j$ ,  $a_{1j}, \dots, a_{pj}$ , are normalized:  $a_{1j}^2 + \dots + a_{pj}^2 = 1$ ,  $j = 1, \dots, p$ . In R, they are called **loadings**.
  - **Scores of the data in the PCs:** this is the data with  $\text{PC}_1, \dots, \text{PC}_p$  variables instead of  $X_1, \dots, X_p$ . The **scores are uncorrelated**. Useful for knowing which PCs have more effect on a certain observation.

Hence, PCA rearranges our variables in an information-equivalent, but more convenient, layout where the

variables are sorted according to the amount of information they are able to explain. From this position, the next step is clear: **stick only with a limited number of PCs such that they explain most of the information** (e.g., 70% of the total variance) and do *dimension reduction*. The effectiveness of PCA in practice varies from the structure present in the dataset. For example, in the case of highly dependent data, it could explain more than the 90% of variability of a dataset with tens of variables with just two PCs.

Let's see how to compute a full PCA in R.

```
# The main function - use cor = TRUE to avoid scale distortions
pca <- princomp(eurojob, cor = TRUE)

# What is inside?
str(pca)
#ans> List of 7
#ans> $ sdev    : Named num [1:9] 1.867 1.46 1.048 0.997 0.737 ...
#ans> ..- attr(*, "names")= chr [1:9] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...
#ans> $ loadings: 'loadings' num [1:9, 1:9] 0.52379 0.00132 -0.3475 -0.25572 -0.32518 ...
#ans> ..- attr(*, "dimnames")=List of 2
#ans> ...$ : chr [1:9] "Agr" "Min" "Man" "Pow" ...
#ans> ...$ : chr [1:9] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...
#ans> $ center   : Named num [1:9] 19.131 1.254 27.008 0.908 8.165 ...
#ans> ..- attr(*, "names")= chr [1:9] "Agr" "Min" "Man" "Pow" ...
#ans> $ scale    : Named num [1:9] 15.245 0.951 6.872 0.369 1.614 ...
#ans> ..- attr(*, "names")= chr [1:9] "Agr" "Min" "Man" "Pow" ...
#ans> $ n.obs    : int 26
#ans> $ scores   : num [1:26, 1:9] -1.71 -0.953 -0.755 -0.853 0.104 ...
#ans> ..- attr(*, "dimnames")=List of 2
#ans> ...$ : chr [1:26] "Belgium" "Denmark" "France" "WGerm" ...
#ans> ...$ : chr [1:9] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...
#ans> $ call     : language princomp(x = eurojob, cor = TRUE)
#ans> - attr(*, "class")= chr "princomp"

# The standard deviation of each PC
pca$sdev
#ans> Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
#ans> 1.86739 1.45951 1.04831 0.99724 0.73703 0.61922 0.47514 0.36985 0.00675

# Weights: the expression of the original variables in the PCs
# E.g. Agr = -0.524 * PC1 + 0.213 * PC5 - 0.152 * PC6 + 0.806 * PC9
# And also: PC1 = -0.524 * Agr + 0.347 * Man + 0.256 * Pow + 0.325 * Con + ...
# (Because the matrix is orthogonal, so the transpose is the inverse)
pca$loadings
#ans>
#ans> Loadings:
#ans>      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
#ans> Agr      0.524          0.213  0.153          0.806
#ans> Min      0.618 -0.201        -0.164 -0.101 -0.726
#ans> Man     -0.347  0.355 -0.150 -0.346 -0.385 -0.288  0.479  0.126  0.366
#ans> Pow     -0.256  0.261 -0.561  0.393  0.295  0.357  0.256 -0.341
#ans> Con     -0.325          0.153 -0.668  0.472  0.130 -0.221 -0.356
#ans> Ser     -0.379 -0.350 -0.115          -0.284  0.615 -0.229  0.388  0.238
#ans> Fin      -0.454 -0.587          0.280 -0.526 -0.187  0.174  0.145
#ans> Soc     -0.387 -0.222  0.312  0.412 -0.220 -0.263 -0.191 -0.506  0.351
#ans> Tra     -0.367  0.203  0.375  0.314  0.513 -0.124          0.545
```

```
#ans>
#ans>           Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
#ans> SS loadings    1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
#ans> Proportion Var 0.111  0.111  0.111  0.111  0.111  0.111  0.111  0.111
#ans> Cumulative Var 0.111  0.222  0.333  0.444  0.556  0.667  0.778  0.889
#ans>           Comp.9
#ans> SS loadings    1.000
#ans> Proportion Var 0.111
#ans> Cumulative Var 1.000

# Scores of the data on the PCs: how is the data reexpressed into PCs
head(pca$scores, 10)
#ans>           Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
#ans> Belgium -1.710 -1.2218 -0.1148  0.3395 -0.3245 -0.0473 -0.3401  0.403
#ans> Denmark -0.953 -2.1278  0.9507  0.5939  0.1027 -0.8273 -0.3029 -0.352
#ans> France  -0.755 -1.1212 -0.4980 -0.5003 -0.2997  0.1158 -0.1855 -0.266
#ans> WGerm   -0.853 -0.0114 -0.5795 -0.1105 -1.1652 -0.6181  0.4446  0.194
#ans> Ireland  0.104 -0.4140 -0.3840  0.9267  0.0152  1.4242 -0.0370 -0.334
#ans> Italy    -0.375 -0.7695  1.0606 -1.4772 -0.6452  1.0021 -0.1418 -0.130
#ans> Luxem   -1.059  0.7558 -0.6515 -0.8352 -0.8659  0.2188 -1.6942  0.547
#ans> Nether   -1.688 -2.0048  0.0637 -0.0235  0.6352  0.2120 -0.3034 -0.591
#ans> UK       -1.630 -0.3731 -1.1409  1.2669 -0.8129 -0.0361  0.0413 -0.349
#ans> Austria  -1.176  0.1431 -1.0434 -0.1577  0.5210  0.8019  0.4150  0.215
#ans>           Comp.9
#ans> Belgium -0.001090
#ans> Denmark  0.015619
#ans> France   -0.000507
#ans> WGerm   -0.006539
#ans> Ireland  0.010879
#ans> Italy    0.005602
#ans> Luxem   0.003453
#ans> Nether   -0.010931
#ans> UK      -0.005478
#ans> Austria -0.002816

# Scatterplot matrix of the scores - See how they are uncorrelated!
scatterplotMatrix(pca$scores, regLine = list(method=lm, lty=1, lwd=2, col="green"), smooth = FALSE,
                  ellipse = FALSE, plot.points = T, col="black",
                  diagonal = TRUE)
```



PCA produces **uncorrelated** variables from the original set  $X_1, \dots, X_p$ . This implies that:

- The PCs are uncorrelated, **but not independent** (uncorrelated does not imply independent).
- An uncorrelated or independent variable in  $X_1, \dots, X_p$  will get a PC only associated to it. In the extreme case where all the  $X_1, \dots, X_p$  are uncorrelated, these coincide with the PCs (up to sign flips).

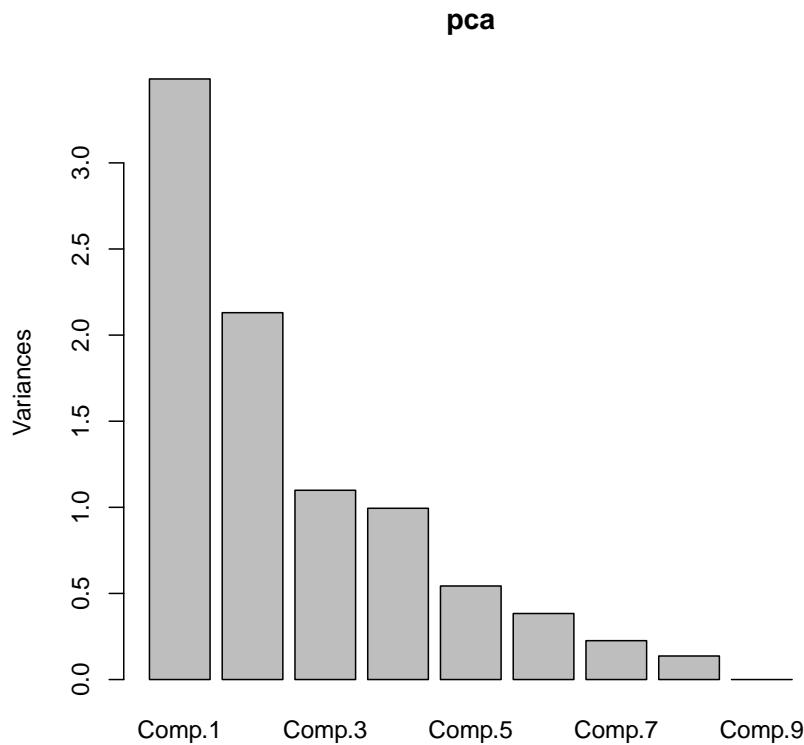
```
# Means of the variables - before PCA the variables are centered
pca$center
#ans>   Agr     Min     Man     Pow     Con     Ser     Fin     Soc     Tra
#ans> 19.131  1.254 27.008  0.908  8.165 12.958  4.000 20.023  6.546

# Rescaling done to each variable
# - if cor = FALSE (default), a vector of ones
# - if cor = TRUE, a vector with the standard deviations of the variables
pca$scale
#ans>   Agr     Min     Man     Pow     Con     Ser     Fin     Soc     Tra
#ans> 15.245  0.951  6.872  0.369  1.614  4.486  2.752  6.697  1.364

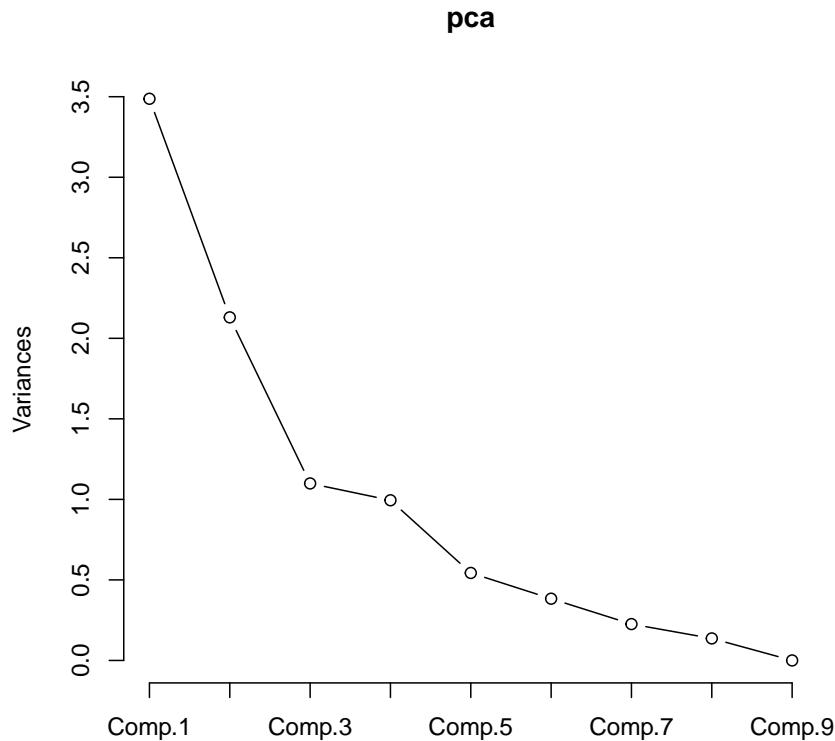
# Summary of the importance of components - the third row is key
summary(pca)
#ans> Importance of components:
#ans>                               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7
#ans> Standard deviation           1.867  1.460  1.048  0.997  0.7370  0.6192  0.4751
```

```
#ans> Proportion of Variance  0.387  0.237  0.122  0.110 0.0604 0.0426 0.0251
#ans> Cumulative Proportion   0.387  0.624  0.746  0.857 0.9171 0.9597 0.9848
#ans>                                         Comp.8   Comp.9
#ans> Standard deviation      0.3699 6.75e-03
#ans> Proportion of Variance  0.0152 5.07e-06
#ans> Cumulative Proportion   1.0000 1.00e+00

# Scree plot - the variance of each component
plot(pca)
```



```
# With connected lines - useful for looking for the "elbow"
plot(pca, type = "l")
```



```
# PC1 and PC2 -- These are the weights of the variables on the first two Principal Components
pca$loadings[, 1:2]
#ans>          Comp. 1   Comp. 2
#ans> Agr    0.52379  0.0536
#ans> Min    0.00132  0.6178
#ans> Man   -0.34750  0.3551
#ans> Pow   -0.25572  0.2611
#ans> Con   -0.32518  0.0513
#ans> Ser   -0.37892 -0.3502
#ans> Fin   -0.07437 -0.4537
#ans> Soc   -0.38741 -0.2215
#ans> Tra   -0.36682  0.2026
```

Based on the weights of the variables on the PCs shown above, we can extract the following interpretation:

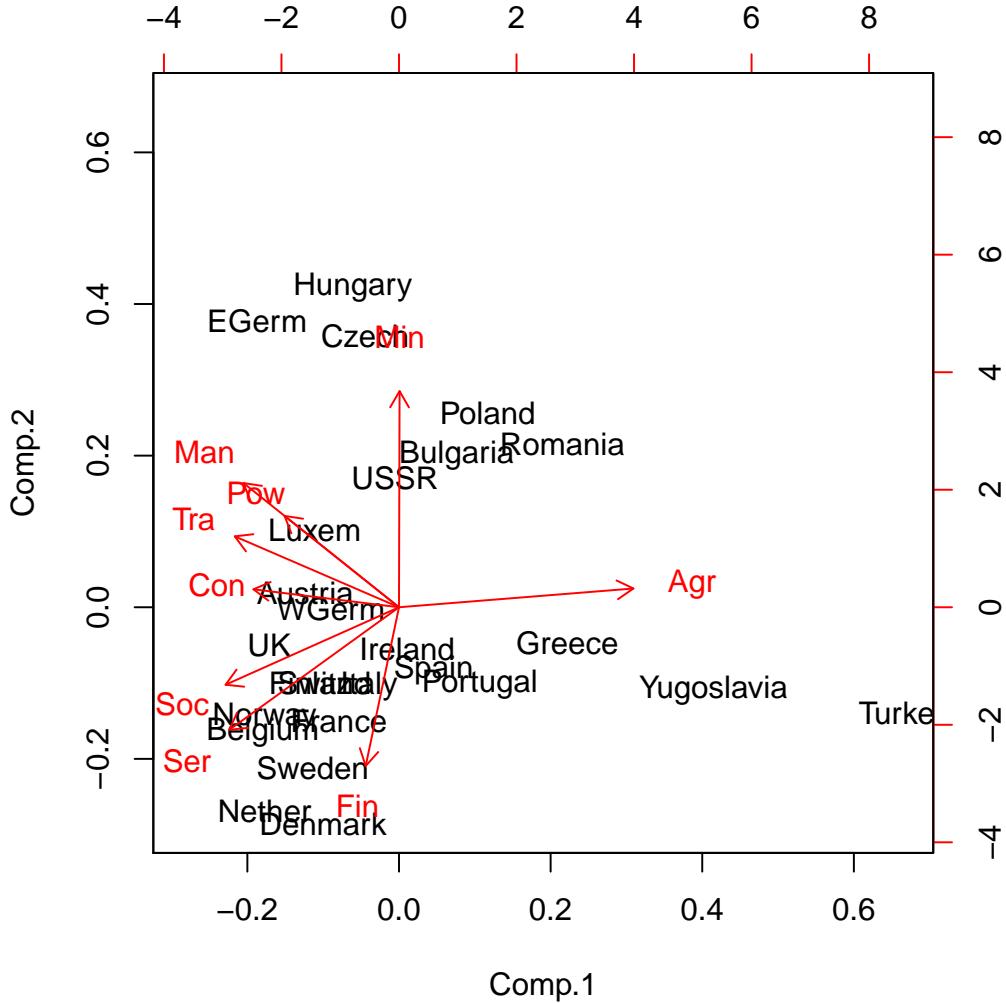
- PC1 is roughly a linear combination of `Agr`, with *negative* weight, and (`Man`, `Pow`, `Con`, `Ser`, `Soc`, `Tra`), with *positive* weights. So it can be interpreted as an *indicator* of the kind of economy of the country: agricultural (negative values) or industrial (positive values).
- PC2 has *negative* weights on (`Min`, `Man`, `Pow`, `Tra`) and *positive* weights in (`Ser`, `Fin`, `Soc`). It can be interpreted as the contrast between relatively large or small service sectors. So it tends to be negative in communist countries and positive in capitalist countries.



The interpretation of the PCs involves inspecting the weights and interpreting the linear combination of the original variables, which might be separating between two clear characteristics of the data

To conclude, let's see how we can represent our original data into a plot called *biplot* that summarizes all the analysis for two PCs.

```
# Biplot - plot together the scores for PC1 and PC2 and the  
# variables expressed in terms of PC1 and PC2  
biplot(pca)
```



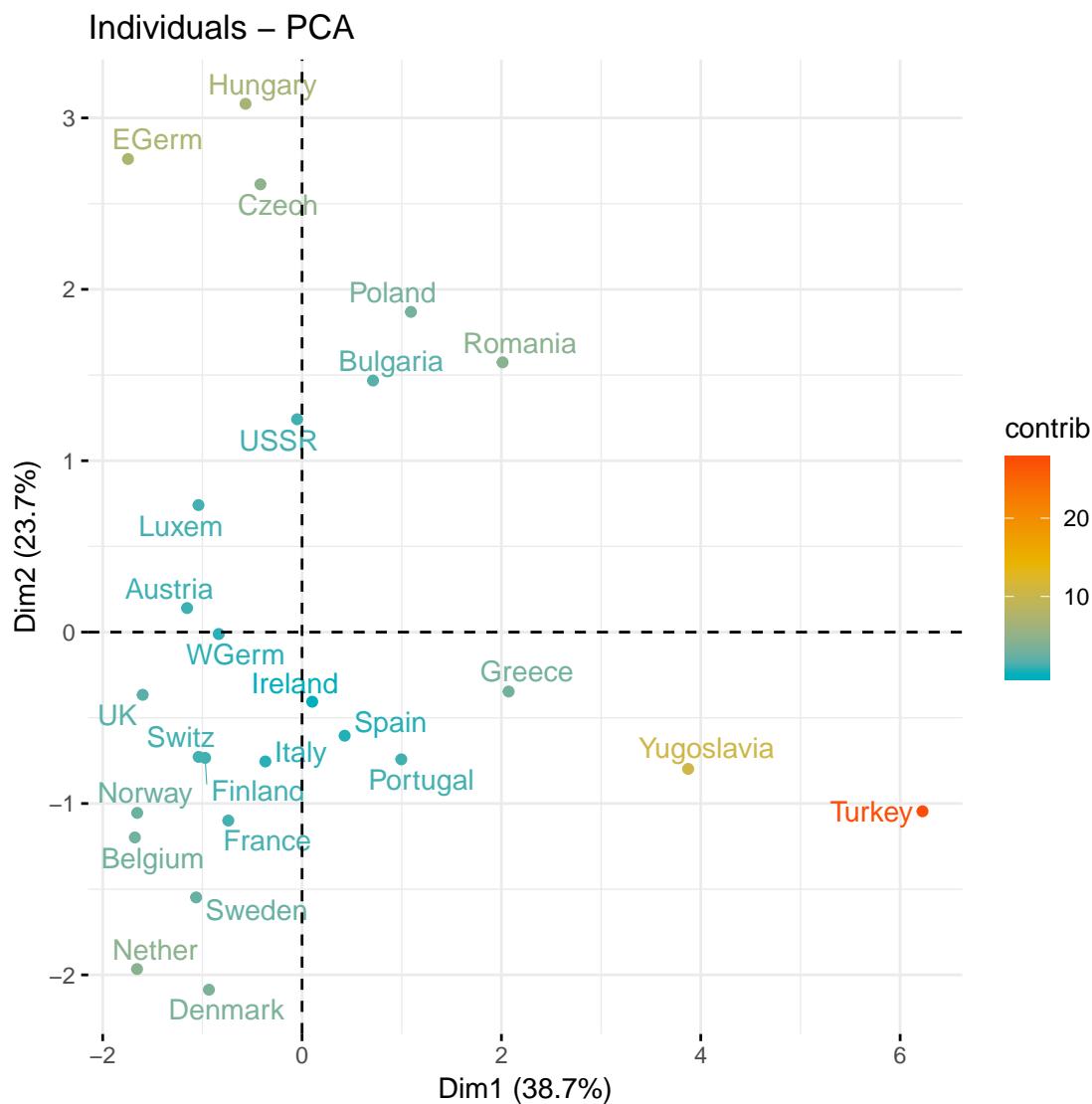
This biplot confirms the above interpretation.

Recall that the biplot is a superposition of two figures: the first one is the scatterplot of the principal component scores (the projections of the observations on the new low dimensional space - also known as graph of individuals) and the second one is the figure of loading vectors (also known as graph of variables). In fact, the graph of variables is a correlation circle, the variables whose unit vectors are close to each other are said to be positively correlated, meaning that their influence on the positioning of individuals is similar (again, these proximities are reflected in the projections of variables on the graph of individuals). However, variables far away from each other will be defined as being negatively correlated. Variables that have perpendicular unit vector are uncorrelated. To see both graphs we can use the `factoextra` package like follows:

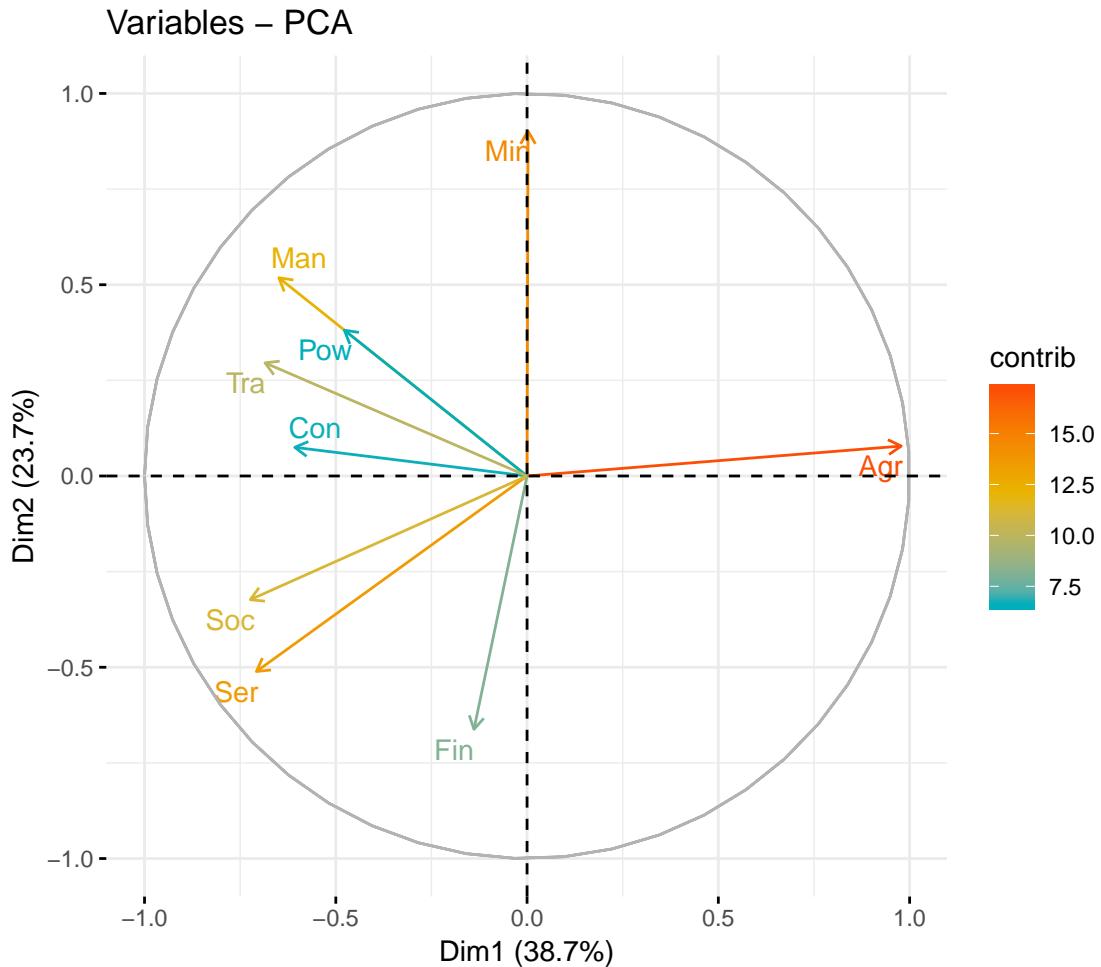
```
# loading factoextra library
require(factoextra)

# Calculating pca using prcomp(), which is a built-in function in R
res.pca <- prcomp(eurojob, scale = TRUE)

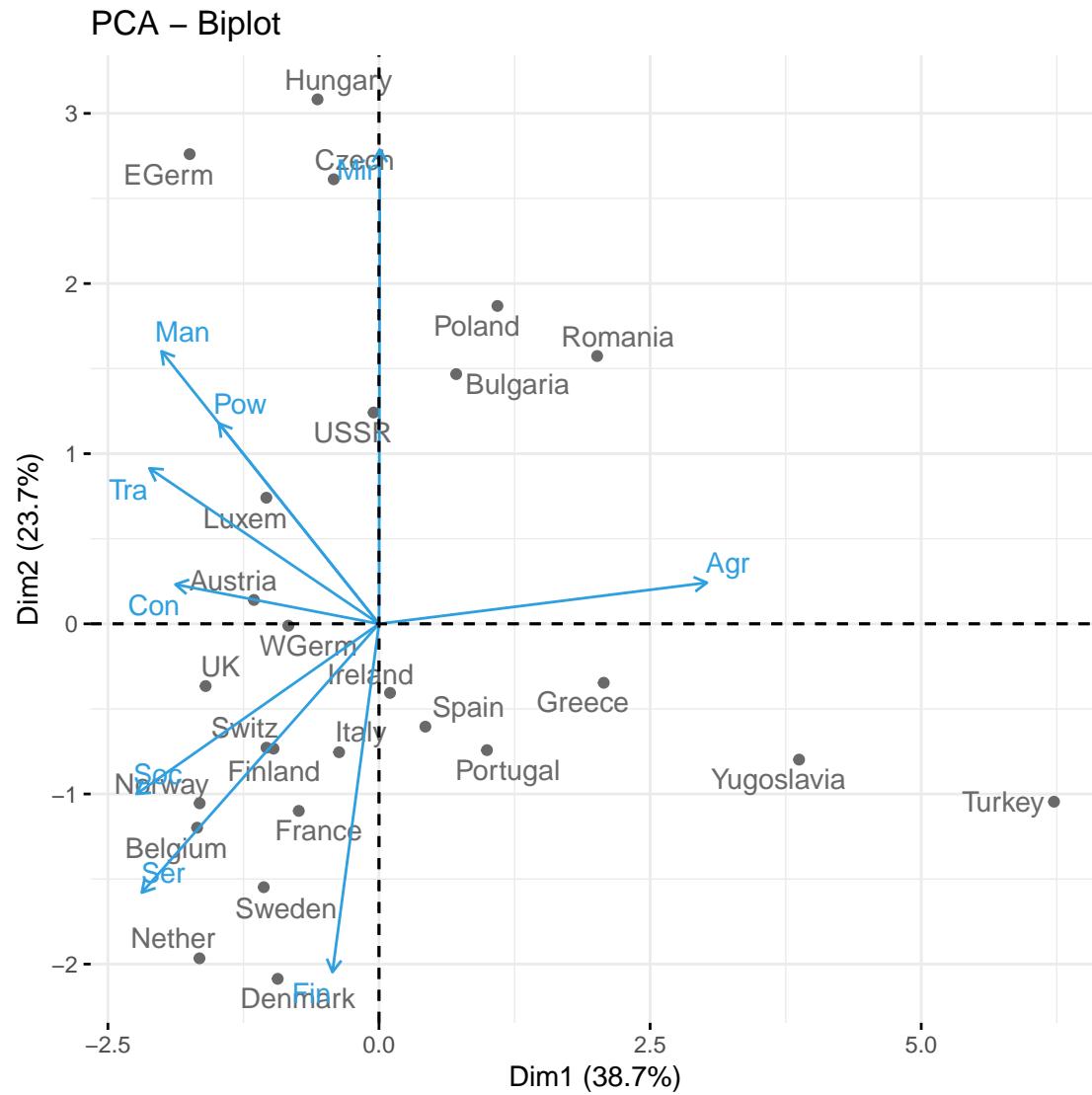
# Graph of individuals
fviz_pca_ind(res.pca,
             col.ind = "contrib", # Color by their contribution to axes
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE
)
```



```
# Graph of variables
fviz_pca_var(res.pca,
             col.var = "contrib",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE
)
```



```
# Biplot of individuals and variables
fviz_pca_biplot(res.pca, repel = TRUE,
                 col.var = "#2E9FDF",
                 col.ind = "#696969"
               )
```





# PW 6

## The Iris Dataset

The *Iris flower dataset* or *Fisher's Iris dataset* is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his **1936** paper (FISHER, 1936).

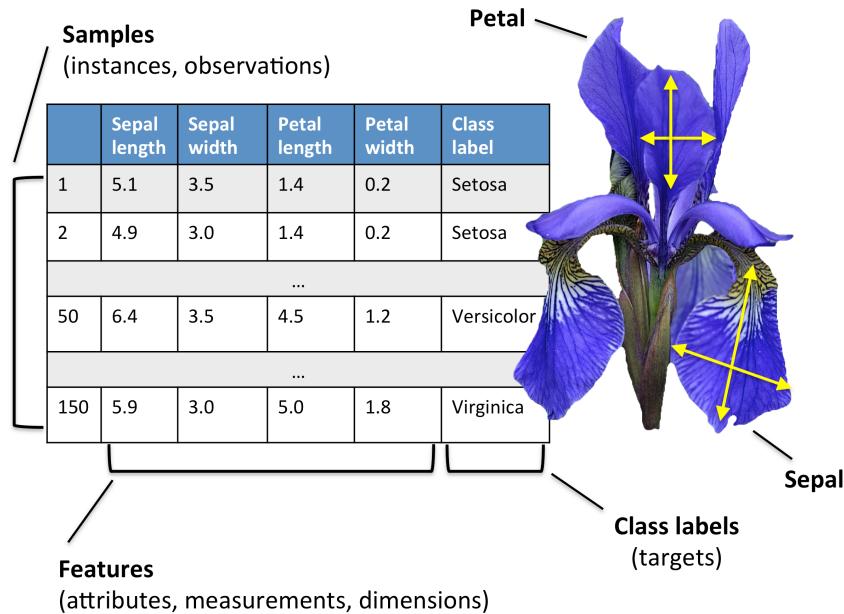
The data set consists of 50 samples from each of three species of Iris. Four features were measured from each sample.

The three species in the Iris dataset are:

1. *Iris-setosa* ( $n_1 = 50$ )
2. *Iris-versicolor* ( $n_2 = 50$ )
3. *Iris-virginica* ( $n_3 = 50$ )

And the four features in Iris dataset are:

1. *sepal length* in cm
2. *sepal width* in cm
3. *petal length* in cm
4. *petal width* in cm



## Loading Data

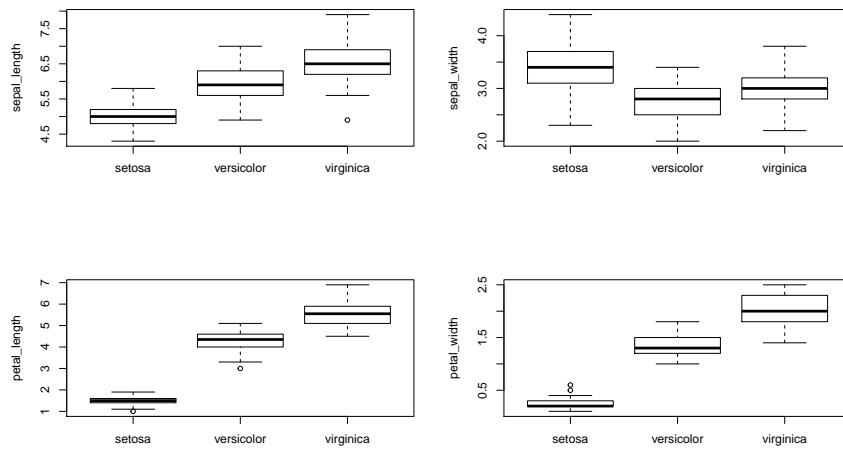
1. Download the iris dataset from [here](#) and import it into R.

## Exploratory analysis

2. Compare the means and the quartiles of the 3 different flower classes for the 4 different features (Plot 4 boxplots into the same figure).



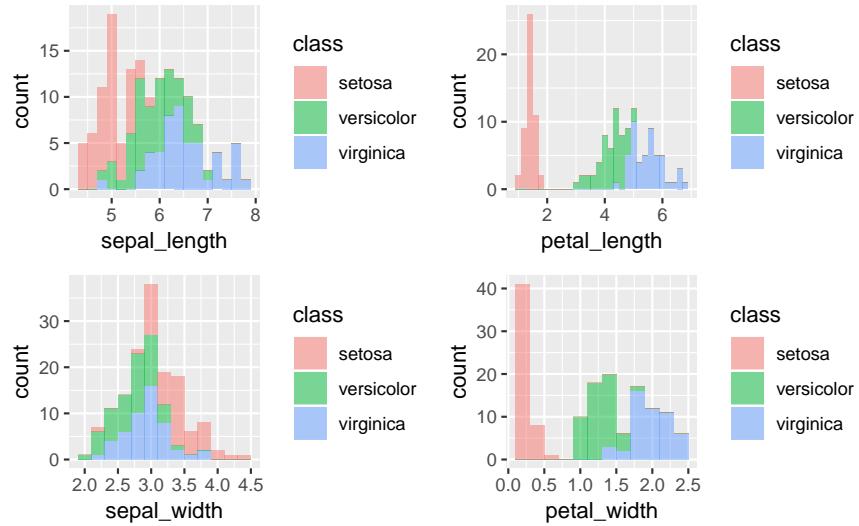
**Hint:** you can use `par(mfrow=c(2,2))` to show multiple plots on the same figure ( $2 \times 2$  plots here). Like follows:



3. To explore how the 3 different flower classes are distributed along the 4 different features, visualize them via histograms using the following code.

```
# Let's use the ggplot2 library
# ggplot2 is the most advanced package for data visualization
# gg corresponds to The Grammar of Graphics.
library(ggplot2) #of course you must install it first if you don't have it already

# histogram of sepal_length
ggplot(iris, aes(x=sepal_length, fill=class)) +
  geom_histogram(binwidth=.2, alpha=.5)
# histogram of sepal_width
ggplot(iris, aes(x=sepal_width, fill=class)) +
  geom_histogram(binwidth=.2, alpha=.5)
# histogram of petal_length
ggplot(iris, aes(x=petal_length, fill=class)) +
  geom_histogram(binwidth=.2, alpha=.5)
# histogram of petal_width
ggplot(iris, aes(x=petal_width, fill=class)) +
  geom_histogram(binwidth=.2, alpha=.5)
```



## PCA using `princomp()`

“



`princomp()` and `prcomp()` are built-in R functions. They both perform PCA.

4. Apply a PCA on the Iris dataset using the `princomp()`.

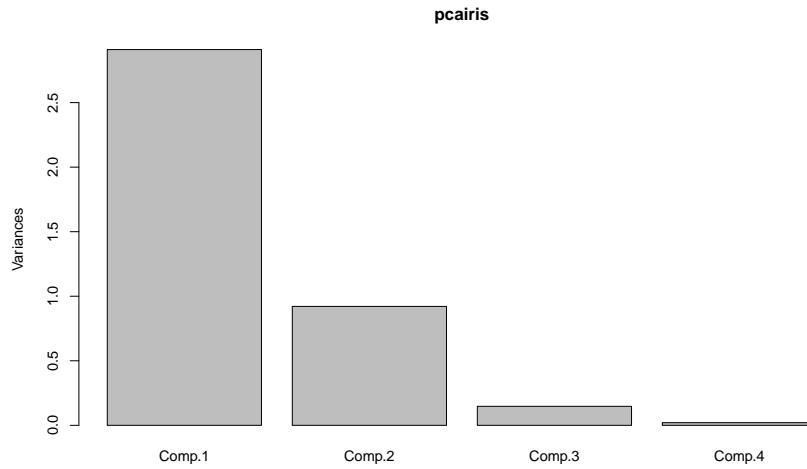
```
pcairis=princomp(iris[,-5], cor=T)
# Note that we take only the numerical columns to apply PCA.
# now pcairis is a R object of type princomp

# To display the internal structure of pcairis
str(pcairis)
#ans> List of 7
#ans> $ sdev   : Named num [1:4] 1.706 0.96 0.384 0.144
#ans> ..- attr(*, "names")= chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
#ans> $ loadings: 'loadings' num [1:4, 1:4] 0.522 -0.263 0.581 0.566 0.372 ...
#ans> ..- attr(*, "dimnames")=List of 2
#ans> ...$ : chr [1:4] "sepal_length" "sepal_width" "petal_length" "petal_width"
#ans> ...$ : chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
#ans> $ center  : Named num [1:4] 5.84 3.05 3.76 1.2
#ans> ..- attr(*, "names")= chr [1:4] "sepal_length" "sepal_width" "petal_length" "petal_width"
#ans> $ scale   : Named num [1:4] 0.825 0.432 1.759 0.761
#ans> ..- attr(*, "names")= chr [1:4] "sepal_length" "sepal_width" "petal_length" "petal_width"
#ans> $ n.obs   : int 150
#ans> $ scores  : num [1:150, 1:4] -2.26 -2.09 -2.37 -2.3 -2.39 ...
#ans> ..- attr(*, "dimnames")=List of 2
#ans> ...$ : NULL
#ans> ...$ : chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
#ans> $ call    : language princomp(x = iris[, -5], cor = T)
#ans> - attr(*, "class")= chr "princomp"

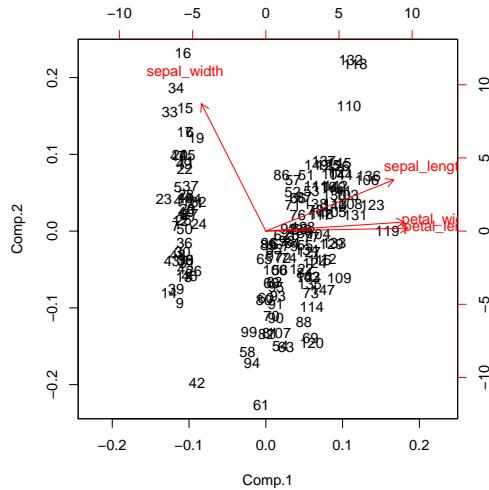
# To see the variance explained by the the pcs
summary(pcairis)
```

```
#ans> Importance of components:
#ans>                                         Comp.1 Comp.2 Comp.3 Comp.4
#ans> Standard deviation      1.706  0.960 0.3839 0.14355
#ans> Proportion of Variance 0.728  0.230 0.0368 0.00515
#ans> Cumulative Proportion  0.728  0.958 0.9948 1.00000

# To plot the variance explained by each pc
plot(pcairis)
```



```
# To plot together the scores for PC1 and PC2 and the
# variables expressed in terms of PC1 and PC2.
biplot(pcairis)
```



## Deeper PCA using factoextra package



To help in the interpretation and in the visualization of PCA we are going to use the package named **factoextra**.

No matter which function or package you decide to use for computing principal component methods, the **factoextra** R package can help to extract easily, in a human readable data format, the analysis results from the different functions mentioned above. **factoextra** provides also convenient solutions to create **ggplot2**-based beautiful graphs.

You can take a look at this link for a detailed example.

### 5. Using **factoextra** packag plot the following:

- The scree plot.
- The graph of individuals.
- The graph of variables.
- The biplot graph.
- The contributions of the variables to the first 2 principal components.

## Step-by-step PCA

In order to understand how PCA works, let's implement it step-by-step.



### Summary of the PCA Approach:

- Standardize the data.
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix or correlation matrix.
- Sort eigenvalues in descending order and choose the  $k$  eigenvectors that correspond to the  $k$  largest eigenvalues, where  $k$  is the number of dimensions of the new feature subspace ( $k \leq p$ ).
- Construct the projection matrix  $\mathbf{A}$  from the selected  $k$  eigenvectors.
- Transform the original dataset  $X$  via  $\mathbf{A}$  to obtain a  $k$ -dimensional feature subspace  $\mathbf{Y}$ .

### 6. First step, split the iris dataset into data $X$ and class labels $y$ .

```
x <- iris[,-5]
y <- iris[,5]
```



The iris dataset is now stored in form of a  $150 \times 4$  matrix where the columns are the different features, and every row represents a separate flower sample. Each sample row  $X^i$  can be pictured as a 4-dimensional vector

$$(X^i)^T = \begin{pmatrix} X_1^i \\ X_2^i \\ X_3^i \\ X_4^i \end{pmatrix} = \begin{pmatrix} \text{sepal length} \\ \text{sepal width} \\ \text{petal length} \\ \text{petal width} \end{pmatrix}$$



### Eigendecomposition - Computing Eigenvectors and Eigenvalues

The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the “core” of a

PCA: The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes.

### Standardizing

7. Scale the 4 features. Store the scaled matrix into a new one (for example, name it `X_scaled`).

### Covariance Matrix

8. The classic approach to PCA is to perform the eigendecomposition on the covariance matrix  $\Sigma$ , which is a  $p \times p$  matrix where each element represents the covariance between two features. Compute the Covariance Matrix of the scaled features (Print the results).



We can summarize the calculation of the covariance matrix via the following matrix equation:

$$\Sigma = \frac{1}{n-1} ((\mathbf{X} - \bar{\mathbf{X}})^T (\mathbf{X} - \bar{\mathbf{X}}))$$

where  $\bar{\mathbf{X}}$  is the mean vector  $\bar{\mathbf{X}} = \frac{1}{n} \sum_{k=1}^n x_k$ .

The mean vector is a  $p$ -dimensional vector where each value in this vector represents the sample mean of a feature column in the dataset.

9. Perform an eigendecomposition on the covariance matrix. Compute the Eigenvectors and the Eigenvalues (you can use the `eigen()` function). What do you obtain?

### Correlation Matrix



Especially, in the field of “Finance”, the correlation matrix typically used instead of the covariance matrix. However, the eigendecomposition of the covariance matrix (if the input data was standardized) yields the same results as a eigendecomposition on the correlation matrix, since the correlation matrix can be understood as the normalized covariance matrix.

10. Perform an eigendecomposition of the standardized data based on the correlation matrix.

11. Perform an eigendecomposition of the raw data based on the correlation matrix. Compare the obtained results with the previous question.



We should see that all three approaches yield the same eigenvectors and eigenvalue pairs:

- Eigendecomposition of the covariance matrix after standardizing the data.
- Eigendecomposition of the correlation matrix.
- Eigendecomposition of the correlation matrix after standardizing the data.

### Selecting Principal Components



The `eigen()` function will, by default, sort the eigenvalues in decreasing order.

### Explained Variance

12. Calculate the individual explained variation and the cumulative explained variation of each principal component. Show the results.

- 13.** Plot the individual explained variation. (scree plot)

### Projection Matrix

- 14.** Construct the projection matrix that will be used to transform the Iris data onto the new feature subspace.



The “projection matrix” is basically just a matrix of our concatenated top  $k$  eigenvectors. Here, the projection matrix  $\mathbf{A}$  is a  $4 \times 2$ -dimensional matrix.

### Projection Onto the New Feature Space

In this last step we will use the  $4 \times 2$ -dimensional projection matrix  $\mathbf{A}$  to transform our samples (observations) onto the new subspace via the equation  $\mathbf{Y} = \mathbf{X} \times \mathbf{A}$  where  $\mathbf{Y}$  is a  $150 \times 2$  matrix of our transformed samples.

- 15.** Compute  $\mathbf{Y}$  (Recall the  $\mathbf{Y}$  is the matrix of scores,  $\mathbf{A}$  is the matrix of loadings).

### Visualization

- 16.** Plot the observations on the new feature space. Name the axis PC1 and PC2.

- 17.** On the same plot, color the observations (the flowers) with respect to their flower classes.

## Part III

# Unsupervised Learning



# Chapter 7

## Clustering

### 7.1 Unsupervised Learning

Previously we considered *supervised* learning methods such as regression and classification, where we typically have access to a set of  $p$  features  $X_1, X_2, \dots, X_p$ , measured on  $n$  observations, and a response  $Y$  also measured on those same  $n$  observations (what we call **labels**). The goal was then to predict  $Y$  using  $X_1, X_2, \dots, X_p$ . From now on we will instead focus on **unsupervised** learning, a set of statistical tools where we have only a set of features  $X_1, X_2, \dots, X_p$  measured on  $n$  observations. We are not interested in prediction, because we do not have an associated response variable  $Y$ . Rather, the goal is to discover interesting things about the measurements on  $X_1, X_2, \dots, X_p$ . Is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations? Unsupervised learning refers to a diverse set of techniques for answering questions such as these. In this chapter, we will focus on a particular type of unsupervised learning: Principal Components Analysis (PCA), a tool used for *data visualization* or *data pre-processing* before supervised techniques are applied. In the next chapters, we will talk about clustering, another particular type of unsupervised learning. Clustering is a broad class of methods for discovering unknown subgroups in data.

Unsupervised learning is often much more challenging than supervised learning. The exercise tends to be more subjective, and there is no simple goal for the analysis, such as prediction of a response. Unsupervised learning is often performed as part of an *exploratory data analysis*. It is hard to assess the results obtained from unsupervised learning methods. If we fit a predictive model using a supervised learning technique, then it is possible to check our work by seeing how well our model predicts the response  $Y$  on observations not used in fitting the model. But in unsupervised learning, there is no way to check our work because we don't know the true answer: the problem is *unsupervised*.

### 7.2 Clustering

**Clustering** (or Cluster analysis) is the collection of techniques designed to find subgroups or *clusters* in a dataset of variables  $X_1, \dots, X_p$ . Depending on the similarities between the observations, these are partitioned in homogeneous groups as separated as possible between them. Clustering methods can be classified into these main categories:

- **Partition methods:** Given a fixed number of cluster  $k$ , these methods aim to assign each observation of  $X_1, \dots, X_p$  to a unique cluster, in such a way that the *within-cluster variation* is as small as possible (the clusters are as homogeneous as possible) while the *between cluster variation* is as large as possible (the clusters are as separated as possible).

- **Distribution models:** These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Poisson, etc..). A popular example of these models is Expectation-maximization algorithm using multivariate Normal distributions.
- **Hierarchical methods:** These methods construct a hierarchy for the observations in terms of their similitudes. This results in a tree-based representation of the data in terms of a *dendrogram*, which depicts how the observations are clustered at different levels – from the smallest groups of one element to the largest representing the whole dataset.
- **Density Models:** These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are DBSCAN and OPTICS.

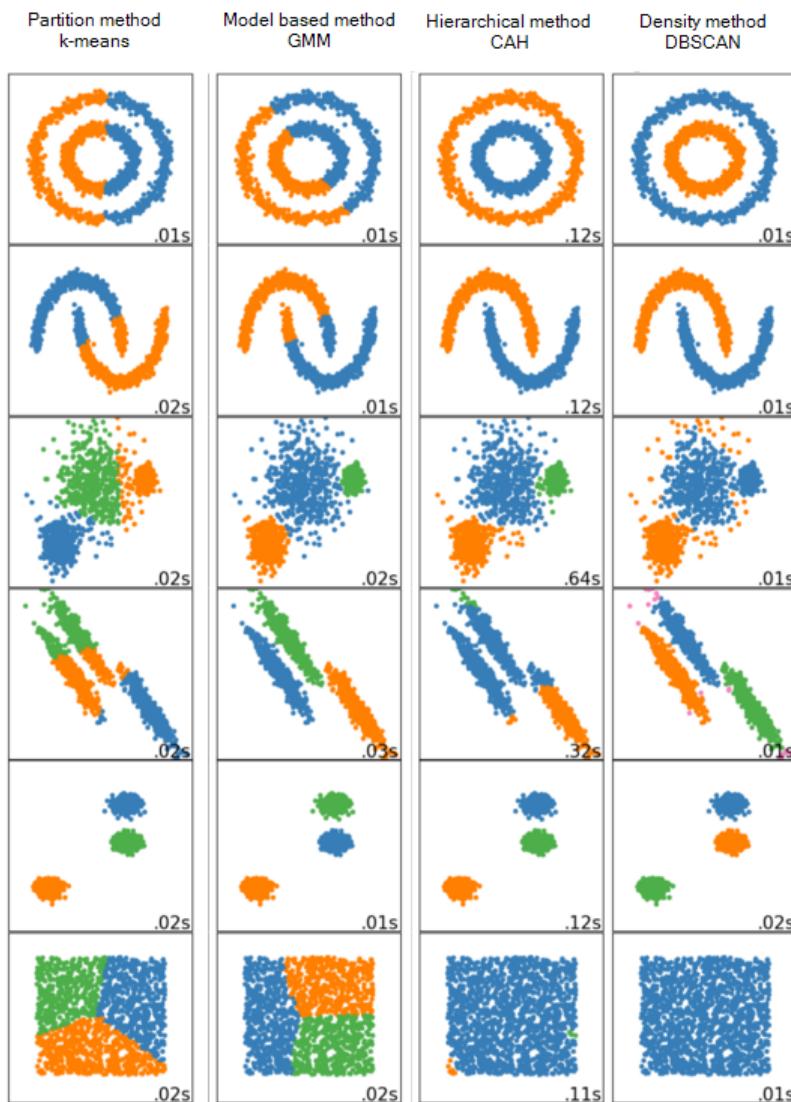


Figure 7.1: Performance comparison of different clustering methods on different datasets

In this chapter we will see the basics of the **partition methods**, and one of the most well-known clustering techniques, namely *k-means clustering*.

## 7.3 Introduction

**Clustering** (or Cluster analysis) is the process of partitioning a set of data objects (observations) into subsets. Each subset is a **cluster**, such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters.

The set of clusters resulting from a cluster analysis can be referred to as a clustering. In this context, different clustering methods may generate different clusterings on the same data set. The partitioning is not performed by humans, but by the clustering algorithm. Hence, clustering is useful in that it can lead to the discovery of previously unknown groups within the data.



Different clustering methods may generate different clusterings on the same data set.

Example: Imagine a Director of Customer Relationships at an Electronics magazine, and he has five managers working for him. He would like to organize all the company's customers into five groups so that each group can be assigned to a different manager. Strategically, he would like that the customers in each group are as similar as possible. Moreover, two given customers having very different business patterns should not be placed in the same group. His intention behind this business strategy is to develop customer relationship campaigns that specifically target each group, based on common features shared by the customers per group. Unlike in classification, the class **label** of each customer is unknown. He needs to discover these groupings. Given a large number of customers and many attributes describing customer profiles, it can be very costly or even infeasible to have a human study the data and manually come up with a way to partition the customers into strategic groups. He needs a *clustering* tool to help.

Clustering has been widely used in many applications such as business intelligence, image pattern recognition, Web search, biology, and security. In business intelligence, clustering can be used to organize a large number of customers into groups, where customers within a group share strong similar characteristics. In image recognition, clustering can be used to discover clusters or “subclasses” in handwritten character recognition systems. Clustering has also found many applications in Web search. For example, a keyword search may often return a very large number of hits (i.e., pages relevant to the search) due to the extremely large number of web pages. Clustering can be used to organize the search results into groups and present the results in a concise and easily accessible way. Moreover, clustering techniques have been developed to cluster documents into topics (remember the google news example?), which are commonly used in information retrieval practice.

Clustering is also called **data segmentation** in some applications because clustering partitions large data sets into groups according to their *similarity*.

As a branch of statistics, clustering has been extensively studied, with the main focus on *distance-based cluster analysis*. Clustering tools were proposed like  **$k$ -means**, **fuzzy  $c$ -means**, and several other methods.

Many clustering algorithms have been introduced in the literature. Since clusters can formally be seen as subsets of the data set, one possible classification of clustering methods can be according to whether the subsets are **fuzzy** or **crisp (hard)**.

### Hard clustering

Hard clustering methods are based on classical set theory, and require that an object either does or does not belong to a cluster. Hard clustering means partitioning the data into a specified number of mutually exclusive subsets. The most common hard clustering method is  $k$ -means.

### Fuzzy clustering

Fuzzy clustering methods, however, allow the objects to belong to several clusters simultaneously, with different degrees of membership. In many situations, fuzzy clustering is more natural than hard clustering.

The most known technique of fuzzy clustering is the fuzzy  $c$ -means.

## 7.4 $k$ -Means

If you have ever watched a group of tourists with a couple of tour guides who hold umbrellas up so that everybody can see them and follow them, then you have seen a dynamic version of the  $k$ -means algorithm.  $k$ -means is even simpler, because the data (playing the part of the tourists) does not move, only the tour guides move.

Suppose that we want to divide our input data into  $K$  categories, where we know the value of  $K$ . We allocate  $K$  *cluster centres* (also called *prototypes* or *centroids*) to our input space, and we would like to position these centres so that there is one cluster centre in the middle of each cluster. However, we don't know where the clusters are, let alone where their 'middle' is, so we need an algorithm that will find them. Learning algorithms generally try to minimize some sort of error, so we need to think of an error criterion that describes this aim. There are two things that we need to define:

**A distance measure:** In order to talk about distances between points, we need some way to measure distances. It is often the normal **Euclidean** distance, but there are other alternatives like Manhattan distance, Correlation distance, Chessboard distance and other.

The Euclidean distance: Let  $x = (x_1, x_2)$  and  $y = (y_1, y_2)$  two observations in a two-dimensional space. The Euclidean distance  $d_{x,y}$  between  $x$  and  $y$  is

$$\begin{aligned} d_{x,y}^2 &= (x_1 - y_1)^2 + (x_2 - y_2)^2 \\ d_{x,y} &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \end{aligned}$$

**The mean average:** Once we have a distance measure, we can compute the central point of a set of data points, which is the mean average. Actually, this is only true in Euclidean space, which is the one we are used to, where everything is nice and flat.

We can now think about a suitable way of positioning the cluster centres: we compute the mean point of each cluster,  $\mathbf{v}_k$ ,  $i = 1, \dots, K$ , and put the cluster centre there. This is equivalent to minimizing the Euclidean distance (which is the sum-of-squares error) from each data point to its cluster centre. Then we decide which points belong to which clusters by associating each point with the cluster centre that it is closest to. This changes as the algorithm iterates. We start by positioning the cluster centres **randomly** though the input space, since we don't know where to put them, and we update their positions according to the data. We decide which cluster each data point belongs to by computing the distance between each data point and all of the cluster centres, and assigning it to the cluster that is the closest. For all the point that are assigned to a cluster, we then compute the mean of them, and move the cluster centre to that place. We iterate the algorithm until the cluster centres stop moving.

It is convenient at this point to define some notation to describe the assignment of data points to clusters. For each data point  $x_i$ , we introduce a corresponding set of binary indicator variables  $u_{ki} \in \{0, 1\}$ , where  $k = 1, \dots, K$  describing which of the  $K$  clusters the data point  $x_i$  is assigned to, so that if data point  $x_i$  is assigned to cluster  $k$  then  $u_{ki} = 1$ , and  $u_{kj} = 0$  for  $j \neq i$ . This is known as the 1-of- $K$  coding scheme. We can then define an objective function (and sometimes called a *distortion measure*), given by

$$J = \sum_{i=1}^N \sum_{k=1}^K u_{ki} \|x_i - \mathbf{v}_k\|^2$$

which represents the sum of the squares of the distances of each data point to its assigned vector  $\mathbf{v}_k$ . The goal is to find values for the  $\{u_{ki}\}$  and the  $\{\mathbf{v}_k\}$  so as to minimize  $J$ . We can do this through an iterative

procedure in which each iteration involves two successive steps corresponding to successive optimizations with respect to the  $u_{ki}$  and the  $\mathbf{v}_k$ . The algorithm of  $k$ -means is described in the following algorithm:

---

The  $k$ -means algorithm

---

**Data:**

$$\mathbf{X} = \{x_{ij}, i = 1, \dots, n, j = 1, \dots, p\}$$

**Result:**

Cluster centres (Prototypes)

**Initialization:**

- Choose a value for  $K$ .
- Choose  $K$  random positions in the input space.
- Assign the prototypes  $\mathbf{v}_k$  to those positions

**Learning: repeat**

**for** each data point  $x_i$  **do**

- compute the distance to each prototype:

$$d_{ki} = \min_k d(x_i, \mathbf{v}_k)$$

- assign the data point to the nearest prototype with distance

$$u_{ki} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \|x_i - \mathbf{v}_k\|^2 \\ 0 & \text{otherwise} \end{cases}$$

**for** each prototype **do**

- move the position of the prototype to the mean of the points in that cluster:

$$\mathbf{v}_k = \frac{\sum_i u_{ki} x_i}{\sum_i u_{ki}}$$

Until the prototypes stop moving.

---



The  $k$ -means algorithm produces

- A final estimate of cluster centroids (i.e. their coordinates).
- An assignment of each point to their respective cluster.

The denominator in the expression  $\mathbf{v}_k = \frac{\sum_i u_{ki} x_i}{\sum_i u_{ki}}$  is equal to the number of points assigned to cluster  $k$ , and so this result has a simple interpretation, namely set  $\mathbf{v}_k$  equal to the mean of all of the data points  $x_i$  assigned to cluster  $k$ . For this reason, the procedure is known as the  $k$ -means algorithm.

The two phases of re-assigning data points to clusters and re-computing the cluster means are repeated in turn until there is no further change in the assignments (or until some maximum number of iterations is exceeded). Because each phase reduces the value of the objective function  $J$ , convergence of the algorithm is assured. However, it may converge to a local rather than global minimum of  $J$ .

The  $k$ -means algorithm is illustrated using the Old Faithful data set <sup>1</sup> in following figure.

---

<sup>1</sup>Old Faithful, is a hydrothermal geyser in Yellowstone National Park in the state of Wyoming, U.S.A., and is a popular tourist attraction. Its name stems from the supposed regularity of its eruptions. The data set comprises 272 observations, each of which represents a single eruption and contains two variables corresponding to the duration in minutes of the eruption, and the time until the next eruption, also in minutes.

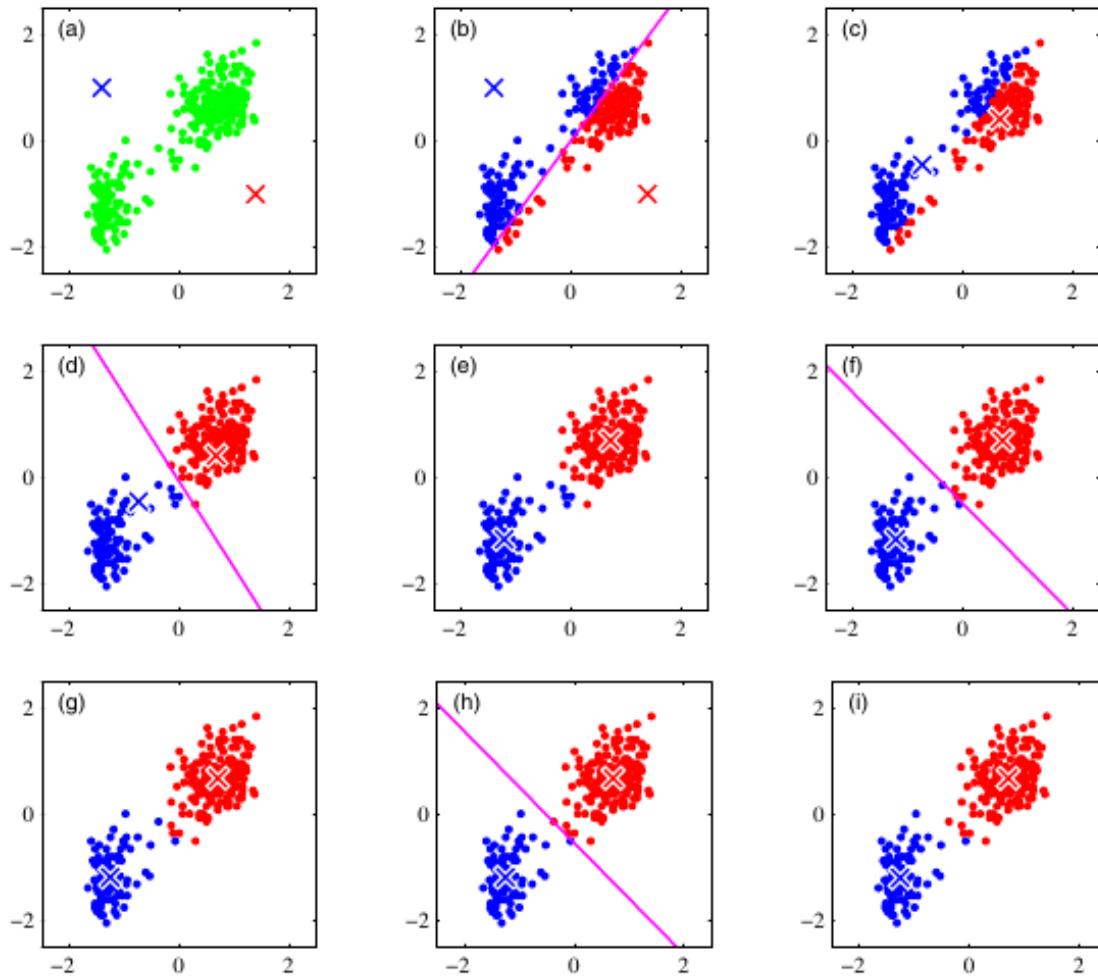


Figure 7.2: Illustration of the  $k$ -means algorithm using the re-scaled Old Faithful data set, where  $k = 2$ . We can see how the  $k$ -means algorithm works. (a) The first thing  $k$ -means has to do is assign an initial set of centroids. (b) The next stage in the algorithm assigns every point in the dataset to the closest centroid. (c) The next stage is the re-calculate the centroids based on the new cluster assignments of the data points. (d) Now we have completed one full cycle of the algorithm we can continue and re-assign points to their (new) closest cluster centroid. (e) And we can update the centroid positions one more time based on the re-assigned points. (g)(h)(f) The algorithm stops when we obtain the same results in consecutive iterations.

The  $k$ -means algorithm is illustrated using the Iris data set in following interactive figure<sup>2</sup>. (Try to modify the X and Y variables and the numbers of chosen clusters and see the result)

## 7.5 $k$ -means in R

We will use an example with simulated data to demonstrate how the  $k$ -means algorithm works. Here we simulate some data from three clusters and plot the dataset below.

```
set.seed(1234)
x <- rnorm(12,mean=rep(1:3,each=4),sd=0.2)
```

<sup>2</sup>Made by Joseph J. Allaire <https://github.com/jjallaire>

```
y <- rnorm(12,mean=rep(c(1,2,1),each=4),sd=0.2)
plot(x,y,col="blue",pch=19,cex=2)
text(x+0.05,y+0.05,labels=as.character(1:12))
```

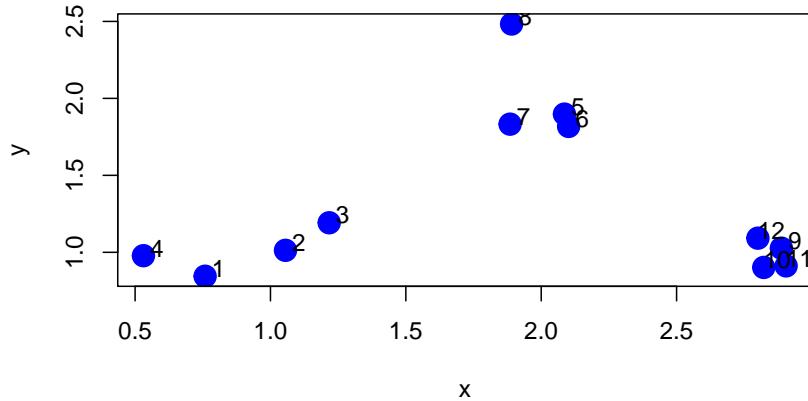


Figure 7.3: Simulated dataset

The `kmeans()` function in R implements the  $k$ -means algorithm and can be found in the `stats` package, which comes with R and is usually already loaded when you start R. Two key parameters that you have to specify are `x`, which is a matrix or data frame of data, and `centers` which is either an integer indicating the number of clusters or a matrix indicating the locations of the initial cluster centroids. The data should be organized so that each row is an observation and each column is a variable or feature of that observation.

```
dataFrame <- data.frame(x,y)
kmeansObj <- kmeans(dataFrame,centers=3)
names(kmeansObj)
#ans> [1] "cluster"      "centers"       "totss"         "withinss"
#ans> [5] "tot.withinss" "betweenss"    "size"          "iter"
#ans> [9] "ifault"
```

You can see which cluster each data point got assigned to by looking at the `cluster` element of the list returned by the `kmeans()` function.

```
kmeansObj$cluster
#ans> [1] 3 3 3 3 1 1 1 1 2 2 2 2
```

Here is a plot of the  $k$ -means clustering solution.

```
plot(x,y,col=kmeansObj$cluster,pch=19,cex=2)
points(kmeansObj$centers,col=1:3,pch=3,cex=3,lwd=3)
```

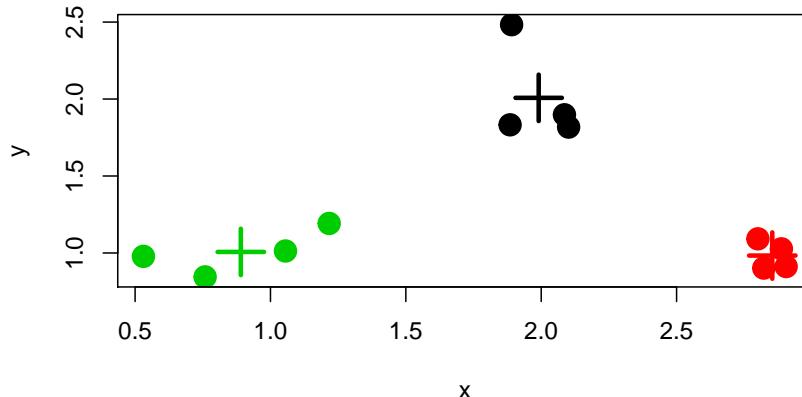


Figure 7.4:  $k$ -means clustering solution

### 7.5.1 Cluster Validity, Choosing the Number of Clusters

The result of a clustering algorithm can be very different from each other on the same data set as the other input parameters of an algorithm can extremely modify the behavior and execution of the algorithm. The aim of the cluster validity is to find the partitioning that best fits the underlying data. Usually 2D data sets are used for evaluating clustering algorithms as the reader easily can verify the result. But in case of high dimensional data the visualization and visual validation is not a trivial tasks therefore some formal methods are needed.

The process of evaluating the results of a clustering algorithm is called cluster validity assessment. Two measurement criteria have been proposed for evaluating and selecting an optimal clustering scheme:

- *Compactness*: The member of each cluster should be as close to each other as possible. A common measure of compactness is the variance.
- *Separation*: The clusters themselves should be widely separated. There are three common approaches measuring the distance between two different clusters: distance between the closest member of the clusters, distance between the most distant members and distance between the centres of the clusters.

There are three different techniques for evaluating the result of the clustering algorithms, and several *Validity measures* are proposed: Validity measures are scalar indices that assess the goodness of the obtained partition. Clustering algorithms generally aim at locating well separated and compact clusters. When the number of clusters is chosen equal to the number of groups that actually exist in the data, it can be expected that the clustering algorithm will identify them correctly. When this is not the case, misclassifications appear, and the clusters are not likely to be well separated and compact. Hence, most cluster validity measures are designed to quantify the separation and the compactness of the clusters.



Check this answer on [stackoverflow](#) containing R code for several methods of computing an optimal value of  $k$  for  $k$ -means cluster analysis: [here](#).

# PW 7

In this practical work we will learn how to create a report in `Rstudio` using `Rmarkdown` files. Then we will apply the *k*-means clustering algorithm using the standard function `kmeans()` in R, we will use the dataset `Ligue1_2017/2018`.

## Reporting

### Markdown

Markdown is a lightweight markup language with plain text formatting syntax designed so that it can be converted to HTML and many other formats (pdf, docx, etc..).

Click here to see an example of a markdown (.md) syntaxes and the result in HTML. The markdown syntaxes are on right and their HTML result is on left. You can modify the source text to see the result.



**Extra:** There are some markdown online editors you can use, like [dillinger.io/](https://dillinger.io/). See the Markdown source file and the HTML preview. Play with the source text to see the result in the preview.

### R Markdown

R Markdown is a variant of Markdown that has embedded R code chunks, to be used with the `knitr` package to make it **easy to create reproducible web-based reports**.

First, in `Rstudio` create a new R Markdown file. A default template will be opened. There is some R code in R chunks. Click on knit, save your file and see the produced output. The output is a html report containing the results of the R codes. If your file is named `report.Rmd`, your report is named `report.html`.



- Make sure to have the latest version of `Rstudio`.
- If you have problems creating a R Markdown file (problem in installing packages, etc..) close your `Rstudio` and reopen it with administrative tools and retry.
- If it doesn't work and in order to not lose more time, write your script in an .R file with your comments and submit it.



- Be ready to submit your report (your .html file) at the end of each class.
- Your report must be named:

`YouLastName_YourFirstName_WeekNumber.html`

You can find all the informations about R Markdown on this site: [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com).

You may also find the following resources helpful:

- The R Markdown Reference Guide
- The R Markdown Cheatsheet

## The report to be submitted

In **Rstudio**, start by creating a R Markdown file. When you create it a default template will be opened with the following first lines:

```
---
title: "Untitled"
output: html_document
---
```

These lines are the YAML header in which you choose the settings of your report (title, author, date, appearance, etc..)

For your submitted report, use the following YAML header:

```
---
title: "Week 7"
subtitle: "Clustering"
author: LastName FirstName
date: "`#r format(Sys.time())`" # remove the # to show the date
output:
  html_document:
    toc: true
    toc_depth: 2
    theme: flatly
---
```



**Very Important Remark:** Click on the settings button of Rstudio's text editor and choose to Chunk Output in Console.

The screenshot shows the RStudio interface with an R Markdown document open. The code editor pane contains R code for setting up a chunk, defining a title and output type, and demonstrating various knitting options. A context menu is open over a code chunk, specifically the one at line 10. The menu items include 'Preview in Window', 'Preview in Viewer Pane', '(No Preview)', 'Preview Images and Equations', 'Show Previews Inline', 'Chunk Output Inline' (which is checked), 'Chunk Output in Console' (which is selected and highlighted in blue), 'Expand All Output', 'Collapse All Output', 'Clear Output', 'Clear All Output', and 'Output Options...'. The 'Chunk Output in Console' option is currently active.

```

1 ---  

2 title: "untitled"  

3 output: html_document  

4 ---  

5  

6 ```{r setup, include=FALSE}  

7 knitr::opts_chunk$set(echo=TRUE)  

8 ...  

9  

10 ## R Markdown  

11  

12 This is an R Markdown document.  

13 It can be easily converted to HTML,  

14 PDF, and MS Word documents.  

15 <http://rmarkdown.rstudio.com>  

16 when you click the Knit button.  

17 You can also embed plots, for example:  

18  

19 ```{r cars}  

20 summary(cars)  

21 ...  

22  

23 ## Including Plots  

24 You can also embed plots, for example:  

25  

26 ```{r pressure, echo=FALSE}  

27 plot(pressure)  

28 ...  

29  

30 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of  

31 the R code that generated the plot.

```

Figure 7.5:

In the core of your report:

- Put every exercise in a section, name the section `Exercise i` (i is the exercise's number).
- Paste the exercise content.
- Write the code of the exercise in R chunks.
- Run the chunk to make sure it works.
- If there is a need, explain the results.
- Click on `knit`



Submit a report following the instructions above. In the first exercise you must continue the analysis of the Boston data set.

## k-means clustering

1. Download the dataset: Ligue1 2017-2018 and import it into R. Put the argument `row.names` to 1.

```
# You can import directly from my website (instead of downloading it...)  

ligue1 <- read.csv("http://mghassany.com/MLcourse/datasets/ligue1_17_18.csv", row.names=1, sep=";")
```

2. Print the first two rows of the dataset and the total number of features in this dataset.



You can create an awesome HTML table by using the function `kable` from the `knitr` library. For example, if you want to show the first 5 lines and 5 columns of your dataset, you can use `knitr::kable(ligue1[1:5,1:5])`. Give it a try and see the result on your html report!

### pointsCards

3. We will first consider a smaller dataset to easily understand the results of  $k$ -means. Create a new dataset in which you consider only `Points` and `Yellow.cards` from the original dataset. Name it `pointsCards`
4. Apply  $k$ -means on `pointsCards`. Chose  $k = 2$  clusters and put the number of iterations to 20. Store your results into `km`. (**Remark:** `kmeans()` uses a random initialization of the clusters, so the results may vary from one call to another. Use `set.seed()` to have reproducible outputs).
5. Print and describe what is inside `km`.
6. What are the coordinates of the centers of the clusters (called also prototypes or centroids) ?
7. Plot the data (`Yellow.cards` vs `Points`). Color the points corresponding to their cluster.
8. Add to the previous plot the clusters centroids and add the names of the observations.
9. Re-run  $k$ -means on `pointsCards` using 3 and 4 clusters and store the results into `km3` and `km4` respectively. Visualize the results like in question 7 and 8.



How many clusters  $k$  do we need in practice? There is not a single answer: the advice is to try several and compare. Inspecting the ‘between\_SS / total\_SS’ for a good trade-off between the number of clusters and the percentage of total variation explained usually gives a good starting point for deciding on  $k$  (criterion to select  $k$  similar to PCA).

There is several methods of computing an optimal value of  $k$  with R code on following stackoverflow answer: [here](#).

10. Visualize the “within groups sum of squares” of the  $k$ -means clustering results (use the code in the link above).
11. Modify the code of the previous question in order to visualize the ‘between\_SS / total\_SS’. Interpret the results.

### Ligue 1

So far, you have only taken the information of two variables for performing clustering. Now you will apply `kmeans()` on the original dataset `ligue1`. Using PCA, we can visualize the clustering performed with all the available variables in the dataset.

By default, `kmeans()` does not standardize the variables, which will affect the clustering result. As a consequence, the clustering of a dataset will be different if one variable is expressed in millions or in tenths. If you want to avoid this distortion, use `scale` to automatically center and standardize the dataset (the result will be a matrix, so you need to transform it to a data frame again).

12. Scale the dataset and transform it to a data frame again. Store the scaled dataset into `ligue1_scaled`.
13. Apply `kmeans()` on `ligue1` and on `ligue1_scaled` using 3 clusters and 20 iterations. Store the results into `km.ligue1` and `km.ligue1_scaled` respectively (do not forget to set a seed)

- 14.** How many observations there are in each cluster of `km.ligue1` and `km.ligue1.scaled`? (you can use `table()`). Do you obtain the same results when you perform `kmeans()` on the scaled and unscaled data?

## PCA

- 15.** Apply PCA on `ligue1` dataset and store you results in `pcaligue1`. Do we need to apply PCA on the scaled dataset? Justify your answer.

- 16.** Plot the observations and the variables on the first two principal components (biplot). Interpret the results.

- 17.** Visualize the teams on the first two principal components and color them with respect to their cluster.

```
# You can use the following code, based on `factoextra` library.
fviz_cluster(km.ligue1, data = ligue1, # km.ligue1 is where you stored your kmeans results
             palette = c("red", "blue", "green"), # 3 colors since 3 clusters
             ggtheme = theme_minimal(),
             main = "Clustering Plot")
)
```

- 18.** Recall that the figure of question **17** is a visualization with PC1 and PC2 of the clustering done with all the variables, not on PC1 and PC2. Now apply the `kmeans()` clustering taking only the first two PCs instead the variables of original dataset. Visualize the results and compare with the question **17**.



By applying  $k$ -means only on the PCs we obtain different and less accurate result, but it is still an insightful way.

## Implementing k-means

In this part, you will perform  $k$ -means clustering manually, with  $k = 2$ , on a small example with  $n = 6$  observations and  $p = 2$  features. The observations are as follows.

Observation	$X_1$	$X_2$
1	1	4
2	1	3
3	0	4
4	5	1
5	6	2
6	4	0

- 19.** Plot the observations.
- 20.** Randomly assign a cluster label to each observation. You can use the `sample()` command in R to do this. Report the cluster labels for each observation.
- 21.** Compute the centroid for each cluster.
- 22.** Create a function that calculates the Euclidean distance for two observations.
- 23.** Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.
- 24.** Repeat **21** and **23** until the answers obtained stop changing.

- 25.** In your plot from **19**, color the observations according to the cluster labels obtained.

# Chapter 8

## Gaussian Mixture Models & EM

In the previous chapter we saw the  $k$ -means algorithm which is considered as a hard clustering technique, such that each point is allocated to only one cluster. In  $k$ -means, a cluster is described only by its centroid. This is not too flexible, as we may have problems with clusters that are *overlapping*, or ones that are not of *circular shape*.

In this chapter, we will introduce a model-based clustering technique, which is **Expectation Maximization** (EM). We will apply it using Gaussian Mixture Models (GMM).

With EM Clustering, we can go a step further and describe each cluster by its centroid (mean), covariance (so that we can have elliptical clusters), and weight (the size of the cluster). The probability that a point belongs to a cluster is now given by a multivariate Gaussian probability distribution (multivariate - depending on multiple variables). That also means that we can calculate the probability of a point being under a Gaussian ‘bell’, i.e. the probability of a point belonging to a cluster. A comparison between the performances of  $k$ -means and EM clustering on an artificial dataset is shown in Figure 8.1.

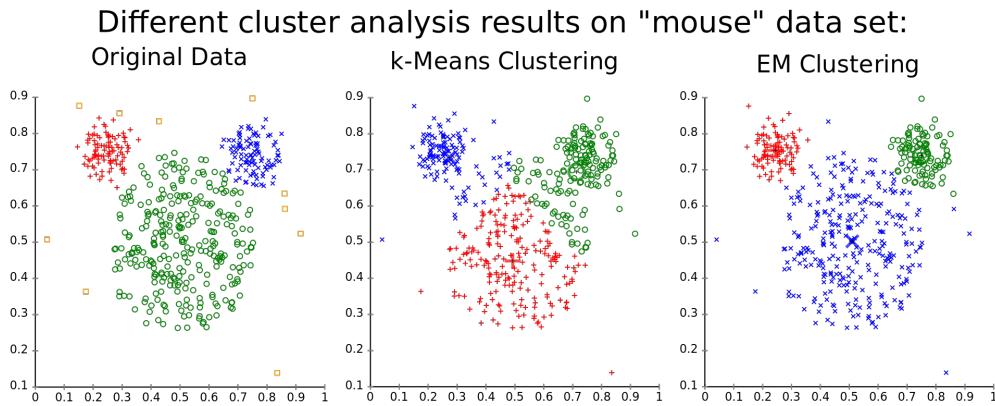


Figure 8.1: Comparison of  $k$ -means and EM on artificial data called [Mouse](<https://elki-project.github.io/datasets/>) dataset. Using the Variances, the EM algorithm can describe the normal distributions exact, while  $k$ -means splits the data in [Voronoi](<https://bit.ly/1rVyJkt>)-Cells.

We start this chapter by reminding what is a Gaussian distribution, then introduce the Mixture of Gaussians and finish by explaining the Expectation-Maximization algorithm.

## 8.1 The Gaussian distribution

The Gaussian, also known as the normal distribution, is a widely used model for the distribution of continuous variables. In the case of a single variable  $x$ , the Gaussian distribution can be written in the form

$$\mathcal{N}(x|m, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x - m)^2\right\} \quad (8.1)$$

where  $m$  is the mean and  $\sigma^2$  is the variance.

For a  $D$ -dimensional vector  $X$ , the multivariate Gaussian distribution take the form

$$\mathcal{N}(X|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(X - \mu)^T \Sigma^{-1} (X - \mu)\right\} \quad (8.2)$$

where  $\mu$  is a  $D$ -dimensional mean vector,  $\Sigma$  is a  $D \times D$  covariance matrix, and  $|\Sigma|$  denotes the determinant of  $\Sigma$ .

The Gaussian distribution arises in many different contexts and can be motivated from a variety of different perspectives. For example, when we consider the sum of multiple random variables. The *central limit theorem* (due to Laplace) tells us that, subject to certain mild conditions, the sum of a set of random variables, which is of course itself a random variable, has a distribution that becomes increasingly Gaussian as the number of terms in the sum increases.

## 8.2 Mixture of Gaussians

While the Gaussian distribution has some important analytical properties, it suffers from significant limitations when it comes to modeling real data sets. Consider the example shown in Figure 8.2 applied on the ‘Old Faithful’ data set, this data set comprises 272 measurements of the eruption of the Old Faithful geyser at Yellowstone National Park in the USA. Each measurement comprises the duration of the eruption in minutes (horizontal axis) and the time in minutes to the next eruption (vertical axis). We see that the data set forms two dominant clumps, and that a simple Gaussian distribution is unable to capture this structure, whereas a linear superposition of two Gaussians gives a better characterization of the data set. Such superpositions, formed by taking linear combinations of more basic distributions such as Gaussians, can be formulated as probabilistic models known as *mixture distributions*.

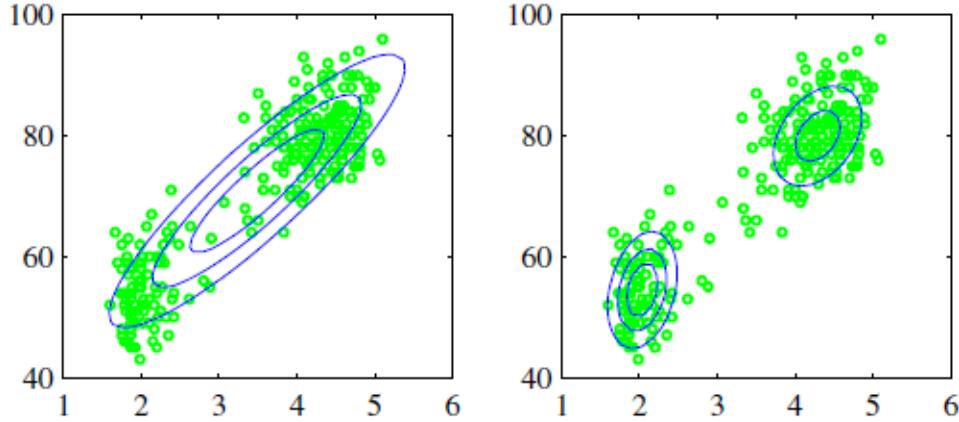


Figure 8.2: Plots of the 'old faithful' data in which the blue curves show contours of constant probability density. On the left is a single Gaussian distribution which has been fitted to the data using maximum likelihood. On the right the distribution is given by a linear combination of two Gaussians which has been fitted to the data by maximum likelihood using the EM technique, and which gives a better representation of the data

In Figure 8.3 we see that a linear combination of Gaussians can give rise to very complex densities. By using a sufficient number of Gaussians, and by adjusting their means and covariances as well as the coefficients in the linear combination, almost any continuous density can be approximated to arbitrary accuracy.

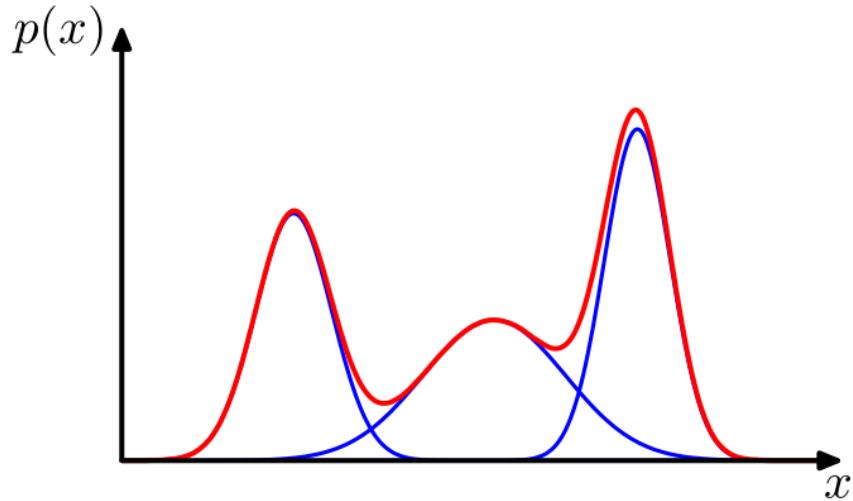


Figure 8.3: Example of a Gaussian mixture distribution in one dimension showing three Gaussians (each scaled by a coefficient) in blue and their sum in red.

We therefore consider a superposition of  $K$  Gaussian densities of the form

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

which is called a *mixture of Gaussians*. Each Gaussian density  $\mathcal{N}(x|\mu_k, \Sigma_k)$  is called a *component* of the mixture and has its own mean  $\mu_k$  and covariance  $\Sigma_k$ .

The parameters  $\pi_k$  are called *mixing coefficients*. They verify the conditions

$$\sum_{k=1}^K \pi_k = 1 \quad \text{and} \quad 0 \leq \pi_k \leq 1$$

In order to find an equivalent formulation of the Gaussian mixture involving an explicit **latent** variable, we introduce a  $K$ -dimensional binary random variable  $z$  having a 1-of- $K$  representation in which a particular element  $z_k$  is equal to 1 and all other elements are equal to 0. The values of  $z_k$  therefore satisfy  $z_k \in \{0, 1\}$  and  $\sum_k z_k = 1$ , and we see that there are  $K$  possible states for the vector  $z$  according to which element is nonzero. The marginal distribution over  $z$  is specified in terms of the mixing coefficients  $\pi_k$ , such that

$$p(z_k = 1) = \pi_k$$



A latent variable is a variable that is not directly measurable, but its value can be inferred by taking other measurements.

This happens a lot in machine learning, robotics, statistics and other fields. For example, you may not be able to directly quantify intelligence (it's not a countable thing like the number of brain cells you have), but we think it exists and we can run experiments that may tell us about intelligence. So your intelligence is a latent variable that affects your performance on multiple tasks even though it can not be directly measured (link).

The conditional distribution of  $x$  given a particular value for  $z$  is a Gaussian

$$p(x|z_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k)$$

The joint distribution is given by  $p(z)p(x|z)$ , and the marginal distribution of  $x$  is then obtained by summing the joint distribution over all possible states of  $z$  to give

$$p(x) = \sum_z p(z)p(x|z) = \sum_{k=1}^K \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

Now, we are able to work with the joint distribution  $p(x|z)$  instead of the marginal distribution  $p(x)$ . This leads to significant simplification, most notably through the introduction of the Expectation-Maximization (EM) algorithm.

Another quantity that play an important role is the conditional probability of  $z$  given  $x$ . We shall use  $r(z_k)$  to denote  $p(z_k = 1|x)$ , whose value can be found using Bayes' theorem

$$\begin{aligned} r(z_k) = p(z_k = 1|x) &= \frac{p(z_k = 1)p(x|(z_k = 1))}{\sum_{j=1}^K p(z_j = 1)p(x|(z_j = 1))} \\ &= \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)} \end{aligned} \tag{8.3}$$

We shall view  $\pi_k$  as the prior probability of  $z_k = 1$ , and the quantity  $r(z_k)$  as the corresponding posterior probability once we have observed  $x$ . As we shall see in next section,  $r(z_k)$  can also be viewed as the *responsibility* that component  $k$  takes for 'explaining' the observation  $x$ .



Doesn't this reminds you of the Equation 4.1 when we used Bayes' theorem for Classification? where  $p_k(x)$  was the *posterior* probability that an observation  $X = x$  belongs to  $k$ -th class. The difference is that here the data is unlabeled (we have no class), so we create a **latent** (hidden, unobserved) variable  $z$  that will play a similar role.

In Figure 8.4 the role of the responsibilities is illustrated on a sample of 500 points drawn from a mixture of three Gaussians.

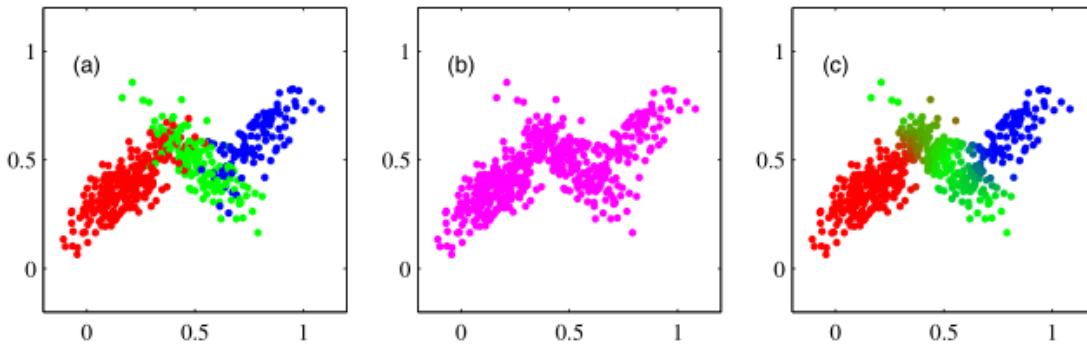


Figure 8.4: Example of 500 points drawn from a mixture of 3 Gaussians. (a) Samples from the joint distribution  $p(z)p(x|z)$  in which the three states of  $z$ , corresponding to the three components of the mixture, are depicted in red, green, and blue, and (b) the corresponding samples from the marginal distribution  $p(x)$ , which is obtained by simply ignoring the values of  $z$  and just plotting the  $x$  values. The data set in (a) is said to be complete, whereas that in (b) is incomplete. (c) The same samples in which the colours represent the value of the responsibilities  $r(z_{nk})$  associated with data point  $x_n$ , obtained by plotting the corresponding point using proportions of red, blue, and green ink given by  $r(z_{nk})$  for  $k = 1, 2, 3$ , respectively.

So the form of the Gaussian mixture distribution is governed by the parameters  $\pi$ ,  $\mu$  and  $\Sigma$ , where we have used the notation  $\pi = \{\pi_1, \dots, \pi_K\}$ ,  $\mu = \{\mu_1, \dots, \mu_K\}$  and  $\Sigma = \{\Sigma_1, \dots, \Sigma_K\}$ . One way to set the values of these parameters is to use maximum likelihood. The log of the likelihood function is given by

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k) \right\}$$

We immediately see that the situation is now much more complex than with a single Gaussian, due to the presence of the summation over  $k$  inside the logarithm. As a result, the maximum likelihood solution for the parameters no longer has a closed-form analytical solution. One approach to maximizing the likelihood function is to use iterative numerical optimization techniques. Alternatively we can employ a powerful framework called **Expectation Maximization**, which will be discussed in this chapter.

## 8.3 EM for Gaussian Mixtures

Suppose we have a data set of observations  $\{x_1, \dots, x_N\}$ , which gives a data set  $X$  of size  $N \times D$ , and we wish to model this data using a mixture of Gaussians. Similarly, the corresponding latent variable are denoted by a  $N \times K$  matrix  $Z$  with rows  $z_n^K$ .



Recall that the objective is to estimate the parameters  $\pi$ ,  $\mu$  and  $\Sigma$  in order to estimate the posterior probabilities (named also *responsibilities*, called  $r(z_k)$  in this chapter). To do so, we find the estimators that maximize the log of the likelihood function.

If we assume that the data points are i.i.d. (independent and identically distributed), then we can calculate the log of the likelihood function, which is given by

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right\} \quad (8.4)$$

An elegant and powerful method for finding maximum likelihood solutions for this models with latent variables is called the **Expectation Maximization** algorithm, or EM algorithm.

Setting the derivatives of  $\ln p(X|\pi, \mu, \Sigma)$  in (8.4) respectively with respect to the  $\mu_k$ ,  $\Sigma_k$  and  $\pi_k$  to zero, we obtain

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N r(z_{nk}) x_n \quad (8.5)$$

where we define

$$N_k = \sum_{n=1}^N r(z_{nk})$$

We can interpret  $N_k$  as the effective number of points assigned to cluster  $k$ . Note carefully the form of this solution. We see that the mean  $\mu_k$  for the  $k$ -th Gaussian component is obtained by taking a weighted mean of all of the points in the data set, in which the weighting factor for data point  $x_n$  is given by the posterior probability  $r(z_{nk})$  that component  $k$  was responsible for generating  $x_n$ .

As for  $\sigma_k$  we obtain

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N r(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T \quad (8.6)$$

which has the same form as the corresponding result for a single Gaussian fitted to the data set, but again with each data point weighted by the corresponding posterior probability and with the denominator given by the effective number of points associated with the corresponding component.

Finally, for the mixing coefficients  $\pi_k$  we obtain

$$\pi_k = \frac{N_k}{N} \quad (8.7)$$

so that the mixing coefficient for the  $k$ -th component is given by the average responsibility which that component takes for explaining the data points.

We first choose some initial values for the means, covariances, and mixing coefficients. Then we alternate between the following two updates that we shall call the **E** step and the **M** step. In the *expectation* step, or E step, we use the current values for the parameters to evaluate the posterior probabilities, or responsibilities, given by Eq. (8.3). We then use these probabilities in the *maximization* step, or M step, to re-estimate the means, covariances, and mixing coefficients using the results in Equations (8.5), (8.6) and (8.7). The algorithm of EM for mixtures of Gaussians is shown in the following Algorithm:

The EM for Gaussian mixtures

**Data:**

$\mathbf{X} = \{x_{kd}, k = 1, \dots, N, d = 1, \dots, D\}$  where  $D$  is the dimension of the feature space.  $Z$  the latent variables matrix. Posterior probabilities  $r(z_{nk})$  and the model parameters  $\mu, \Sigma$  and  $\pi$ .

**Initialization:**

- Choose a value for  $K$ ,  $1 < K < N$ .
- Initialize the means  $\mu_k$ , the covariances  $\Sigma_k$  and mixing coefficients  $\pi_k$  randomly.
- Evaluate the initial value of the log likelihood.

**Learning: repeat**

**E step:**

- Evaluate the responsibilities using the current parameter values:

$$r(z_{nk}) = \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)}$$

**M step:**

- Re-estimate the parameters using the current responsibilities:

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N r(z_{nk}) x_n$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N r(z_{nk}) (x_n - \mu_k) (x_n - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N}$$

$$\text{where } N_k = \sum_{n=1}^N r(z_{nk})$$

- Evaluate the log likelihood:

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right\}$$

Until convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to E step.

The EM algorithm for a mixture of two Gaussians applied to the rescaled Old Faithful data set is illustrated in Figure 8.5. In plot (a) we see the initial configuration, the Gaussian component are shown as blue and red circles. Plot (b) shows the result of the initial E step where we update the responsibilities. Plot (c) shows the M step where we update the parameters. Plots (d), (e), and (f) show the results after 2, 5, and 20 complete cycles of EM, respectively. In plot (f) the algorithm is close to convergence.

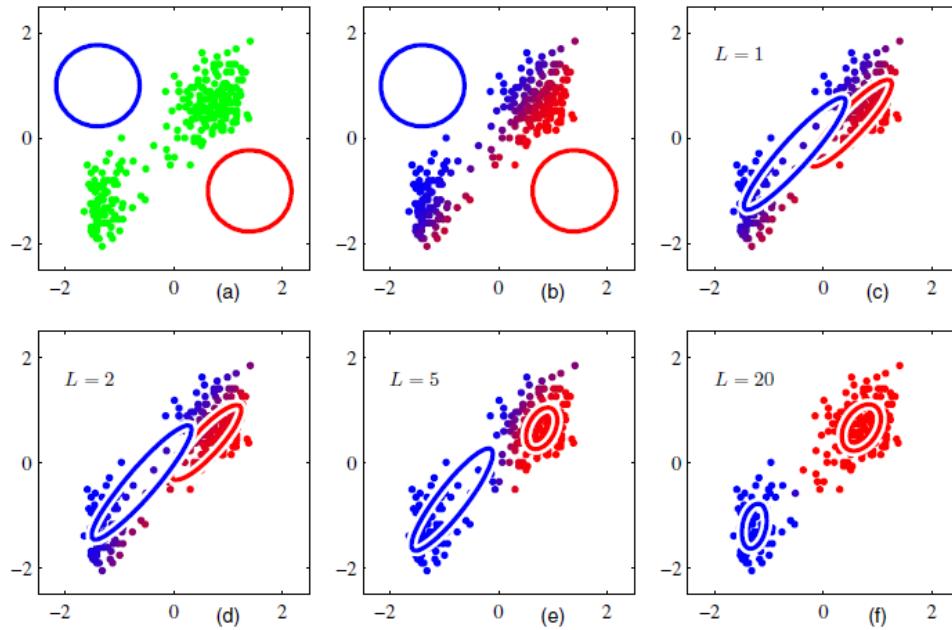


Figure 8.5: Illustration of the EM algorithm using the Old Faithful dataset. A mixture of two Gaussians is used.



Summary of this chapter:

- Gaussian Mixture Models (GMM) take a Gaussian and add another Gaussian(s).
- This allows to model more complex data.
- We fit a GMM with the Expectation-Maximization (EM) algorithm.
- Expectation-Maximization (EM) algorithm is a series of steps to find good parameter estimates when there are latent variables.
- EM steps:
  1. Initialize the parameter estimates.
  2. Given the current parameter estimates, find the minimum log likelihood for  $Z$  (data + latent variables).
  3. Given the current data, find better parameter estimates.
  4. Repeat steps 2 & 3.

# PW 8

In the first part of this PW you will:

- Learn how to fit a Gaussian Mixture Model (GMM) using the `mclust` R package.
- Compare  $k$ -means and GMM on artificial data (2-D data).
- Fit a GMM on univariate (1-D) simulated data.

In the second part of this PW you will build an algorithm from **scratch**. Your algorithm must fit a GMM model using the **Expectation-Maximization (EM)** technique on any multi dimensional dataset.

## Report template

For this session, write your report in a RMarkdown script, in which you must use the following YAML header settings (replace the default YAML header by this one, edit the author's name and show the date):

```
---
```

```
title: "Week 8"
subtitle: "Gaussian Mixture Models & EM"
author: LastName FirstName
date: "`#r format(Sys.time())`" # remove the # to show the date
output:
  html_document:
    toc: true
    toc_depth: 2
    toc_float: true
    theme: cerulean
    highlight: espresso
---
```

## 8.4 EM using `mclust`

### GMM vs $k$ -means

In this section, we will use two artificial (simulated) datasets in which we know the ground truth (true labels) in order to compare the performances of  $k$ -means and GMM. To fit a GMM using EM technique you need to install and use the package `mclust`.

1. Download and import `Data1` and `Data2`. Plot both of the datasets on the same window. Color the observations with respect to the ground truth, like in Figure 8.6.

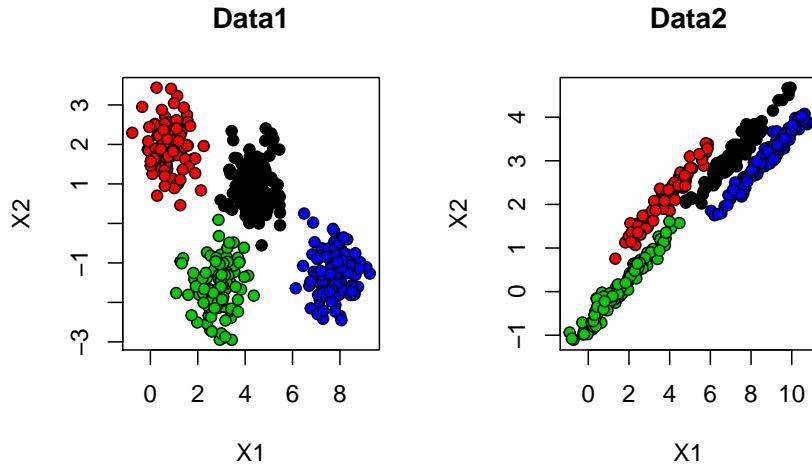


Figure 8.6: Data1 is plotted on left. Data2 on right. Colors shown with respect to ground truth.

2. Apply  $k$ -means on both datasets with 4 clusters. Plot both of the dataset on the same window and color the observations with respect to  $k$ -means results. Interpret the results.



One way to think about the  $k$ -means model is that it places a circle (or, in higher dimensions, a hyper-sphere) at the center of each cluster, with a radius defined by the most distant point in the cluster. This radius acts as a hard cutoff for cluster assignment within the training set: any point outside this circle is not considered a member of the cluster. You can try to visualize the circles on your plots.

3. Now fit a GMM model on the datasets. To do so, load the `mclust` library. Then you can use the function `Mclust()` on your data (this function will choose automatically the number of mixtures, basing on BIC criterion). Use the clustering results from your GMM model to visualize the results on both of the datasets, color the observations with respect to the clusters obtained from the GMM model. Interpret the results.

**In the following questions from this section, explore the `mclust` library and what it offers. Apply its functions on Data2.**



`mclust` is a contributed R package for model-based clustering, classification, and density estimation based on finite normal mixture modelling. It provides functions for parameter estimation via the EM algorithm for normal mixture models with a variety of covariance structures, and functions for simulation from these models. Also included are functions that combine model-based hierarchical clustering, EM for mixture estimation and the Bayesian Information Criterion (BIC) in comprehensive strategies for clustering, density estimation and discriminant analysis. Additional functionalities are available for displaying and visualizing fitted models along with clustering, classification, and density estimation results.

4. Show the summary of the GMM model you fitted on Data2. Explain what it shows.
5. `mclust` package offers some visualization. To plot your two-dimensional data, use the standard plot function applied on your model. Apply the following code, given that the model is named `gmm_model`, and interpret what it shows.

```
plot(gmm_model, what = "classification")
plot(gmm_model, what = "uncertainty")
```

6. mclust package uses the Bayesian Information Criterion (BIC) to choose the best number of mixtures. To see the values of BIC for different number of mixtures use the following code.

```
plot(gmm_model, what = "BIC")
```

Information criteria are based on penalised forms of the log-likelihood. As the likelihood increases with the addition of more components, a penalty term for the number of estimated parameters is subtracted from the log-likelihood. The BIC is a popular choice in the context of GMMs, and takes the form

$$\text{BIC} \approx 2\ell(X|\hat{\theta}) - \nu \log(n)$$

where  $\theta$  is the set of parameters (in GMM it is  $\theta = \{\mu, \Sigma, \pi\}$ ), and  $\ell(X|\hat{\theta})$  is the log-likelihood at the Maximum Likelihood Estimators  $\hat{\theta}$  for the model,  $n$  is the sample size, and  $\nu$  is the number of estimated parameters. We select the model that maximises BIC.



What you see on the figure showing the BIC values are different parameterisations of the within-group covariance matrix  $\Sigma_k$ . In GMM, clusters are ellipsoidal, centered at the mean vector  $\mu_k$ , and with other geometric features, such as volume, shape and orientation, determined by the covariance matrix  $\Sigma_k$ .

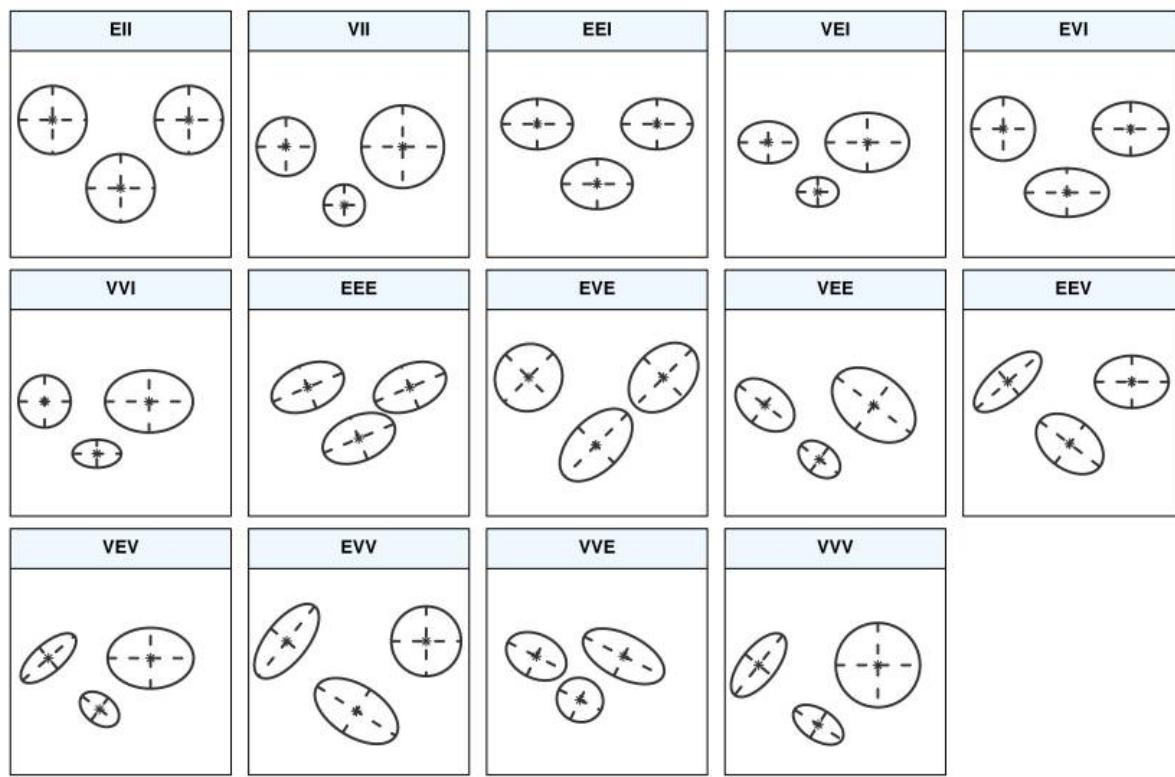


Figure 8.7:

7. Though GMM is often categorized as a clustering algorithm, fundamentally it is an algorithm for density estimation. That is to say, the result of a GMM fit to some data is technically not a clustering model, but a generative probabilistic model describing the distribution of the data. Density estimation plays an important role in applied statistical data analysis and theoretical research. A density estimate based on GMM can be obtained using the function `densityMclust()`. Apply it on Data2 and visualize the estimated densities (show an “image” and a “perspective” plot of the bivariate density estimate).

## EM on 1D

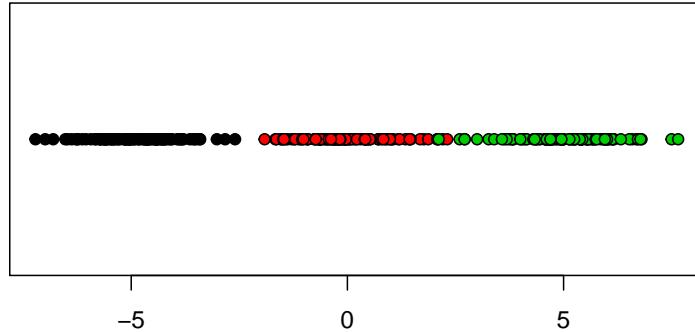
In this part you must fit a GMM model on a one dimensional simulated data.

8. Create a data table of 300 observations in which you have two columns:

- The first column contains generated data. Those data are generated from three Gaussian distributions with different parameters.
- The second column corresponds to the ground truth (every observation was generated from which Gaussian).
- Hint: functions you may need are `rnorm()`, `rep()`, `rbind()` or `cbind()`.
- You must of course **set a seed** (your student\_pk). An example of 9 generated values from three Gaussians is shown in the following table:

X	source
-5.626	1
-4.816	1
-5.836	1
1.595	2
0.3295	2
-0.8205	2
5.487	3
5.738	3
5.576	3

9. Show your generated data on one axis (this kind of figures are called stripchart), color them with respect to ground truth, you must obtain something like:



10. Plot the histogram corresponding to your generated data. Interpret it.

**11.** Fit a GMM model on your generated data. Print the summary and visualize your results. Explain your results.

**12.** Apply a density estimate on your generated data and visualize it. Interpret the obtained figure.

## 8.5 EM from scratch

In this second part of this PW you will build a GMM model from scratch, you must develop the EM technique to fit the model.

**2.1** Generate a two-dimensional dataset from a  $k$ -component Gaussian mixture density with different means and different covariance matrices. It is up to you to choose the mixing proportions  $\{\pi_1, \dots, \pi_k\}$ .

**2.2** Implement the EM algorithm to fit a GMM on your generated data:

- Initialize the mixing proportions and the covariance matrices (e.g., you can initialize with equal mixing proportions and Identity covariance matrices).
- Initialize the means “randomly” (by your own choice of  $k$ ).
- In the EM training loop, store the value of the observed-data log-likelihood at each iteration.
- At convergence, plot the log-likelihood curve.

**2.3** Create a function that selects the number of mixture components by computing the values of BIC criterion for  $k$  varying from 1 to 10.

**2.4** On your generated data, compare your results obtained with the algorithm you developed and the ground truth (in terms of the chosen number of mixture components; and in terms of error rate).

**2.5** Apply the algorithm you developed on  Iris dataset.



To visualize your results on Iris dataset, you can use PCA projection and coloring with respect to clustering results.

The package `mclust` provides also a dimensionality reduction technique in function `MclustDR()`.



## Appendix A

# Introduction to RStudio

RStudio is the most employed Integrated Development Environment (IDE) for R nowadays. When you start RStudio you will see a window similar to Figure A.1. There are a lot of items in the GUI, most of them described in the RStudio IDE Cheat Sheet. The most important things to keep in mind are:

1. The code is written in scripts in the *source panel* (upper-right panel in Figure A.1);
2. for running a line or code selection from the script in the *console* (first tab in the lower-right panel in Figure A.1), you do it with the keyboard shortcut '**Ctrl+Enter**' (Windows and Linux) or '**Cmd+Enter**' (Mac OS X).

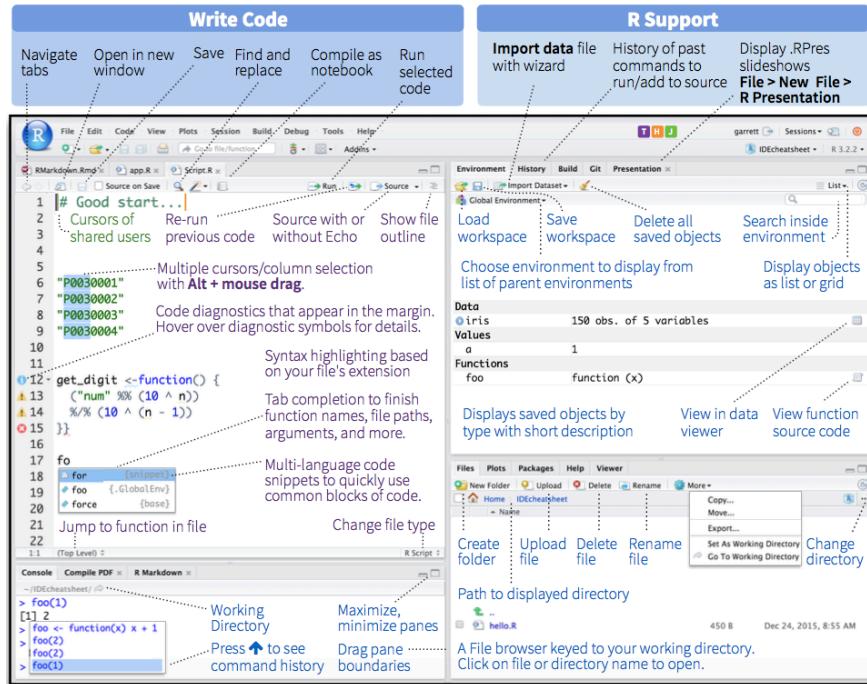


Figure A.1: Main window of ‘RStudio’. The red shows the code panel and the yellow shows the console output. Extracted from [here](<https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf>).



## Appendix B

# Review on hypothesis testing

The process of hypothesis testing has an interesting analogy with a trial that helps on understanding the elements present in a formal hypothesis test in an intuitive way.

Hypothesis testing	Trial
Null hypothesis $H_0$	<b>Accused of committing a crime.</b> It has the “presumption of innocence”, which means that it is <i>not guilty</i> until there is enough evidence to supporting its guilt
Sample $X_1, \dots, X_n$	Collection of small <b>evidences supporting innocence and guilt.</b> These evidences contain a certain degree of uncontrollable randomness because of how they were collected and the context regarding the case
Statistic $T_n$	<b>Summary of the evicences presented</b> by the prosecutor and defense lawyer
Distribution of $T_n$ under $H_0$	The <b>judge</b> conducting the trial. Evaluates the evidence presented by both sides and presents a verdict for $H_0$
Significance level $\alpha$	$1 - \alpha$ is the <b>strength of evidences required by the judge for condemning <math>H_0</math>.</b> The judge allows evidences that on average condemn $100\alpha\%$ of the innocents, due to the randomness inherent to the evidence collection process. $\alpha = 0.05$ is considered a reasonable level
$p$ -value	<b>Decision</b> of the judge that measures the degree of compatibility, in a scale 0–1, of the presumption of innocence with the summary of the evidences presented. If $p\text{-value} < \alpha$ , $H_0$ is declared guilty. Otherwise, is declared not guilty
$H_0$ is rejected	$H_0$ is declared guilty: there are <b>strong evidences supporting its guilt</b>
$H_0$ is not rejected	$H_0$ is declared not guilty: either is <b>innocent or there are no enough evidences supporting its guilt</b>

More formally, the  $p$ -value of an hypothesis test about  $H_0$  is defined as:

The  $p$ -value is the probability of obtaining a statistic more unfavourable to  $H_0$  than the observed, assuming that  $H_0$  is true.

Therefore, if the  **$p$ -value is small** (smaller than the chosen level  $\alpha$ ), it is **unlikely that the evidence against  $H_0$  is due to randomness**. As a consequence,  $H_0$  is rejected. If the  $p$ -value is large (larger

than  $\alpha$ ), then it is more possible that the evidences against  $H_0$  are merely due to the randomness of the data. In this case, we do not reject  $H_0$ .



If  $H_0$  holds, then the  $p$ -value (which is a random variable) is distributed uniformly in  $(0, 1)$ . If  $H_0$  does not hold, then the distribution of the  $p$ -value is not uniform but concentrated at 0 (where the rejections of  $H_0$  take place).

## Appendix C

# Use of qualitative predictors

An important situation is how to deal with *qualitative*, and not *quantitative*, predictors when we fit a regression model. Qualitative predictors, also known as *categorical* variables or, in R's terminology, *factors*, are very common, for example in social sciences. Dealing with them requires some care and proper understanding of how these variables are represented.

The simplest case is the situation with **two levels**. A binary variable  $C$  with two levels (for example,  $a$  and  $b$ ) can be represented as

$$D = \begin{cases} 1, & \text{if } C = b, \\ 0, & \text{if } C = a. \end{cases}$$

$D$  now is a *dummy variable*: it codifies with zeros and ones the two possible levels of the categorical variable. An example of  $C$  could be *gender*, which has levels *male* and *female*. The dummy variable associated is  $D = 0$  if the gender is male and  $D = 1$  if the gender is female.

The advantage of this *dummification* is its interpretability in regression models. Since level  $a$  corresponds to 0, it can be seen as the *reference level* to which level  $b$  is compared. This is the key point in dummification: **set one level as the reference and codify the rest as departures from it** with ones.

R does the dummification automatically (translates a categorical variable  $C$  into its dummy version  $D$ ) if it detects that a factor variable is present in the regression model.

Let's see now the case with **more than two levels**, for example, a categorical variable  $C$  with levels  $a$ ,  $b$ , and  $c$ . If we take  $a$  as the reference level, this variable can be represented by *two* dummy variables:

$$D_1 = \begin{cases} 1, & \text{if } C = b, \\ 0, & \text{if } C \neq b \end{cases}$$

and

$$D_2 = \begin{cases} 1, & \text{if } C = c, \\ 0, & \text{if } C \neq c. \end{cases}$$

Then  $C = a$  is represented by  $D_1 = D_2 = 0$ ,  $C = b$  is represented by  $D_1 = 1, D_2 = 0$  and  $C = c$  is represented by  $D_1 = 0, D_2 = 1$ .

In general, if we have a categorical variable with  $J$  levels, then the number of dummy variables required is  $J - 1$ . Again, R does the dummification automatically for you if it detects that a factor variable is present in the regression model.



It may happen that one dummy variable, say  $D_1$  is not significant, while other dummy variables, say  $D_2$ , are significant.



**Do not codify a categorical variable as a discrete variable.** This constitutes a major methodological fail that will flaw the subsequent statistical analysis.

For example if you have a categorical variable `party` with levels `partyA`, `partyB`, and `partyC`, do not encode it as a discrete variable taking the values 1, 2, and 3, respectively. If you do so:

- You assume implicitly an order in the levels of `party`, since `partyA` is closer to `partyB` than to `partyC`.
- You assume implicitly that `partyC` is three times larger than `partyA`.
- The codification is completely arbitrary – why not considering 1, 1.5, and 1.75 instead of?

The right way of dealing with categorical variables in regression is to set the variable as a factor and let R do internally the dummification.

## Appendix D

# Model Selection

Linear Model Selection and Best Subset Selection

Forward Stepwise Selection

Backward Stepwise Selection

Estimating Test Error Using Mallow's Cp, AIC, BIC, Adjusted R-squared

Estimating Test Error Using Cross-Validation

## Examples

Best Subset Selection

Forward Stepwise Selection and Model Selection Using Validation Set

Model Selection Using Cross-Validation



## Appendix E

## References



# Bibliography

- Dalal, S. R., Fowlkes, E. B., and Hoadley, B. (1989). Risk analysis of the space shuttle: Pre-challenger prediction of failure. *Journal of the American Statistical Association*, 84(408):945–957.
- FISHER, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.
- Presidential Commission on the Space Shuttle Challenger Accident (1986). *Report of the Presidential Commission on the Space Shuttle Challenger Accident (Vols. 1 & 2)*. Washington, DC.