

# Notes for Predictive Modeling

MSc in Big Data Analytics at Carlos III University of Madrid

*Eduardo García-Portugués*

*2018-02-20, v6.3*



# Contents

|          |   |            |
|----------|---|------------|
| <b>1</b> | <b>Introduction</b>   | <b>5</b>   |
| 1.1      | Course overview . . . . .   | 5          |
| 1.2      | Required packages . . . . .   | 5          |
| 1.3      | What is <i>predictive modeling</i> ? . . . . .                        | 6          |
| 1.4      | General notation and background . . . . .                             | 9          |
| 1.5      | Datasets for the course . . . . .                                     | 11         |
| 1.6      | Main references and credits . . . . .                                 | 13         |
| <b>2</b> | <b>Linear models I: multiple linear model</b>                         | <b>15</b>  |
| 2.1      | Case study: <i>The Bordeaux equation</i> . . . . .                    | 15         |
| 2.2      | Model formulation and least squares . . . . .                         | 17         |
| 2.3      | Assumptions of the model . . . . .                                    | 29         |
| 2.4      | Inference for model parameters . . . . .                              | 31         |
| 2.5      | Prediction . . . . .  | 39         |
| 2.6      | ANOVA . . . . .   | 42         |
| 2.7      | Model fit . . . . .   | 45         |
| <b>3</b> | <b>Linear models II: model selection, extensions, and diagnostics</b> | <b>55</b>  |
| 3.1      | Case study: <i>Housing values in Boston</i> . . . . .                 | 55         |
| 3.2      | Model selection . . . . .   | 57         |
| 3.3      | Use of qualitative predictors . . . . .                               | 67         |
| 3.4      | Nonlinear relationships . . . . .                                     | 72         |
| 3.5      | Model diagnostics . . . . .   | 92         |
| 3.6      | Dimension reduction techniques . . . . .                              | 113        |
| <b>4</b> | <b>Linear models III: shrinkage and big data</b>                      | <b>147</b> |
| 4.1      | Shrinkage . . . . .   | 147        |
| 4.2      | A note of caution with inference after model-selection . . . . .      | 169        |
| 4.3      | Big data considerations . . . . .                                     | 173        |
| <b>5</b> | <b>Generalized linear models</b>                                      | <b>183</b> |
| 5.1      | Case study: <i>The Challenger disaster</i> . . . . .                  | 183        |
| 5.2      | Model formulation and estimation . . . . .                            | 187        |
| 5.3      | Inference for model parameters . . . . .                              | 201        |
| 5.4      | Prediction . . . . .  | 206        |
| 5.5      | Deviance . . . . .  | 208        |
| 5.6      | Model selection . . . . .   | 215        |
| 5.7      | Model diagnostics . . . . .   | 219        |
| 5.8      | Shrinkage . . . . .   | 228        |
| 5.9      | Big data considerations . . . . .                                     | 238        |
| <b>6</b> | <b>Nonparametric regression</b>                                       | <b>245</b> |

|          |  |            |
|----------|--|------------|
| 6.1      | Kernel density estimation . . . . .                    | 245        |
| 6.2      | Kernel regression estimation . . . . .                 | 249        |
| 6.3      | Asymptotic properties . . . . .                        | 255        |
| 6.4      | Bandwidth selection . . . . .                          | 256        |
| 6.5      | Local likelihood . . . . .                             | 259        |
| <b>A</b> | <b>Installation of R and RStudio</b>                   | <b>265</b> |
| <b>B</b> | <b>Introduction to RStudio</b>                         | <b>267</b> |
| <b>C</b> | <b>Introduction to R</b>                               | <b>269</b> |
| <b>D</b> | <b>Review on hypothesis testing</b>                    | <b>297</b> |
| <b>E</b> | <b>Least squares and maximum likelihood estimation</b> | <b>299</b> |
| <b>F</b> | <b>Dealing with missing data</b>                       | <b>301</b> |
| <b>G</b> | <b>Multinomial logistic regression</b>                 | <b>311</b> |
| <b>H</b> | <b>Course evaluation</b>                               | <b>315</b> |

# Chapter 1

## Introduction



The **animations** of these notes will not be displayed the first time they are browsed<sup>1</sup>. See for example Figure 1.3. **To see them**, click on the caption's link “*Application also available here*”. 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 (I guarantee you I will not do anything evil!). The next time the animation will show up correctly within the notes.

### 1.1 Course overview

Welcome to the notes for *Predictive Modeling*. These notes contain both the theory and practice for the statistical methods presented in the course. The emphasis is placed in building intuition behind the methods, gaining insights into their properties, and showing their application through the use of statistical software. The topics we will cover are an in-depth analysis of **linear models** (I, II, and III), their extension to **generalized linear models**, and an introduction to **nonparametric regression**.

The notes contain a substantial amount of snippets of code that are fully self-contained within the chapter in which they are included. This allows to see how the methods and theory translate neatly to the practice. The **software** employed in the course is *the statistical language R* and its most common IDE (Integrated Development Environment) nowadays, **RStudio**. A basic prior knowledge of both is assumed<sup>2</sup>. The appendix presents some basic introductions to **RStudio** and **R** for those students lacking basic expertise on them.

The **Shiny interactive apps** on the notes can be downloaded and run locally, which allows in particular to examine their codes. Check this GitHub repository for the sources.

### 1.2 Required packages

We will employ several packages that are not contained in **R** (*note: list to be updated*). These can be installed as:

```
# Installation of required packages
install.packages(c("MASS", "car", "readxl", "rgl", "nortest", "pca3d",
                  "ISLR", "pls", "glmnet", "biglm", "leaps", "viridis",
                  "ffbase", "KernSmooth", "np", "locfit", "manipulate",
                  "mice", "VIM", "nnet", "lubridate"))
```

<sup>2</sup>Among others: basic programming in **R**, ability to work with objects and data structures, ability to produce graphics, knowledge of the main statistical functions, and ability to run scripts in **RStudio**.

The codes in the notes may assume that the packages have been loaded, so it is better to do it now (*note: list to be updated*):

```
# Load packages
library(MASS)
library(car)
library(readxl)
library(rgl)
library(nortest)
library(pca3d)
library(ISLR)
library(pls)
library(glmnet)
library(biglm)
library(leaps)
library(viridis)
library(fffbase)
library(KernSmooth)
library(np)
library(locfit)
library(mice)
library(VIM)
library(nnet)
```

### 1.3 What is *predictive modeling*?

Predictive modeling is the process of developing a mathematical tool or model that generates an accurate prediction about a random quantity of interest.

In predictive modeling we are interested in predicting a random variable, typically denoted by  $Y$ , from a set of related variables  $X_1, \dots, X_p$ . The focus is on learning what is the probabilistic model that relates  $Y$  with  $X_1, \dots, X_p$ , and use that acquired knowledge for predicting  $Y$  given an observation of  $X_1, \dots, X_p$ . Some concrete examples of this are:

- Predicting the wine quality ( $Y$ ) from a set of environmental variables ( $X_1, \dots, X_p$ ).
- Predicting the number of sales ( $Y$ ) from a set of marketing actions ( $X_1, \dots, X_p$ ).
- Modelling the average house value in a given suburb ( $Y$ ) from a set of community-related features ( $X_1, \dots, X_p$ ).
- Predicting the probability of failure ( $Y$ ) of a rocket launcher from the ambient temperature ( $X_1$ ).

The process of predictive modelling can be statistically abstracted in the following way. We believe that  $Y$  and  $X_1, \dots, X_p$  are related by a *regression model* of the form

$$Y = m(X_1, \dots, X_p) + \varepsilon, \quad (1.1)$$

where  $m$  is the *regression function* and  $\varepsilon$  is a random error (which accounts for the uncertainty of knowing  $Y$  if  $X_1, \dots, X_p$  are known) with zero mean. The function  $m : \mathbb{R}^p \rightarrow \mathbb{R}$  is unknown in practice and is the target of predictive modelling:  $m$  **encodes the relation<sup>3</sup> between  $Y$  and  $X_1, \dots, X_p$** .  $m$  captures the *trend* of the relation between  $Y$  and  $X_1, \dots, X_p$ , and  $\varepsilon$  the *stochasticity* of that relation.

Therefore, knowing  $m$  allows to predict  $Y$ . We are going to devote this course to see some statistical models to come up with an estimate of  $m$ , denoted by  $\hat{m}$ , and use  $\hat{m}$  to predict  $Y$ .

---

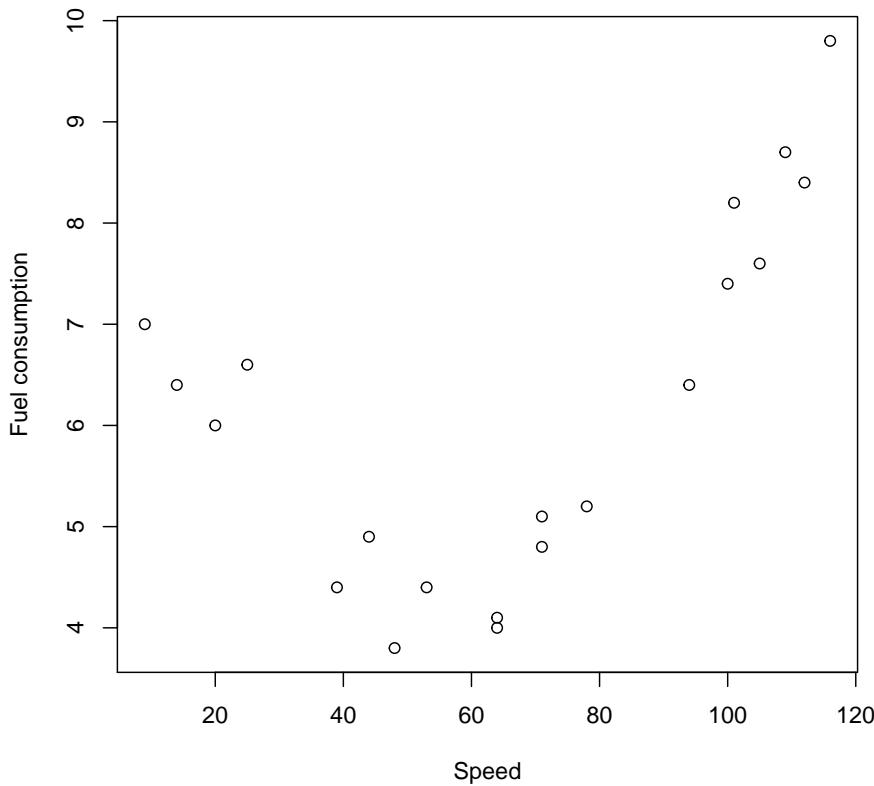
<sup>3</sup>The relation is encoded in *average* by means of the *conditional expectation*.

Let's see a concrete example of this with an artificial dataset. Suppose  $Y$  represents average fuel consumption (l/100km) of a car and  $X$  is the average speed (km/h). It is well known from physics that the energy and speed have a quadratic relationship, and therefore we may assume that  $Y$  and  $X$  are *truly* quadratically-related for the sake of exposition:

$$Y = a + bX + cX^2 + \varepsilon.$$

Then  $m : \mathbb{R} \rightarrow \mathbb{R}$  ( $p = 1$ ) with  $m(x) = a + bx + cx^2$ . Suppose the following data are measurements from a given car model, measured in different drivers and conditions (we do not have data for accounting for all those effects, which go to the  $\varepsilon$  term):

```
x <- c(64, 20, 14, 64, 44, 39, 25, 53, 48, 9,
      100, 112, 78, 105, 116, 94, 71, 71, 101, 109)
y <- c(4, 6, 6.4, 4.1, 4.9, 4.4, 6.6, 4.4, 3.8, 7,
      7.4, 8.4, 5.2, 7.6, 9.8, 6.4, 5.1, 4.8, 8.2, 8.7)
plot(x, y, xlab = "Speed", ylab = "Fuel consumption")
```



From this data, we can estimate  $m$  by means of a polynomial model<sup>4</sup>:

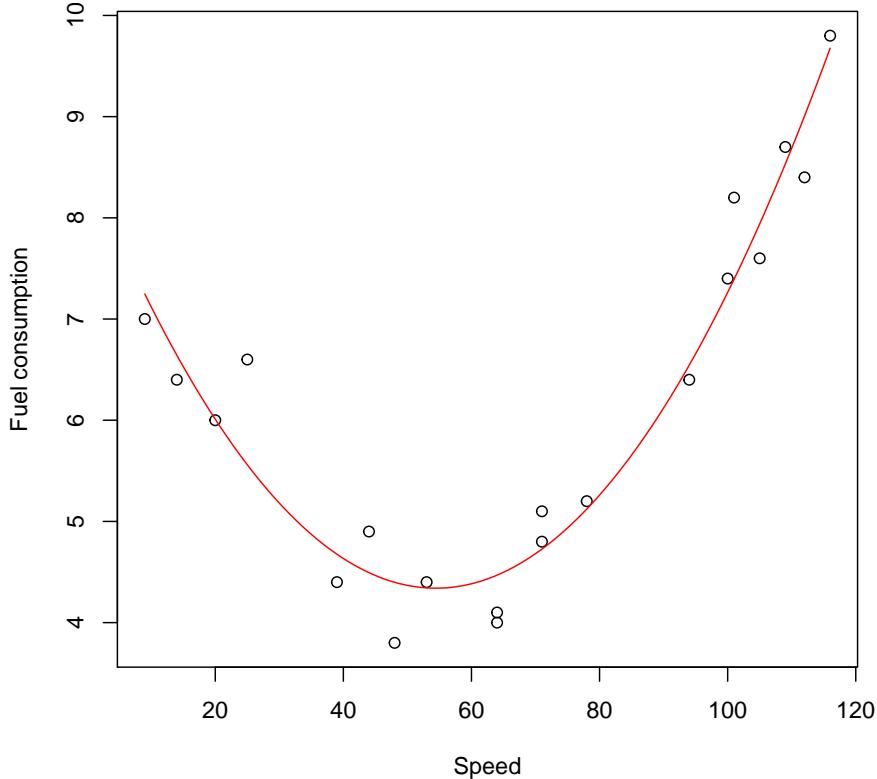
```
# Estimates for a, b, and c
lm(y ~ x + I(x^2))
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2))
##
## Coefficients:
## (Intercept)          x      I(x^2)
##     8.512421     -0.153291      0.001408
```

<sup>4</sup>Note we use the information that  $m$  has to be of a particular form (in this case quadratic) which is an unrealistic situation for other data applications.

Then the estimate of  $m$  is  $\hat{m}(x) = \hat{a} + \hat{b}x + \hat{c}x^2 = 8.512 - 0.153x + 0.001x^2$  and its fit to the data is pretty good. As a consequence, we can use this precise mathematical function to predict the  $Y$  from a particular observation  $X$ . For example, the average fuel consumption of at a speed 90 km/h is  $8.512421 - 0.153291 * 90 + 0.001408 * 90^2 = 6.1210$ .

```
plot(x, y, xlab = "Speed", ylab = "Fuel consumption")
curve(8.512421 - 0.153291 * x + 0.001408 * x^2, add = TRUE, col = 2)
```



There are a number of generic issues and decisions to take when building and estimating regression models that are worth to highlight:

- The **prediction accuracy versus interpretability** trade-off. *Prediction accuracy* is key in any predictive model. However, some of them achieve this predictive accuracy in exchange of a clear *interpretability* of the model (the so-called *black boxes*). Interpretability is key in order to gain insights on the prediction process, to know which variables are most influential in  $Y$ , to be able to interpret the parameters of the model, and to translate the prediction process to non-experts. The models covered in these notes are clearly interpretable and hence make a sacrifice in terms of prediction accuracy at an exchange of a neat interpretability.
- **Model correctness versus model usefulness.** Correctness and usefulness are two very different concepts in modeling. The first refers to the model being *statistically* correct, this is, if the **assumptions** on which the model between  $Y$  and  $X_1, \dots, X_p$  is built are satisfied. The second refers to the model being **useful for explaining or predicting**  $Y$  from  $X_1, \dots, X_p$ . Both dimensions are related (if the model is correct/useful, then *likely* it is useful/correct) but neither is implied by the other. For example, a regression model might be correct but useless if the variance of  $\varepsilon$  is large (too much noise). And yet if the model is not completely correct, it may give useful insights and predictions (but inference will be problematic!).
- **Flexibility versus simplicity.** *The best model* is the one which is very simple (low number of parameters), highly interpretable, and delivers great predictions. This is often unachievable in practice. What it can be achieved is a *good model*: the one that balances the simplicity with the prediction

accuracy, which is often increased the more flexible the model is. However, **flexibility comes at a price**: more flexible (hence more complex) models use more parameters that need to be estimated from a finite amount of information – the sample. This is problematic, as overly flexible models are more dependent on the sample of  $Y$  and  $X_1, \dots, X_p$ , and therefore do not estimate the true relation between  $Y$  and  $X_1, \dots, X_p$ . This phenomenon is called **over-fitting** and it can be avoided by splitting the dataset in two parts: the *training dataset* (used for estimating the model) and the *testing dataset* (used for evaluating its predictive performance). On the other hand, excessive simplicity (**under-fitting**) is also problematic, since the true relation between  $Y$  and  $X_1, \dots, X_p$  may not be that simple. Therefore, a trade-off in the degree of flexibility has to be attained for having a good model. This is often referred as the **bias-variance trade-off** (low flexibility increases the bias of the fitted model, high flexibility increases the variance). An illustration of this point is given in Figure 1.3.

Illustration of over-fitting in polynomial regression. The left plot shows the training dataset and the right plot the testing dataset. Better fitting of the training data with a higher polynomial order does not imply better performance in new observations (prediction), but just an over-fitting of the available data with an overly-parametrized model (too flexible for the amount of information available). Reduction in the predictive error is only achieved with fits (in red) of polynomial degrees close to the true regression (in black). Application also available here.

## 1.4 General notation and background

We use capital letters to denote *random variables*, such as  $X$ , and lowercase, such as  $x$ , to denote deterministic values. For example  $\mathbb{P}[X = x]$  means “the probability that the random variable  $X$  takes the particular value  $x$ ”. In predictive modelling we are concerned about the prediction or explanation of a *response*  $Y$  from a set of *predictors*  $X_1, \dots, X_p$ . Both  $Y$  and  $X_1, \dots, X_p$  are random variables, but we use them in a different way: our interest lays in prediction or explaining  $Y$  from  $X_1, \dots, X_p$ . Other name for  $Y$  is *dependent variable* and  $X_1, \dots, X_p$  are sometimes referred as *independent variables*, *covariates*, or *explanatory variables*. We will not use these terminologies.

The *cumulative distribution function* (cdf) of a random variable  $X$  is  $F(x) := \mathbb{P}[X \leq x]$  and is a function that completely characterizes the randomness of  $X$ . Continuous random variables are also characterized by the *probability density function* (pdf)  $f(x) = F'(x)$  (respectively,  $F(x) = \int_{-\infty}^x f(t)dt$ ), which represents the *infinitesimal relative probability* of  $X$  per unit of length. On the other hand, discrete random variables are also characterized by the *probability mass function*  $\mathbb{P}[X = x]$ . We write  $X \sim F$  (or  $X \sim f$  if  $X$  is continuous) to denote that  $X$  has a cdf  $F$  (or a pdf  $f$ ). If two random variables  $X$  and  $Y$  have the same distribution, we write  $X \stackrel{d}{=} Y$ .

For a random variable  $X \sim F$ , the *expectation* of  $g(X)$  is

$$\begin{aligned}\mathbb{E}[g(X)] &:= \int g(x)dF(x) \\ &:= \begin{cases} \int g(x)f(x)dx, & \text{if } X \text{ is continuous,} \\ \sum_{\{i: \mathbb{P}[X=x_i]>0\}} g(x_i)\mathbb{P}[X=x_i], & \text{if } X \text{ is discrete.} \end{cases}\end{aligned}$$

Unless otherwise stated, the integration limits of any integral are  $\mathbb{R}$  or  $\mathbb{R}^p$ . The *variance* is defined as  $\text{Var}[X] := \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ .

We employ boldface to denote vectors (assumed to be column matrices, although sometimes written in row-layout), like  $\mathbf{a}$ , and matrices, like  $\mathbf{A}$ . We denote by  $\mathbf{A}'$  to the transpose of  $\mathbf{A}$ . Boldfaced capitals will be used simultaneously for denoting matrices and also *random vectors*  $\mathbf{X} = (X_1, \dots, X_p)$ , which are collections random variables  $X_1, \dots, X_p$ . The (joint) cdf of  $\mathbf{X}$  is

$$F(\mathbf{x}) := \mathbb{P}[\mathbf{X} \leq \mathbf{x}] := \mathbb{P}[X_1 \leq x_1, \dots, X_p \leq x_p]$$

(understood as the probability that  $X_1 \leq x_1$  and  $\dots$  and  $X_p \leq x_p$ ) and, if  $\mathbf{X}$  is continuous, its (joint) pdf is  $f := \frac{\partial^p}{\partial x_1 \dots \partial x_p} F$ . The *marginals* of  $F$  and  $f$  are the cdf and pdf of  $X_j$ ,  $j = 1, \dots, p$ , respectively. They are defined as:

$$F_{X_j}(x_j) := \mathbb{P}[X_j \leq x] = \int_{\mathbb{R}^{p-1}} F(\mathbf{x}) d\mathbf{x}_{-i},$$

$$f_{X_j}(x_j) := \frac{\partial}{\partial x_j} F_{X_j}(x_j) = \int_{\mathbb{R}^{p-1}} f(\mathbf{x}) d\mathbf{x}_{-i},$$

where  $\mathbf{x}_{-i} := (x_1, \dots, x_{i-1}, x_{i+1}, x_p)$ . The definitions can be extended analogously to the marginals of the cdf and pdf of different subsets of  $\mathbf{X}$ .

The *conditional* cdf and pdf of  $X_1 | (X_2, \dots, X_p)$  are defined, respectively, as

$$F_{X_1 | \mathbf{X}_{-1} = \mathbf{x}_{-1}}(x_1) := \mathbb{P}[X_1 \leq x_1 | \mathbf{X}_{-1} = \mathbf{x}_{-1}],$$

$$f_{X_1 | \mathbf{X}_{-1} = \mathbf{x}_{-1}}(x_1) := \frac{f(\mathbf{x})}{f_{\mathbf{X}_{-1}}(\mathbf{x}_{-1})}.$$

The *conditional expectation* of  $Y | X$  is the following random variable<sup>5</sup>

$$\mathbb{E}[Y | X] := \int y dF_{Y|X}(y | X).$$

For two random variables  $X_1$  and  $X_2$ , the *covariance* between them is defined as

$$\text{Cov}[X_1, X_2] := \mathbb{E}[(X_1 - \mathbb{E}[X_1])(X_2 - \mathbb{E}[X_2])] = \mathbb{E}[X_1 X_2] - \mathbb{E}[X_1] \mathbb{E}[X_2],$$

and the *correlation* between them is defined as

$$\text{Cor}[X_1, X_2] := \frac{\text{Cov}[X_1, X_2]}{\sqrt{\text{Var}[X_1] \text{Var}[X_2]}}.$$

The variance and covariance are extended to a random vector  $\mathbf{X} = (X_1, \dots, X_p)'$  by means of the so-called *variance-covariance matrix*:

$$\begin{aligned} \text{Var}[\mathbf{X}] &:= \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])'] = \mathbb{E}[\mathbf{X}\mathbf{X}'] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]' \\ &= \begin{pmatrix} \text{Var}[X_1] & \text{Cov}[X_1, X_2] & \dots & \text{Cov}[X_1, X_p] \\ \text{Cov}[X_2, X_1] & \text{Var}[X_2] & \dots & \text{Cov}[X_2, X_p] \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}[X_p, X_1] & \text{Cov}[X_p, X_2] & \dots & \text{Var}[X_p] \end{pmatrix}, \end{aligned}$$

where  $\mathbb{E}[\mathbf{X}] := (\mathbb{E}[X_1], \dots, \mathbb{E}[X_p])'$ . As in the univariate case, the expectation is a linear operator, which means that

$$\mathbb{E}[\mathbf{A}\mathbf{X}\mathbf{B} + \mathbf{C}] = \mathbf{A}\mathbb{E}[\mathbf{X}]\mathbf{B} + \mathbf{C}, \quad \text{for matrices } \mathbf{A}, \mathbf{B}, \text{ and } \mathbf{C}. \quad (1.2)$$

---

<sup>5</sup>Recall that the  $X$ -part is random!

The  $p$ -dimensional normal of mean  $\boldsymbol{\mu} \in \mathbb{R}^p$  and covariance matrix  $\boldsymbol{\Sigma}$  (a  $p \times p$  symmetric and positive definite matrix) is denoted by  $\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  and is the generalization to  $p$  random variables of usual the *normal* distribution. Its (joint) pdf is given by

$$\phi(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) := \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}, \quad \mathbf{x} \in \mathbb{R}^p.$$

Notice that when  $p = 1$ , and  $\boldsymbol{\mu} = \mu$  and  $\boldsymbol{\Sigma} = \sigma^2$ , then the pdf of the usual normal is recovered:  $\phi(x; \mu, \sigma) := \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ . When  $p = 2$ , the pdf is expressed in terms of  $\boldsymbol{\mu} = (\mu_1, \mu_2)$  and  $\boldsymbol{\Sigma} = (\sigma_1^2, \rho\sigma_1\sigma_2; \rho\sigma_1\sigma_2, \sigma_2^2)$ , for  $\mu_1, \mu_2 \in \mathbb{R}$ ,  $\sigma_1, \sigma_2 > 0$ , and  $-1 < \rho < 1$ :

$$\phi(x_1, x_2; \mu_1, \mu_2, \sigma_1, \sigma_2, \rho) := \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} e^{-\frac{1}{2(1-\rho^2)} \left[ \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \right]}. \quad (1.3)$$

The surface defined by (1.3) that can be regarded as a bell (in three dimensions). In addition, it serves to provide concrete examples of the functions introduced above:

- Joint pdf:  $f(x_1, x_2) = \phi(x_1, x_2; \mu_1, \mu_2, \sigma_1, \sigma_2, \rho)$ .
- Marginal pdfs:  $f_{X_1}(x_1) = \int \phi(x_1, t_2; \mu_1, \mu_2, \sigma_1, \sigma_2, \rho) dt_2 = \phi(x_1; \mu_1, \sigma_1)$  and  $\phi(x_2; \mu_2, \sigma_2)$ . Hence  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2)$ .
- Conditional pdfs:  $f_{X_1|X_2=x_2}(x_1) = \frac{f(x_1, x_2)}{f_{X_2}(x_2)} = \phi\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1 - \rho^2)\sigma_1^2\right)$  and  $\phi\left(x_2; \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1), (1 - \rho^2)\sigma_2^2\right)$ . Hence

$$X_1|X_2=x_2 \sim \mathcal{N}\left(\mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1 - \rho^2)\sigma_1^2\right),$$

$$X_2|X_1=x_1 \sim \mathcal{N}\left(\mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1), (1 - \rho^2)\sigma_2^2\right).$$

- Conditional expectations:  $\mathbb{E}[X_1|X_2=x_2] = \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2)$  and  $\mathbb{E}[X_2|X_1=x_1] = \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1)$ .
- Joint cdf:  $\int_{-\infty}^{x_2} \int_{-\infty}^{x_1} \phi(t_1, t_2; \mu_1, \mu_2, \sigma_1, \sigma_2, \rho) dt_1 dt_2$ .
- Marginal cdfs:  $\int_{-\infty}^{x_1} \phi(t; \mu_1, \sigma_1) dt =: \Phi(x_1; \mu_1, \sigma_1)$  and  $\Phi(x_2; \mu_2, \sigma_2)$ .
- Conditional cdfs:  $\int_{-\infty}^{x_1} \phi(t; \mu_1, \sigma_1) dt = \Phi\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1 - \rho^2)\sigma_1^2\right)$  and  $\Phi\left(x_2; \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1), (1 - \rho^2)\sigma_2^2\right)$ .

In the predictive models we will consider an *independent and identically distributed* (iid) sample of the response and the predictors. We use the following notation:  $Y_i$  is the  $i$ -th observation of the response  $Y$  and  $X_{ij}$  represents the  $i$ -th observation of the  $j$ -th predictor  $X_j$ .

## 1.5 Datasets for the course

This is a (constantly updated) handy list of all the relevant datasets used in the course. To download them, simply **save the link as a file** in your browser.

- **wine.csv** (download). The dataset is formed by the auction Price of 27 red Bordeaux vintages, five vintage descriptors (`WinterRain`, `AGST`, `HarvestRain`, `Age`, `Year`), and the population of France in the year of the vintage, `FrancePop`.
- **least-squares.RData** (download). Contains a single `data.frame`, named `leastSquares`, with 50 observations of the variables `x`, `yLin`, `yQua`, and `yExp`. These are generated as  $X \sim \mathcal{N}(0, 1)$ ,  $Y_{\text{lin}} = -0.5 + 1.5X + \varepsilon$ ,  $Y_{\text{qua}} = -0.5 + 1.5X^2 + \varepsilon$ , and  $Y_{\text{exp}} = -0.5 + 1.5 \cdot 2^X + \varepsilon$ , with  $\varepsilon \sim \mathcal{N}(0, 0.5^2)$ . The purpose of the dataset is to illustrate the least squares fitting.

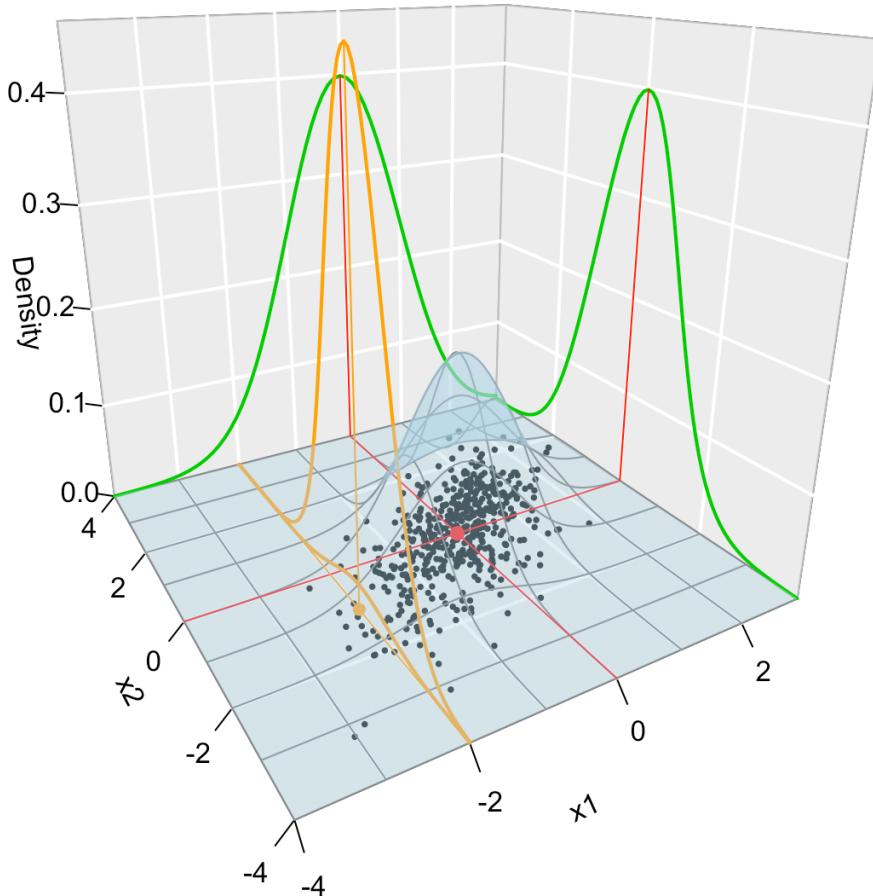


Figure 1.1: Visualization of the joint pdf (in blue), marginal pdfs (green), conditional pdf of  $X_2|X_1 = x_1$  (orange), expectation (red point), and conditional expectation  $\mathbb{E}[X_2|X_1 = x_1]$  (orange point) of a 2-dimensional normal. The conditioning point of  $X_1$  is  $x_1 = -2$ . Note the different scales of the densities, as they have to integrate one over different supports. Note how the conditional density (upper orange curve) is *not* the joint pdf  $f(2, x_2)$  (lower orange curve) but a rescaling of this curve by  $\frac{1}{f_{X_1}(x_1)}$ . The parameters of the 2-dimensional normal are  $\mu_1 = \mu_2 = 0$ ,  $\sigma_1 = \sigma_2 = 1$  and  $\rho = 0.75$ . 500 observations sampled from the distribution are shown in black.

- **least-squares-3D.RData** (download). Contains a single `data.frame`, named `leastSquares3D`, with 50 observations of the variables `x1`, `x2`, `x3`, `yLin`, `yQua`, and `yExp`. These are generated as  $X_1, X_2 \sim \mathcal{N}(0, 1)$ ,  $X_3 = X_1 + \mathcal{N}(0, 0.05^2)$ ,  $Y_{\text{lin}} = -0.5 + 0.5X_1 + 0.5X_2 + \varepsilon$ ,  $Y_{\text{qua}} = -0.5 + X_1^2 + 0.5X_2 + \varepsilon$ , and  $Y_{\text{exp}} = -0.5 + 0.5e^{X_2} + X_3 + \varepsilon$ , with  $\varepsilon \sim \mathcal{N}(0, 1)$ . The purpose of the dataset is to illustrate the least squares fitting with several predictors.
- **assumptions.RData** (download). Contains the data frame `assumptions` with 200 observations of the variables `x1`, ..., `x9` and `y1`, ..., `y9`. The purpose of the dataset is to identify which regression `y1 ~ x1, ..., y9 ~ x9` fulfills the assumptions of the linear model. The dataset `moreAssumptions.RData` (download) has the same structure.
- **assumptions3D.RData** (download). Contains the data frame `assumptions3D` with 200 observations of the variables `x1.1`, ..., `x1.8`, `x2.1`, ..., `x2.8` and `y.1`, ..., `y.8`. The purpose of the dataset is to identify which regression `y.1 ~ x1.1 + x2.1, ..., y.8 ~ x1.8 + x2.8` fulfills the assumptions of the linear model.
- **Boston.xlsx** (download). The dataset contains 14 variables describing 506 suburbs in Boston. Among those variables, `medv` is the median house value, `rm` is the average number of rooms per house and `crim`

is the per capita crime rate. The full description is available in [?Boston](#).

- [cpus.txt](#) (download) and [gpus.txt](#) (download). The datasets contain 102 and 35 rows, respectively, of commercial CPUs and GPUs appeared since the first models up to nowadays. The variables in the datasets are `Processor`, `Transistor count`, `Date of introduction`, `Manufacturer`, `Process`, and `Area`.
- [la-liga-2015-2016.xlsx](#) (download). Contains 19 performance metrics for the 20 football teams in La Liga 2015/2016.
- [challenger.txt](#) (download). Contains data for 23 space-shuttle launches. There are 8 variables. Among them: `temp` (the temperature in Celsius degrees at the time of launch), and `fail.field` and `fail.nozzle` (indicators of whether there were an incidents in the O-rings of the field joints and nozzles of the solid rocket boosters).
- [species.txt](#) (download). Contains data for 90 country parcels in which the `Biomass`, `pH` of the terrain (categorical variable), and number of `Species` were measured.
- [Chile.txt](#) (download). Contains data for 2700 respondents on a survey for the voting intentions in the 1988 Chilean national plebiscite. There are 8 variables: `region`, `population`, `sex`, `age`, `education`, `income`, `statusquo` (scale of support for the status quo), and `vote`. `vote` is a factor with levels A (abstention), N (against Pinochet), U (undecided), and Y (for Pinochet). Available in R through the package `car` and `data(Chile)`.

## 1.6 Main references and credits

Several great reference books have been used for preparing these notes. The following list details the sections in which each of them has been consulted (*note: list to be updated and to be more detailed*):

- Durbán (2017)
- Kuhn and Johnson (2013)
- James et al. (2013)
- Fan and Gijbels (1996)
- McCullagh and Nelder (1983)
- Peña (2002)
- Loader (1999)
- Wood (2006)
- Wand and Jones (1995)
- Wasserman (2004)
- Wasserman (2006)

In addition, these notes are possible due to the existence of these incredible pieces of software: Xie (2016a), Xie (2016b), Allaire et al. (2017), and R Core Team (2017).

The notes have benefited notably from contributions from the following people:

- Katherine Botz (performed a major proofreading of the course materials)
- José Ángel Fernández (provided fixes for several typos)
- Gulgur Demir (indicated a couple of typos)

The icons used in the notes were designed by madebyoliver, freepik, and roundicons from Flaticon.

All material in these notes is licensed under CC BY-NC-SA 4.0.



# Chapter 2

## Linear models I: multiple linear model

The multiple linear model is a simple but useful statistical model. In short, it allows us to analyse the (assumed) linear relation between a response  $Y$  and *multiple* predictors,  $X_1, \dots, X_p$  in a proper way:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

The simplest case corresponds to  $p = 1$ , known as the *simple* linear model:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

This model would be useful, for example, to predict  $Y$  given  $X$  from a sample  $(X_1, Y_1), \dots, (X_n, Y_n)$  such that its scatterplot is the one in Figure 2.1.

### 2.1 Case study: *The Bordeaux equation*

ABC interview to Orley Ashenfelter, broadcasted in 1992.

Calculate the winter rain and the harvest rain (in millimeters). Add summer heat in the vineyard (in degrees centigrade). Subtract 12.145. And what do you have? A very, very passionate argument over wine.

— “Wine Equation Puts Some Noses Out of Joint”, The New York Times, 04/03/1990

This case study is motivated by the study of Princeton professor Orley Ashenfelter (Ashenfelter et al., 1995) on the quality of red Bordeaux vintages. The study became mainstream after disputes with the wine press, especially with Robert Parker, Jr., one of the most influential wine critics in America. See a short review of the story at the Financial Times (Google’s cache) and at the video in Figure 2.1.

Red Bordeaux wines have been produced in Bordeaux, one of most famous and prolific wine regions in the world, in a very similar way for hundreds of years. However, *the quality of vintages is largely variable* from one season to another due to a long list of random factors, such as the weather conditions. Because Bordeaux wines taste better when they are older (young wines are astringent, when the wines age they lose their astringency), there is an incentive to store the young wines until they are mature. Due to the important difference in taste, it is hard to determine the quality of the wine when it is so young just by tasting it, because it is going to change substantially when the aged wine is in the market. Therefore, being able to *predict the quality of a vintage* is valuable information for investing resources, for determining a fair price for vintages, and for understanding what factors are affecting the wine quality. The purpose of this case study is to answer:

- Q1. *Can we predict the quality of a vintage effectively?*
- Q2. *What is the interpretation of such prediction?*

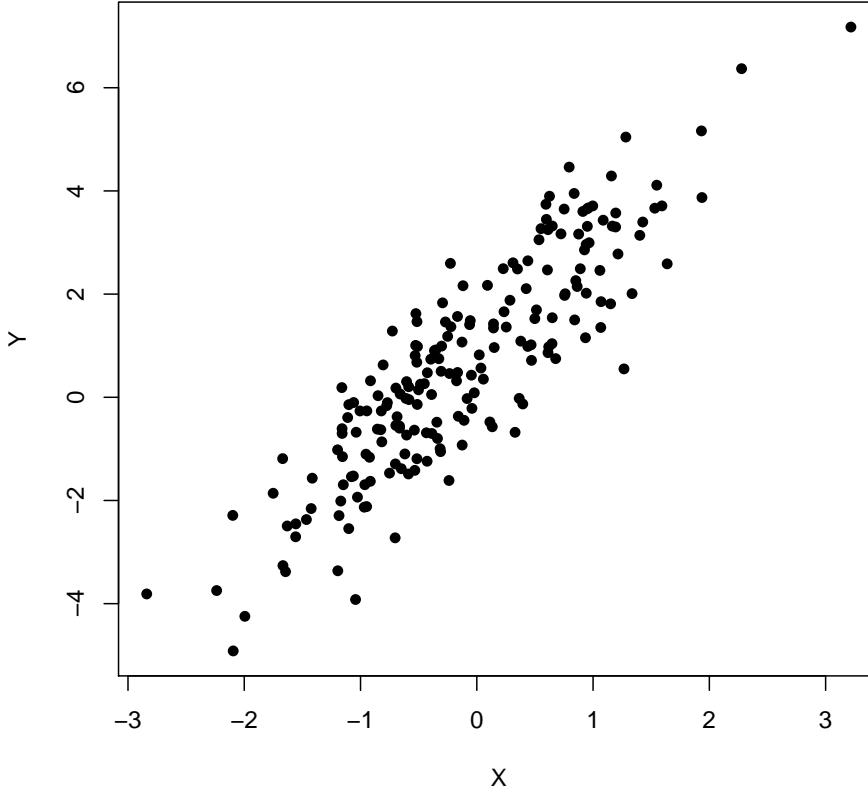


Figure 2.1: Scatterplot of a sample  $(X_1, Y_1), \dots, (X_n, Y_n)$  showing a linear pattern.

The `wine.csv` file (download) contains 27 red Bordeaux vintages. The data is the same data originally employed by Ashenfelter et al. (1995), except for the inclusion of the variable `Year`, the exclusion of NAs and the reference price used for the wine. The original source is here. Each row has the following variables:

- `Year`: year in which grapes were harvested to make wine.
- `Price`: *logarithm* of the average market price for Bordeaux vintages according to 1990–1991 auctions. The price is relative to the price of the 1961 vintage, regarded as the best one ever recorded.
- `WinterRain`: winter rainfall (in mm).
- `AGST`: Average Growing Season Temperature (in Celsius degrees).
- `HarvestRain`: harvest rainfall (in mm).
- `Age`: age of the wine measured as the number of years stored in a cask.
- `FrancePop`: population of France at `Year` (in thousands).

The *quality* of the wine is quantified as the `Price`, a clever way of quantifying a qualitative measure. The data is shown in Table 2.1.

Table 2.1: First 15 rows of the `wine` dataset.

| Year | Price  | WinterRain | AGST    | HarvestRain | Age | FrancePop |
|------|--------|------------|---------|-------------|-----|-----------|
| 1952 | 7.4950 | 600        | 17.1167 | 160         | 31  | 43183.57  |
| 1953 | 8.0393 | 690        | 16.7333 | 80          | 30  | 43495.03  |
| 1955 | 7.6858 | 502        | 17.1500 | 130         | 28  | 44217.86  |
| 1957 | 6.9845 | 420        | 16.1333 | 110         | 26  | 45152.25  |
| 1958 | 6.7772 | 582        | 16.4167 | 187         | 25  | 45653.81  |
| 1959 | 8.0757 | 485        | 17.4833 | 187         | 24  | 46128.64  |
| 1960 | 6.5188 | 763        | 16.4167 | 290         | 23  | 46584.00  |

|      |        |     |         |     |    |          |
|------|--------|-----|---------|-----|----|----------|
| 1961 | 8.4937 | 830 | 17.3333 | 38  | 22 | 47128.00 |
| 1962 | 7.3880 | 697 | 16.3000 | 52  | 21 | 48088.67 |
| 1963 | 6.7127 | 608 | 15.7167 | 155 | 20 | 48798.99 |
| 1964 | 7.3094 | 402 | 17.2667 | 96  | 19 | 49356.94 |
| 1965 | 6.2518 | 602 | 15.3667 | 267 | 18 | 49801.82 |
| 1966 | 7.7443 | 819 | 16.5333 | 86  | 17 | 50254.97 |
| 1967 | 6.8398 | 714 | 16.2333 | 118 | 16 | 50650.41 |
| 1968 | 6.2435 | 610 | 16.2000 | 292 | 15 | 51034.41 |

We will see along the chapter how to answer Q1 and Q2 and how to obtain quantitative insights on the effects of the predictors on the price.

## 2.2 Model formulation and least squares

In order to simplify the introduction of the foundations of the linear model, we first present the simple linear model and then we extend to the multiple case.

### 2.2.1 Simple linear model

The simple linear model is *constructed by assuming* that the linear relation

$$Y = \beta_0 + \beta_1 X + \varepsilon \quad (2.1)$$

holds between  $X$  and  $Y$ . In (2.1),  $\beta_0$  and  $\beta_1$  are known as the *intercept* and *slope*, respectively.  $\varepsilon$  is a random variable with mean zero and independent from  $X$ . It describes the *error* around the mean, or the effect of other variables that we do not model. Another way of looking at (2.1) is

$$\mathbb{E}[Y|X = x] = \beta_0 + \beta_1 x, \quad (2.2)$$

since  $\mathbb{E}[\varepsilon|X = x] = 0$ .

The Left Hand Side (LHS) of (2.2) is the *conditional expectation* of  $Y$  given  $X$ . It represents how the mean of the random variable  $Y$  is changing according to a particular value  $x$  of the random variable  $X$ . With the RHS, what we are saying is that the mean of  $Y$  is changing in a *linear* fashion with respect to the value of  $X$ . Hence the **interpretation of the coefficients**:

- $\beta_0$ : is the mean of  $Y$  when  $X = 0$ .
- $\beta_1$ : is the increment in mean of  $Y$  for an increment of one unit in  $X = x$ .

If we have a sample  $(X_1, Y_1), \dots, (X_n, Y_n)$  for our random variables  $X$  and  $Y$ , we can estimate the *unknown* coefficients  $\beta_0$  and  $\beta_1$ . A possible way of estimating  $(\beta_0, \beta_1)$  is by looking for certain optimality, for example the minimization of the *Residual Sum of Squares* (RSS):

$$\text{RSS}(\beta_0, \beta_1) := \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_i)^2.$$

In other words, we look for the estimators  $(\hat{\beta}_0, \hat{\beta}_1)$  such that

$$(\hat{\beta}_0, \hat{\beta}_1) = \arg \min_{(\beta_0, \beta_1) \in \mathbb{R}^2} \text{RSS}(\beta_0, \beta_1).$$

The motivation for minimizing the RSS is geometrical, as shown by Figure 2.2.1. We look to minimize the squares of the distances of points projected vertically onto the line determined by  $(\hat{\beta}_0, \hat{\beta}_1)$ .

The effect of the kind of distance in the error criterion. The choices of intercept and slope that minimize the sum of squared distances for one kind of distance are not the optimal choices for a different kind of distance. Application also available here.

It can be seen that *the minimizers of the RSS*<sup>1</sup> are

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}, \quad \hat{\beta}_1 = \frac{s_{xy}}{s_x^2}, \quad (2.3)$$

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  $r_{xy} = \frac{s_{xy}}{s_x s_y}$ .

There are some important points hidden behind the election of RSS as the error criterion for obtaining  $(\hat{\beta}_0, \hat{\beta}_1)$ :

- *Why the vertical distances and not horizontal or perpendicular?* Because we want to minimize the error in the *prediction* of  $Y$ ! Note that the treatment of the variables is *not symmetrical*.
- *Why the squares in the distances and not the absolute value?* Due to mathematical convenience. Squares are nice to differentiate and are closely related with maximum likelihood estimation under the normal distribution (see Appendix E).

Let's see how to obtain automatically the minimizers of the error in Figure 2.2.1 by the `lm` (linear model) function. The data of the figure has been generated with the following code:

```
# Generates 50 points from a N(0, 1): predictor and error
set.seed(34567)
x <- rnorm(n = 50)
eps <- rnorm(n = 50)

# Responses
yLin <- -0.5 + 1.5 * x + eps
yQua <- -0.5 + 1.5 * x^2 + eps
yExp <- -0.5 + 1.5 * 2^x + eps

# Data
leastSquares <- data.frame(x = x, yLin = yLin, yQua = yQua, yExp = yExp)
```

For a simple linear model, `lm` has the syntax `lm(formula = response ~ predictor, data = data)`, where `response` and `predictor` are the names of two variables in the data frame `data`. Note that the LHS of `~` represents the response and the RHS the predictors.

<sup>1</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  $\frac{\partial}{\partial \beta_1} \text{RSS}(\beta_0, \beta_1) = 0$ .

```

# Call lm
lm(yLin ~ x, data = leastSquares)
##
## Call:
## lm(formula = yLin ~ x, data = leastSquares)
##
## Coefficients:
## (Intercept)          x
## -0.6154        1.3951
lm(yQua ~ x, data = leastSquares)
##
## Call:
## lm(formula = yQua ~ x, data = leastSquares)
##
## Coefficients:
## (Intercept)          x
## 0.9710        -0.8035
lm(yExp ~ x, data = leastSquares)
##
## Call:
## lm(formula = yExp ~ x, data = leastSquares)
##
## Coefficients:
## (Intercept)          x
## 1.270        1.007

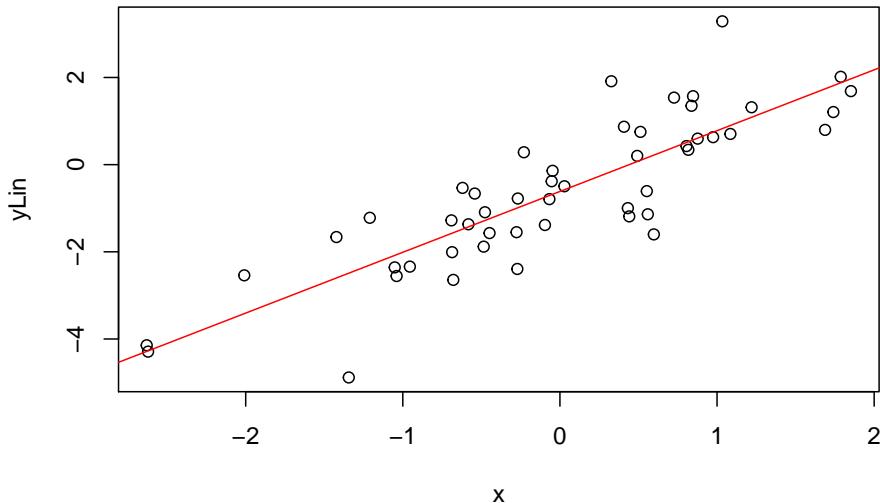
# The lm object
mod <- lm(yLin ~ x, data = leastSquares)
mod
##
## Call:
## lm(formula = yLin ~ x, data = leastSquares)
##
## Coefficients:
## (Intercept)          x
## -0.6154        1.3951

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

# We can access these elements by $
mod$coefficients
## (Intercept)          x
## -0.6153744  1.3950973

# We can produce a plot with the linear fit easily
plot(x, yLin)
abline(coef = mod$coefficients, col = 2)

```



Check that you can *not* improve the error in Figure 2.2.1 when using the coefficients given by `lm`, *if* vertical distances are selected. Check also that these coefficients are *only* optimal for vertical distances.

An interesting exercise is to check that `lm` is actually implementing the estimates given in (2.3):

```
# Covariance
Sxy <- cov(x, yLin)

# Variance
Sx2 <- var(x)

# Coefficients
beta1 <- Sxy / Sx2
beta0 <- mean(yLin) - beta1 * mean(x)
c(beta0, beta1)
## [1] -0.6153744  1.3950973

# Output from lm
mod <- lm(yLin ~ x, data = leastSquares)
mod$coefficients
## (Intercept)           x
## -0.6153744  1.3950973
```



The *population regression coefficients*,  $(\beta_0, \beta_1)$ , **are not the same** as the *estimated regression coefficients*,  $(\hat{\beta}_0, \hat{\beta}_1)$ :

- $(\beta_0, \beta_1)$  are the theoretical and **always** unknown quantities (except under controlled scenarios).
- $(\hat{\beta}_0, \hat{\beta}_1)$  are the estimates computed from the data. In particular, they are the output of `lm`. They are *random variables*, since they are computed from the random sample  $(X_1, Y_1), \dots, (X_n, Y_n)$ .

In an abuse of notation, the term *regression line* is often used to denote both the *theoretical* ( $y = \beta_0 + \beta_1 x$ ) and the *estimated* ( $y = \hat{\beta}_0 + \hat{\beta}_1 x$ ) regression lines.

### 2.2.1.1 Case study application

Let's get back to the `wine` dataset and compute some simple linear regressions. Prior to that, let's begin by summarizing the information in Table 2.1 to get a grasp of the structure of the data. For that, we first correctly import the dataset into R:

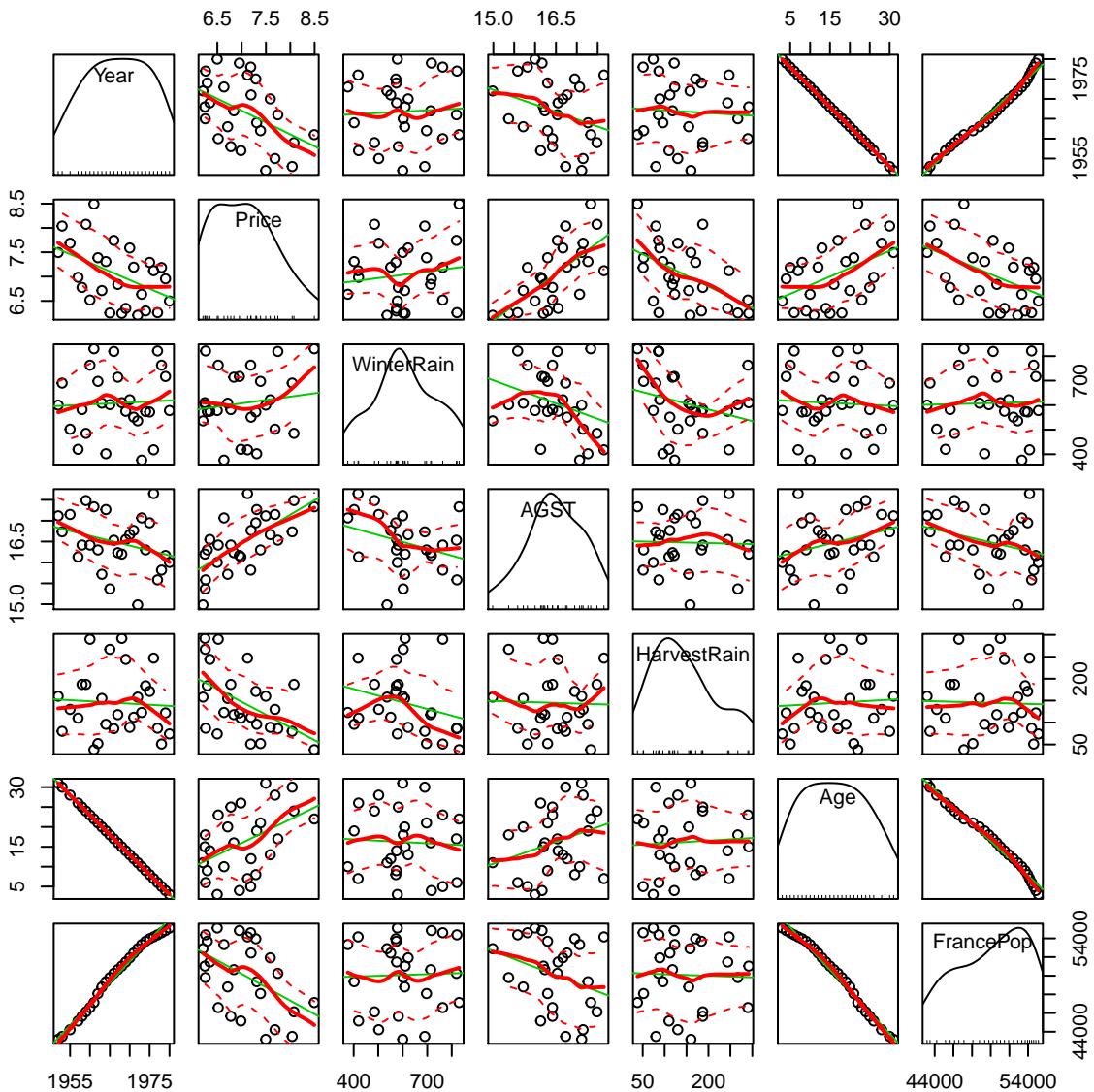
```
# Read data
wine <- read.table(file = "wine.csv", header = TRUE, sep = ",")
```

Now we can conduct a quick exploratory analysis to have insights into the data:

```
# Numerical - marginal distributions
summary(wine)

##      Year          Price        WinterRain        AGST
##  Min.  :1952  Min.   :6.205  Min.   :376.0  Min.   :14.98
##  1st Qu.:1960 1st Qu.:6.508  1st Qu.:543.5  1st Qu.:16.15
##  Median  :1967  Median :6.984  Median :600.0  Median :16.42
##  Mean    :1967  Mean   :7.042  Mean   :608.4  Mean   :16.48
##  3rd Qu.:1974  3rd Qu.:7.441  3rd Qu.:705.5  3rd Qu.:17.01
##  Max.    :1980  Max.   :8.494  Max.   :830.0  Max.   :17.65
##      HarvestRain        Age        FrancePop
##  Min.   : 38.0  Min.   : 3.00  Min.   :43184
##  1st Qu.: 88.0  1st Qu.: 9.50  1st Qu.:46856
##  Median  :123.0  Median :16.00  Median :50650
##  Mean    :144.8  Mean   :16.19  Mean   :50085
##  3rd Qu.:185.5  3rd Qu.:22.50  3rd Qu.:53511
##  Max.    :292.0  Max.   :31.00  Max.   :55110

# Graphical - pairwise relations
scatterplotMatrix(wine)
```



As we can see, `Year` and `FrancePop` are very dependent, and `Year` and `Age` are *perfectly* dependent. This is so because  $\text{Age} = 1983 - \text{Year}$ . Therefore, we opt to remove the predictor `Year` and use it to set the case names, which can be helpful later for identifying outliers:

```
# Set row names to Year - useful for outlier identification
row.names(wine) <- wine$Year
wine$Year <- NULL
```

Remember that the objective is to **predict** `Price`. Based on the above matrix scatterplot, the best we can predict `Price` by a simple linear regression seems to be with `AGST` or `HarvestRain`. Let's see which one yields the larger  $R^2$ , which, as we will see later in the chapter, is indicative of the performance of the linear model.

```
# Price ~ AGST
modAGST <- lm(Price ~ AGST, data = wine)

# Summary of the model
summary(modAGST)
## 
## Call:
```

```

## lm(formula = Price ~ AGST, data = wine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78370 -0.23827 -0.03421  0.29973  0.90198
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -3.5469    2.3641  -1.500 0.146052    
## AGST         0.6426    0.1434   4.483 0.000143 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4819 on 25 degrees of freedom
## Multiple R-squared:  0.4456, Adjusted R-squared:  0.4234 
## F-statistic: 20.09 on 1 and 25 DF,  p-value: 0.0001425

# The summary is also an object
sumModAGST <- summary(modAGST)
names(sumModAGST)
## [1] "call"          "terms"        "residuals"      "coefficients" 
## [5] "aliased"       "sigma"        "df"            "r.squared"    
## [9] "adj.r.squared" "fstatistic"    "cov.unscaled" 

# R 2
sumModAGST$r.squared
## [1] 0.4455894

```



Complete the analysis by computing the linear models `Price ~ FrancePop`, `Price ~ Age` and `Price ~ WinterRain`. Name them as `modFrancePop`, `modAge`, and `modWinterRain`. Obtain their  $R^2$  and display them in a table like:

| Predictor   | $R^2$  |
|-------------|--------|
| AGST        | 0.4456 |
| HarvestRain | 0.2572 |
| FrancePop   | 0.2314 |
| Age         | 0.2120 |
| WinterRain  | 0.0181 |

It seems that none of these simple linear models on their own are properly explaining `Price`. Intuitively, it would make sense to *bind* them together to achieve a better explanation of `Price`. Let's see how to do that.

## 2.2.2 Multiple linear model

The multiple linear model extends the simple linear model by describing the relation between several random variables  $X_1, \dots, X_p$ <sup>2</sup> and  $Y$ . Therefore, as before, the multiple linear model is *constructed by assuming* that the linear relation

<sup>2</sup>Note that now  $X_1$  represents the first predictor and not the first element of a sample of  $X$ .

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \varepsilon \quad (2.4)$$

holds between the predictors  $X_1, \dots, X_p$  and the response  $Y$ . In (2.4),  $\beta_0$  is the *intercept* and  $\beta_1, \dots, \beta_p$  are the *slopes*, respectively.  $\varepsilon$  is a random variable with mean zero and independent from  $X_1, \dots, X_p$ . Another way of looking at (2.4) is

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p, \quad (2.5)$$

since  $\mathbb{E}[\varepsilon|X_1 = x_1, \dots, X_p = x_p] = 0$ .

The LHS of (2.5) is the conditional expectation of  $Y$  given  $X_1, \dots, X_p$ . It represents how the mean of the random variable  $Y$  is changing, now according to particular values of several predictors. With the RHS, what we are saying is that the mean of  $Y$  is changing in a *linear* fashion with respect to the values of  $X_1, \dots, X_p$ . Hence the interpretation of the coefficients:

- $\beta_0$ : is the mean of  $Y$  when  $X_1 = \dots = X_p = 0$ .
- $\beta_j$ ,  $1 \leq j \leq p$ : is the increment in mean of  $Y$  for an increment of one unit in  $X_j = x_j$ , provided that the remaining variables  $X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_p$  do not change.

Figure 2.2.2 illustrates the geometrical interpretation of a multiple linear model: a plane in the  $(p+1)$ -dimensional space. If  $p = 1$ , the plane is the regression line for simple linear regression. If  $p = 2$ , then the plane can be visualized in a three-dimensional plot.

The least squares regression plane  $y = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$  and its dependence on the kind of squared distance considered. Application also available here.

The estimation of  $\beta_0, \beta_1, \dots, \beta_p$  is done as in simple linear regression, by minimizing the RSS (which now accounts for the sum of squared distances of the data to the vertical projections in the plane). First we need to introduce some helpful matrix notation:

- A sample of  $(X_1, \dots, X_p, Y)$  is  $(X_{11}, \dots, X_{1p}, Y_1), \dots, (X_{n1}, \dots, X_{np}, Y_n)$ , where  $X_{ij}$  denotes the  $i$ -th observation of the  $j$ -th predictor  $X_j$ . We denote with  $\mathbf{X}_i := (X_{i1}, \dots, X_{ip})$  to the  $i$ -th observation of  $(X_1, \dots, X_p)$ , so the sample simplifies to  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$ .
- The *design matrix* contains all the information of the predictors and a column of ones

$$\mathbf{X} := \begin{pmatrix} 1 & X_{11} & \cdots & X_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & \cdots & X_{np} \end{pmatrix}_{n \times (p+1)}.$$

- The *vector of responses*  $\mathbf{Y}$ , the *vector of coefficients*  $\boldsymbol{\beta}$  and the *vector of errors* are, respectively,

$$\mathbf{Y} := \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}_{n \times 1}, \quad \boldsymbol{\beta} := \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}_{(p+1) \times 1} \quad \text{and} \quad \boldsymbol{\varepsilon} := \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}_{n \times 1}.$$

Thanks to the matrix notation, we can turn the sample version of the multiple linear model, namely

$$Y_i = \beta_0 + \beta_1 X_{i1} + \dots + \beta_p X_{ip} + \varepsilon_i, \quad i = 1, \dots, n,$$

into something as compact as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$



Recall that if  $p = 1$  we have the simple linear model. In this case:

$$\mathbf{X} = \begin{pmatrix} 1 & X_{11} \\ \vdots & \vdots \\ 1 & X_{n1} \end{pmatrix}_{n \times 2} \quad \text{and} \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}_{2 \times 1}.$$

With this notation, the RSS for the multiple linear regression is

$$\begin{aligned} \text{RSS}(\boldsymbol{\beta}) &:= \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \dots - \beta_p X_{ip})^2 \\ &= (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}). \end{aligned} \quad (2.6)$$

The RSS aggregates the *squared vertical distances* from the data to a regression plane given by  $\boldsymbol{\beta}$ . The least squares estimators are *the minimizers of the RSS*<sup>3</sup>:

$$\hat{\boldsymbol{\beta}} := \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \text{RSS}(\boldsymbol{\beta}).$$

Luckily, thanks to the matrix form of (2.6), it is possible<sup>4</sup> to compute a closed-form expression for the least squares estimates:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}. \quad (2.7)$$



There are some similarities between (2.7) 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  $Y$  weighted by the variance of  $\mathbf{X}$ .

Let's check that indeed the coefficients given by `lm` are the ones given by equation (2.7). For that purpose we consider the `leastSquares3D` data frame in the `least-squares-3D.RData` dataset (download). Among other variables, the data frame contains the response `yLin` and the predictors `x1` and `x2`.

```
load(file = "leastSquares3D.RData")
```

Let's compute the coefficients of the regression of `yLin` into the predictors `x1` and `x2`, which is denoted by `yLin ~ x1 + x2`. Note the use of `+` for including all the predictors. This does *not* mean that they are all summed and then the regression is done on the sum<sup>5</sup>. Instead, this notation is designed to resemble the multiple linear model. ].

```
# Output from lm
mod <- lm(yLin ~ x1 + x2, data = leastSquares3D)
mod$coefficients
## (Intercept)          x1          x2
## -0.5702694  0.4832624  0.3214894
```

<sup>3</sup>They are unique and always exist.

<sup>4</sup>If follows from  $\frac{\partial \mathbf{Ax}}{\partial \mathbf{x}} = \mathbf{A}$  and  $\frac{\partial f(\mathbf{x})'g(\mathbf{x})}{\partial \mathbf{x}} = f(\mathbf{x})' \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} + g(\mathbf{x})' \frac{\partial}{\partial f(\mathbf{x})}$  for two vector-valued functions  $f$  and  $g$ .

<sup>5</sup>If you wanted to do so, you will need the function `I()` for indicating that `+` is not including predictors in the model, but is acting as a sum operator.

```

# Matrix X
X <- cbind(1, x1, x2)

# Vector Y
Y <- yLin

# Coefficients
beta <- solve(t(X) %*% X) %*% t(X) %*% Y
# %*% multiplies matrices
# solve() computes the inverse of a matrix
# t() transposes a matrix
beta
##          [,1]
## -0.5702694
## x1  0.4832624
## x2  0.3214894

```



Compute  $\beta$  for the regressions  $yLin \sim x1 + x2$ ,  $yQua \sim x1 + x2$ , and  $yExp \sim x2 + x3$  using:

- equation (2.7) and
- the function `lm`.

Check that both are the same.

Once we have the least squares estimates  $\hat{\beta}$ , we can define the next concepts:

- The *fitted values*  $\hat{Y}_1, \dots, \hat{Y}_n$ , where

$$\hat{Y}_i := \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \dots + \hat{\beta}_p X_{ip}, \quad i = 1, \dots, n.$$

They are the vertical projections of  $Y_1, \dots, Y_n$  into the fitted plane (see Figure 2.2.2). In a matrix form, inputting (2.6)

$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} = \mathbf{H}\mathbf{Y},$$

where  $\mathbf{H} := \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$  is called the *hat matrix* because it “puts the hat into  $\mathbf{Y}$ ”. What it does is to project  $\mathbf{Y}$  into the regression plane (see Figure 2.2.2).

- The *residuals* (or estimated errors)  $\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_n$ , where

$$\hat{\varepsilon}_i := Y_i - \hat{Y}_i, \quad i = 1, \dots, n.$$

They are the vertical distances between actual data and fitted data.

These objects are present in the output of `lm`:

```

# Fitted values
mod$fitted.values

# Residuals
mod$residuals

```

We conclude with an insight on the relation of multiple and simple linear regressions. It is illustrated in Figure 2.2.



Consider the multiple linear model  $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$  and its associated simple linear models  $Y = \alpha_0 + \alpha_1 X_1 + \varepsilon$  and  $Y = \gamma_0 + \gamma_1 X_2 + \varepsilon$ . Assume that we have a sample  $(X_{11}, X_{12}, Y_1), \dots, (X_{n1}, X_{n2}, Y_n)$ . Then, in general,  $\hat{\alpha}_0 \neq \hat{\beta}_0$ ,  $\hat{\alpha}_1 \neq \hat{\beta}_1$ ,  $\hat{\gamma}_0 \neq \hat{\beta}_0$ , and  $\hat{\gamma}_1 \neq \hat{\beta}_1$ . That is, in general, **the inclusion of a new predictor changes the coefficient estimates**.

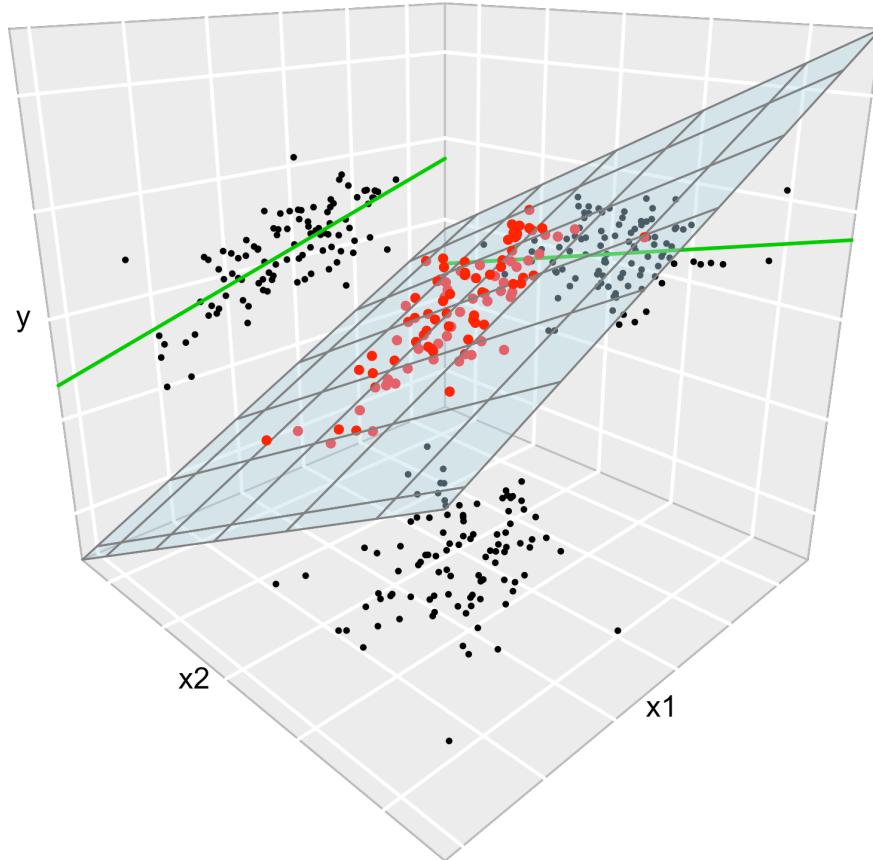


Figure 2.2: The regression plane (blue) and its relation with the simple linear regressions (green lines). The red points represent the sample for  $(X_1, X_2, Y)$  and the black points the subsamples for  $(X_1, X_2)$  (bottom),  $(X_1, Y)$  (left), and  $(X_2, Y)$  (right).

The data used in Figure 2.2 is:

```
set.seed(212542)
n <- 100
x1 <- rnorm(n, sd = 2)
x2 <- rnorm(n, mean = x1, sd = 3)
y <- 1 + 2 * x1 - x2 + rnorm(n, sd = 1)
data <- data.frame(x1 = x1, x2 = x2, y = y)
```



With the above `data`, check how the fitted coefficients change for  $y \sim x_1$ ,  $y \sim x_2$ , and  $y \sim x_1 + x_2$ .

### 2.2.2.1 Case study application

A natural step now is to extend these simple regressions to increase both the  $R^2$  and the prediction accuracy for `Price` by means of the multiple linear regression:

```
# Regression on all the predictors
modWine1 <- lm(Price ~ Age + AGST + FrancePop + HarvestRain + WinterRain, data = wine)

# A shortcut
modWine1 <- lm(Price ~ ., data = wine)
modWine1
##
## Call:
## lm(formula = Price ~ ., data = wine)
##
## Coefficients:
## (Intercept) WinterRain          AGST      HarvestRain          Age
## -2.343e+00   1.153e-03   6.144e-01   -3.837e-03   1.377e-02
## FrancePop
## -2.213e-05

# Summary
summary(modWine1)
##
## Call:
## lm(formula = Price ~ ., data = wine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46541 -0.24133  0.00413  0.18974  0.52495
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.343e+00 7.697e+00 -0.304 0.76384
## WinterRain   1.153e-03 4.991e-04  2.311 0.03109 *
## AGST         6.144e-01 9.799e-02  6.270 3.22e-06 ***
## HarvestRain -3.837e-03 8.366e-04 -4.587 0.00016 ***
## Age          1.377e-02 5.821e-02  0.237 0.81531
## FrancePop   -2.213e-05 1.268e-04 -0.175 0.86313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.293 on 21 degrees of freedom
## Multiple R-squared:  0.8278, Adjusted R-squared:  0.7868
## F-statistic: 20.19 on 5 and 21 DF,  p-value: 2.232e-07
```

The fitted regression is  $\text{Price} = -2.343 + 0.013 \times \text{Age} + 0.614 \times \text{AGST} - 0.000 \times \text{FrancePop} - 0.003 \times \text{HarvestRain} + 0.001 \times \text{WinterRain}$ . Recall that the 'Multiple R-squared' has almost doubled with respect to the best simple linear regression! This tells us that combining several predictors may lead to important performance gains in the prediction of the response. However, note that the  $R^2$  of the multiple linear model is *not* the sum of the  $R^2$ 's of the simple linear models. The performance gain of combining predictors is hard to anticipate from the single-predictor models.

## 2.3 Assumptions of the model

Why do we need assumptions? To make **inference** on the model parameters. In other words, to infer properties about the *unknown* population coefficients  $\beta$  from the sample  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$ .

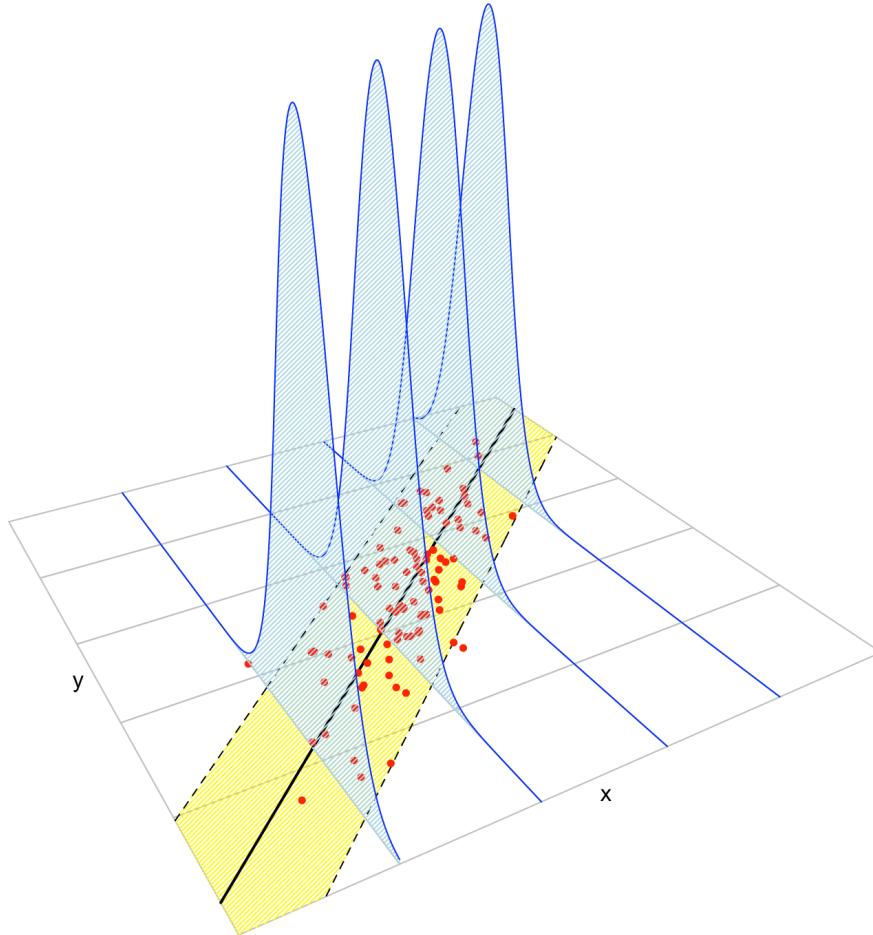


Figure 2.3: The key concepts of the simple linear model. The yellow band denotes where 95% of the data is, according to the model.

The assumptions of the multiple linear model are:

- i. **Linearity:**  $\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$ .
- ii. **Homoscedasticity:**  $\text{Var}[\varepsilon|X_1 = x_1, \dots, X_p = x_p] = \sigma^2$ .
- iii. **Normality:**  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ .
- iv. **Independence of the errors:**  $\varepsilon_1, \dots, \varepsilon_n$  are independent (or uncorrelated,  $\mathbb{E}[\varepsilon_i \varepsilon_j] = 0$ ,  $i \neq j$ , since they are assumed to be normal).

A good one-line summary of the linear model is the following (independence is implicit)

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim \mathcal{N}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p, \sigma^2). \quad (2.8)$$

Recall that, except assumption iv, the rest are expressed in terms of the random variables, not in terms of the sample. Thus are *population versions*, rather than *sample versions*. It is however trivial to express (2.8) in terms of assumptions about the sample  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$ :

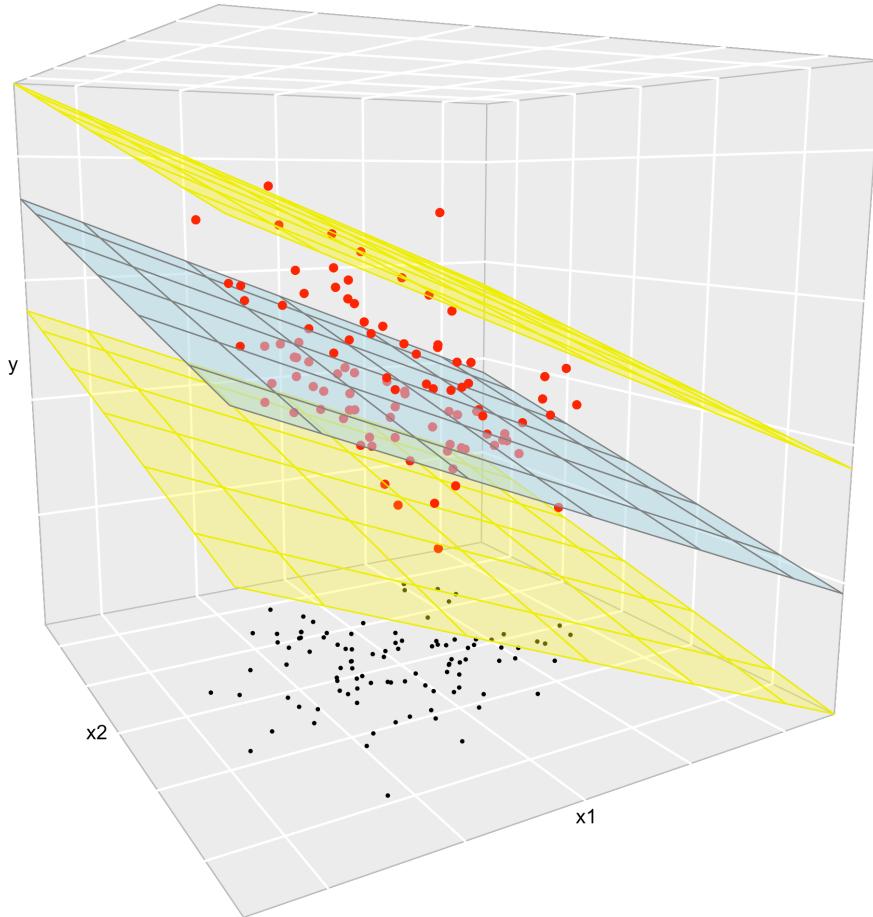


Figure 2.4: The key concepts of the multiple linear model when  $p = 2$ . The space between the yellow planes denotes where 95% of the data is, according to the model.

$$Y_i | (X_{i1} = x_{i1}, \dots, X_{ip} = x_{ip}) \sim \mathcal{N}(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}, \sigma^2), \quad i = 1, \dots, n \quad (2.9)$$

with  $Y_1, \dots, Y_n$  being independent conditionally on the sample of predictors. Equivalently stated in a compact matrix way, the assumptions of the model on the sample are:

$$\mathbf{Y} | \mathbf{X} \sim \mathcal{N}_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}). \quad (2.10)$$



Recall:

- Nothing is said about the distribution of  $X_1, \dots, X_p$ . They could be deterministic (called *fixed design*) or random (*random design*). They could be discrete or continuous.
- $X_1, \dots, X_p$  are **not required to be independent** between them.
- $Y$  **has to be continuous**, since the errors are normal – recall (2.1).

Figures 2.5 and 2.6 represent situations where the assumptions of the model for  $p = 1$  are respected and violated, respectively.

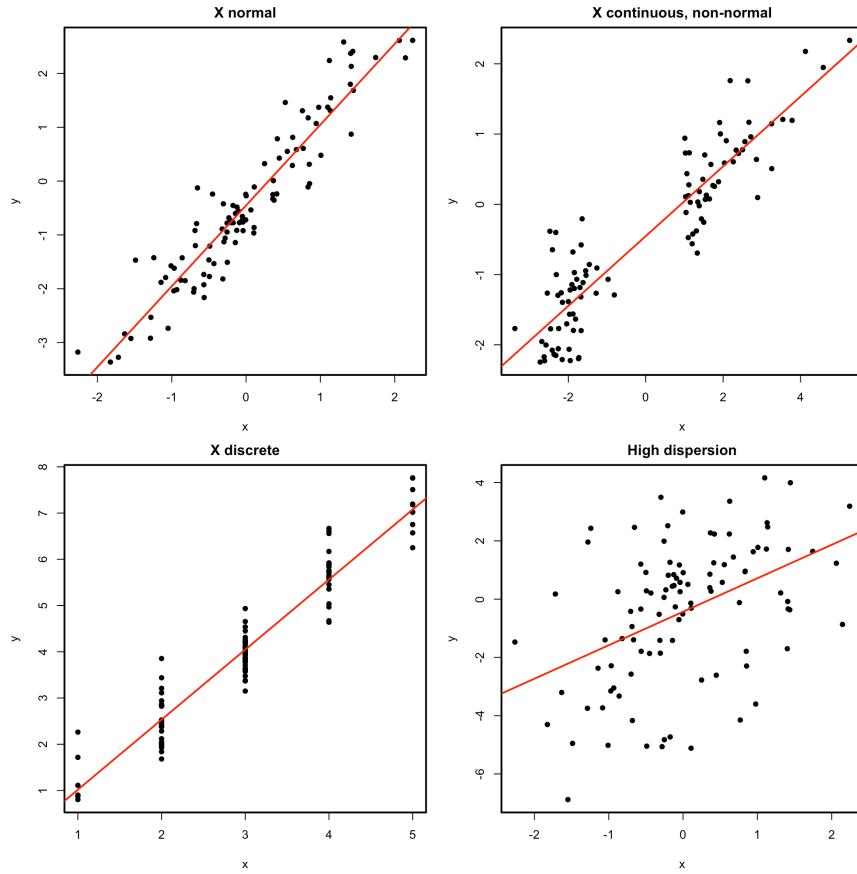


Figure 2.5: Perfectly valid simple linear models (all the assumptions are verified).

Figure 2.3 represents situations where the assumptions of the model are respected and violated, for the situation with two predictors. Clearly, the inspection of the scatterplots for identifying strange patterns is more complicated than in simple linear regression – and here we are dealing only with two predictors.

Valid (all the assumptions are verified) and problematic (a single assumption does not hold) multiple linear models, when there are two predictors. Application also available here.



The dataset **assumptions.RData** (download) contains the variables  $x_1, \dots, x_9$  and  $y_1, \dots, y_9$ . For each regression  $y_1 \sim x_1, \dots, y_9 \sim x_9$ :

- Check whether the assumptions of the linear model are being satisfied (make a scatterplot with a regression line).
- State which assumption(s) are violated and justify your answer.

## 2.4 Inference for model parameters

The assumptions introduced in the previous section allow to specify what is the distribution of the *random vector*  $\hat{\beta}$ . The distribution is derived conditionally on the sample predictors  $\mathbf{X}_1, \dots, \mathbf{X}_n$ . In other words, we assume that the randomness of  $\mathbf{Y} = \mathbf{X}\beta + \varepsilon$  comes only from the error terms and not from the predictors. To denote this, we employ lowercase for the sample predictors  $\mathbf{x}_1, \dots, \mathbf{x}_n$ .

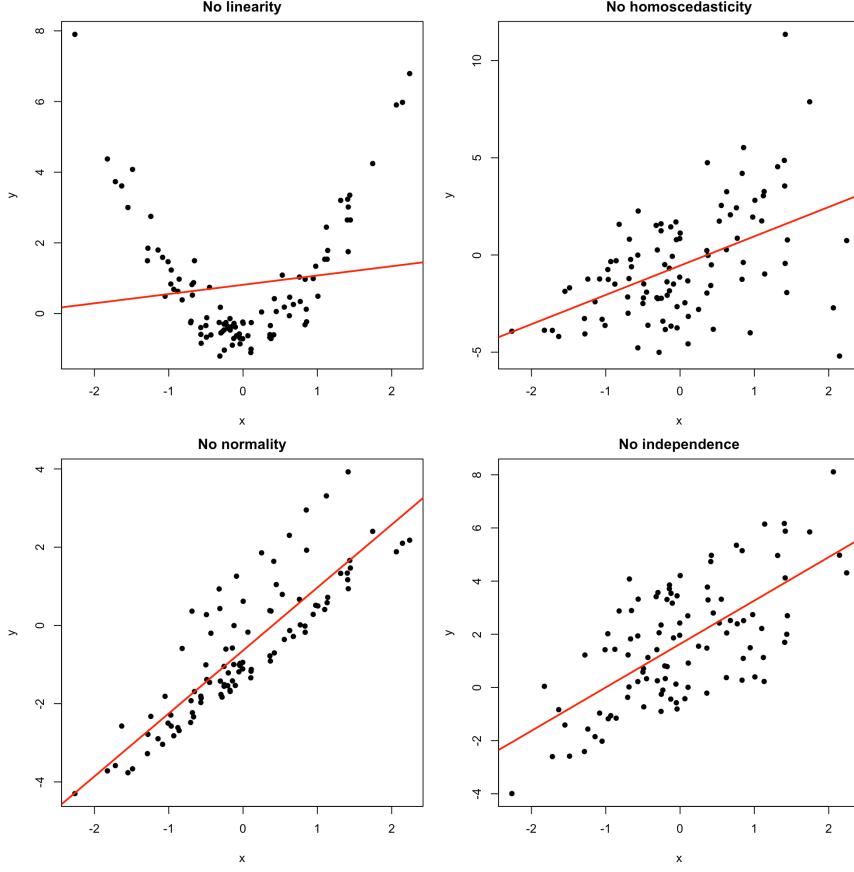


Figure 2.6: Problematic simple linear models (a single assumption does not hold).

### 2.4.1 Distributions of the fitted coefficients

The distribution of  $\hat{\beta}$  is:

$$\hat{\beta} \sim \mathcal{N}_{p+1}(\beta, \sigma^2(\mathbf{X}'\mathbf{X})^{-1}). \quad (2.11)$$

This result can be obtained from the form of  $\hat{\beta}$  given in (2.7) and the sample version of the model assumptions given in (2.10). Equation (2.11) implies that each *individual*  $\hat{\beta}_j$  satisfies

$$\hat{\beta}_j \sim \mathcal{N}(\beta_j, \text{SE}(\hat{\beta}_j)^2), \quad (2.12)$$

where  $\text{SE}(\hat{\beta}_j)$  is the *standard error*,  $\text{SE}(\hat{\beta}_j)^2 := \sigma^2 v_j$ , and

$v_j$  is the  $j$ -th element of the diagonal of  $(\mathbf{X}'\mathbf{X})^{-1}$ .

Recall that an equivalent form for (2.12) is (why?)

$$\frac{\hat{\beta}_j - \beta_j}{\text{SE}(\hat{\beta}_j)} \sim \mathcal{N}(0, 1).$$

The interpretation of (2.12) is simpler in the case with  $p = 1$ , where

$$\hat{\beta}_0 \sim \mathcal{N}(\beta_0, \text{SE}(\hat{\beta}_0)^2), \quad \hat{\beta}_1 \sim \mathcal{N}(\beta_1, \text{SE}(\hat{\beta}_1)^2) \quad (2.13)$$

with

$$\text{SE}(\hat{\beta}_0)^2 = \frac{\sigma^2}{n} \left[ 1 + \frac{\bar{X}^2}{s_x^2} \right], \quad \text{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{ns_x^2}. \quad (2.14)$$

Therefore, some important remarks on (2.13) and (2.14) are

- **Bias.** Both estimates are unbiased. That means that their expectations are the true coefficients.
- **Variance.** The variances  $\text{SE}(\hat{\beta}_0)^2$  and  $\text{SE}(\hat{\beta}_1)^2$  have an interesting interpretation in terms of its components:
  - *Sample size n.* As the sample size grows, the precision of the estimators increases, since both variances decrease.
  - *Error variance  $\sigma^2$ .* The more disperse the error is, the less precise the estimates are, since more vertical variability is present.
  - *Predictor variance  $s_x^2$ .* If the predictor is spread out (large  $s_x^2$ ), then it is easier to fit a regression line: we have information about the data trend over a long interval. If  $s_x^2$  is small, then all the data is concentrated on a narrow vertical band, so we have a much more limited view of the trend.
  - *Mean  $\bar{X}$ .* It has influence only on the precision of  $\hat{\beta}_0$ . The larger  $\bar{X}$  is, the less precise  $\hat{\beta}_0$  is.

Illustration of the randomness of the fitted coefficients  $(\hat{\beta}_0, \hat{\beta}_1)$  and the influence of  $n, \sigma^2$  and  $s_x^2$ . The sample predictors  $x_1, \dots, x_n$  are fixed and new responses  $Y_1, \dots, Y_n$  are generated each time from a linear model  $Y = \beta_0 + \beta_1 X + \varepsilon$ . Application also available here.

The insights about (2.11) are more convoluted. The following broad remarks, extensions of what happened when  $p = 1$ , apply:

- **Bias.** Both estimates are unbiased.
- **Variance.** Depends on:
  - *Sample size n.* Hidden inside  $\mathbf{X}'\mathbf{X}$ . As  $n$  grows, the precision of the estimators increases.
  - *Error variance  $\sigma^2$ .* The larger  $\sigma^2$  is, the less precise  $\hat{\beta}$  is.
  - *Predictor sparsity  $(\mathbf{X}'\mathbf{X})^{-1}$ .* The more *sparse* the predictor is (small  $|(\mathbf{X}'\mathbf{X})^{-1}|$ ), the more precise  $\hat{\beta}$  is.

The problem with (2.11) is that  $\sigma^2$  is *unknown* in practice, so we need to estimate  $\sigma^2$  from the data. We do so by computing a rescaled sample variance of the residuals  $\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_n$ :

$$\hat{\sigma}^2 := \frac{1}{n-p-1} \sum_{i=1}^n \hat{\varepsilon}_i^2. \quad (2.15)$$

Note the  $n-p-1$  in the denominator. Now  $n-p-1$  are the *degrees of freedom*, the number of data points minus the number of already fitted parameters ( $p$  slopes and 1 intercept). For the interpretation of  $\hat{\sigma}^2$ , it is key to realize that *the mean of the residuals  $\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_n$  is zero*, this is  $\bar{\hat{\varepsilon}} = 0$ . Therefore,  $\hat{\sigma}^2$  is indeed a rescaled sample variance of the residuals which estimates the variance of  $\varepsilon$ .

If we use the estimate  $\hat{\sigma}^2$  instead of  $\sigma^2$ , we get more useful distributions than (2.12)

$$\frac{\hat{\beta}_j - \beta_j}{\hat{\text{SE}}(\hat{\beta}_j)} \sim t_{n-p-1}, \quad \hat{\text{SE}}(\hat{\beta}_j)^2 := \hat{\sigma}^2 v_j^2 \quad (2.16)$$

where  $t_{n-p-1}$  represents the Student's  $t$  distribution with  $n - p - 1$  degrees of freedom.

The LHS of (2.16) is the  $t$ -statistic for  $\beta_j$ ,  $j = 0, \dots, p$ . We will employ them for building confidence intervals and hypothesis tests.

### 2.4.2 Confidence intervals for the coefficients

Thanks to (2.16), we can have the  $100(1 - \alpha)\%$  Confidence Intervals (CI) for the coefficient  $\beta_j$ ,  $j = 0, \dots, p$ :

$$\left( \hat{\beta}_j \pm \hat{\text{SE}}(\hat{\beta}_j) t_{n-p-1; \alpha/2} \right) \quad (2.17)$$

where  $t_{n-p-1; \alpha/2}$  is the  $\alpha/2$ -upper quantile of the  $t_{n-p-1}$ . Usually,  $\alpha = 0.10, 0.05, 0.01$  are considered.

This *random CI contains the unknown coefficient  $\beta_j$  "with a probability of  $1 - \alpha$ "*. The previous quoted statement has to be understood as follows. Suppose you have 100 samples generated according to a linear model. If you compute the CI for a coefficient, then in approximately  $100(1 - \alpha)$  of the samples the true coefficient would be actually inside the random CI. Note also that the CI is symmetric around  $\hat{\beta}_j$ . This is illustrated in Figure 2.4.2.

Illustration of the randomness of the CI for  $\beta_0$  at  $100(1 - \alpha)\%$  confidence. The plot shows 100 random CIs for  $\beta_0$ , computed from 100 random datasets generated by the same simple linear model, with intercept  $\beta_0$ . The illustration for  $\beta_1$  is completely analogous. Application also available here.

### 2.4.3 Testing on the coefficients

The distributions in (2.16) also allow us to conduct a formal *hypothesis test* on the coefficients  $\beta_j$ ,  $j = 0, \dots, p$ . For example the test for *significance* (shortcut for *significantly difference from zero*) is especially important, that is, the test of the hypotheses

$$H_0 : \beta_j = 0$$

for  $j = 0, \dots, p$ . The test of  $H_0 : \beta_j = 0$  with  $1 \leq j \leq p$  is especially interesting, since it allows us to answer whether *the variable  $X_j$  has a significant linear effect on  $Y$* . The statistic used for testing for significance is the  $t$ -statistic

$$\frac{\hat{\beta}_j - 0}{\hat{\text{SE}}(\hat{\beta}_j)},$$

which is distributed as a  $t_{n-p-1}$  under the (veracity of) the null hypothesis.

The null hypothesis  $H_0$  is tested *against* the *alternative hypothesis*,  $H_1$ . If  $H_0$  is rejected, it is *rejected in favor* of  $H_1$ . The alternative hypothesis can be *bilateral*, such as

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

or *unilateral*, such as

$$H_0 : \beta_j \geq (\leq)0 \quad \text{vs} \quad H_1 : \beta_j < (>)0.$$

For the moment, we will focus only on bilateral hypothesis.

Remember the following insights about hypothesis testing.



The analogy of conducting an hypothesis test and a **trial** can be seen in Appendix D.



In an hypothesis test, the *p-value* measures the degree of veracity of  $H_0$  according to the data. The rule of thumb is the following:

**Is the p-value lower than  $\alpha$ ?**

- Yes → **reject  $H_0$** .
- No → **do not reject  $H_0$** .

The connection of a *t*-test for  $H_0 : \beta_j = 0$  and the CI for  $\beta_j$ , both at level  $\alpha$ , is the following:

**Is 0 inside the CI for  $\beta_j$ ?**

- Yes ↔ **do not reject  $H_0$** .
- No ↔ **reject  $H_0$** .

The unilateral test  $H_0 : \beta_j \geq 0$  (respectively,  $H_0 : \beta_j \leq 0$ ) vs  $H_1 : \beta_j < 0$  ( $H_1 : \beta_j > 0$ ) can be done by means of the CI for  $\beta_j$ . If  $H_0$  is rejected, they allow us to conclude that  $\hat{\beta}_j$  is *significantly negative (positive)* and that *for the considered regression model,  $X_j$  has a significant negative (positive) effect on  $Y$* . The rule of thumb is the following:

**Is the CI for  $\beta_j$  below (above) 0 at level  $\alpha$ ?**

- Yes → **reject  $H_0$  at level  $\alpha$ . Conclude  $X_j$  has a significant negative (positive) effect on  $Y$  at level  $\alpha$** .
- No → **the criterion is not conclusive**.

#### 2.4.4 Case study application

Let's analyse the multiple linear model we have considered now that we know how to make inference on the model parameters. The relevant information is obtained with the **summary** of the model:

```
# Fit
modWine1 <- lm(Price ~ ., data = wine)

# Summary
sumModWine1 <- summary(modWine1)
sumModWine1
## 
## Call:
## lm(formula = Price ~ ., data = wine)
```

```

## 
## Residuals:
##   Min     1Q   Median     3Q    Max
## -0.46541 -0.24133  0.00413  0.18974  0.52495
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -2.343e+00 7.697e+00 -0.304 0.76384    
## WinterRain   1.153e-03 4.991e-04  2.311 0.03109 *  
## AGST        6.144e-01 9.799e-02  6.270 3.22e-06 *** 
## HarvestRain -3.837e-03 8.366e-04 -4.587 0.00016 *** 
## Age         1.377e-02 5.821e-02  0.237 0.81531    
## FrancePop   -2.213e-05 1.268e-04 -0.175 0.86313    
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.293 on 21 degrees of freedom
## Multiple R-squared:  0.8278, Adjusted R-squared:  0.7868 
## F-statistic: 20.19 on 5 and 21 DF,  p-value: 2.232e-07

# Contains the estimation of sigma ("Residual standard error")
sumModWine1$sigma
## [1] 0.2930287

# Which is the same as
sqrt(sum(modWine1$residuals^2) / modWine1$df.residual)
## [1] 0.2930287

```

The **Coefficients** block of the output of `summary` contains the next elements regarding the significance of each coefficient  $\beta_j$ , this is, the test  $H_0 : \beta_j = 0$  vs  $H_1 : \beta_j \neq 0$ :

- **Estimate**: least squares estimate  $\hat{\beta}_j$ .
- **Std. Error**: estimated standard error  $\hat{SE}(\hat{\beta}_j)$ .
- **t value**:  $t$ -statistic  $\frac{\hat{\beta}_j}{\hat{SE}(\hat{\beta}_j)}$ .
- **Pr(>|t|)**:  $p$ -value of the  $t$ -test.
- **Signif. codes**: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1: codes indicating the size of the  $p$ -value. The more stars, the more evidence supporting that  $H_0$  does not hold.

Note then that **many predictors are not significant** in `modWine1`: `FrancePop`, `Age`, and the intercept are not significant. This is an indication of an **excess of predictors** adding little information to the response. An explanation is the almost perfect correlation between `FrancePop` and `Age` shown before: one of them is not adding any extra information to explain `Price`. This complicates the model unnecessarily and, more importantly, it has the undesirable effect of making the **coefficient estimates less precise**. We opt to remove the predictor `FrancePop` from the model since it is exogenous to the wine context (notice this is a context-guided decision, not data-driven). A data-driven justification of the removal of this variable is that it is the least significant in `modWine1`.

Then, the model without `FrancePop` (notice the use of `-` for *excluding* a particular predictor) is:

```

modWine2 <- lm(Price ~ . - FrancePop, data = wine)
summary(modWine2)
## 
## Call:
## lm(formula = Price ~ . - FrancePop, data = wine)
## 
```

```

## Residuals:
##      Min     1Q  Median     3Q     Max
## -0.46024 -0.23862  0.01347  0.18601  0.53443
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -3.6515703  1.6880876 -2.163  0.04167 *  
## WinterRain   0.0011667  0.0004820  2.420  0.02421 *  
## AGST        0.6163916  0.0951747  6.476 1.63e-06 *** 
## HarvestRain -0.0038606  0.0008075 -4.781 8.97e-05 *** 
## Age         0.0238480  0.0071667  3.328  0.00305 **  
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2865 on 22 degrees of freedom
## Multiple R-squared:  0.8275, Adjusted R-squared:  0.7962 
## F-statistic: 26.39 on 4 and 22 DF,  p-value: 4.057e-08

```

All the coefficients are significant at level  $\alpha = 0.05$ . Therefore, there is no clear redundant information. In addition, the  $R^2$  is very similar to the full model, but the 'Adjusted R-squared', a weighting of the  $R^2$  to account for the number of predictors used by the model, is slightly larger. As we will see in Section 2.7.2, this means that, compared to the number of predictors used, `modWine2` explains more variability of `Price` than `modWine1`.

A handy way of comparing the coefficients of both models is `compareCoefs`:

```

compareCoefs(modWine1, modWine2)
##
## Call:
## 1: lm(formula = Price ~ ., data = wine)
## 2: lm(formula = Price ~ . - FrancePop, data = wine)
##             Est. 1     SE 1     Est. 2     SE 2
## (Intercept) -2.34e+00 7.70e+00 -3.65e+00 1.69e+00
## WinterRain   1.15e-03 4.99e-04  1.17e-03 4.82e-04
## AGST        6.14e-01 9.80e-02  6.16e-01 9.52e-02
## HarvestRain -3.84e-03 8.37e-04 -3.86e-03 8.07e-04
## Age         1.38e-02 5.82e-02  2.38e-02 7.17e-03
## FrancePop   -2.21e-05 1.27e-04

```

Note how the coefficients for `modWine2` have smaller errors than `modWine1`.

The individual CIs for the unknown  $\beta_j$ 's can be obtained by applying the `confint` function to an `lm` object. Let's compute the CIs for the model coefficients of `modWine1`, `modWine2`, and a new `modWine3`:

```

# Fit a new model
modWine3 <- lm(Price ~ Age + WinterRain, data = wine)
summary(modWine3)
##
## Call:
## lm(formula = Price ~ Age + WinterRain, data = wine)
## 
## Residuals:
##      Min     1Q  Median     3Q     Max
## -0.88964 -0.51421 -0.00066  0.43103  1.06897
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -3.6515703  1.6880876 -2.163  0.04167 *  
## Age         0.0238480  0.0071667  3.328  0.00305 **  
## WinterRain  0.0011667  0.0004820  2.420  0.02421 *  
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.2865 on 22 degrees of freedom
## Multiple R-squared:  0.8275, Adjusted R-squared:  0.7962 
## F-statistic: 26.39 on 4 and 22 DF,  p-value: 4.057e-08

```

```

##             Estimate Std. Error t value Pr(>|t|) 
## (Intercept) 5.9830427  0.5993667  9.982 5.09e-10 ***
## Age         0.0360559  0.0137377  2.625  0.0149 *  
## WinterRain  0.0007813  0.0008780  0.890  0.3824 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.5769 on 24 degrees of freedom
## Multiple R-squared:  0.2371, Adjusted R-squared:  0.1736 
## F-statistic:  3.73 on 2 and 24 DF,  p-value: 0.03884

# Confidence intervals at 95%
# CI: (lwr, upr)
confint(modWine3)
##                  2.5 %      97.5 % 
## (Intercept) 4.746010626 7.220074676
## Age         0.007702664 0.064409106
## WinterRain -0.001030725 0.002593278

# Confidence intervals at other levels
confint(modWine3, level = 0.90)
##                  5 %      95 % 
## (Intercept) 4.9575969417 7.008488360
## Age         0.0125522989 0.059559471
## WinterRain -0.0007207941 0.002283347
confint(modWine3, level = 0.99)
##                  0.5 %      99.5 % 
## (Intercept) 4.306650310 7.659434991
## Age         -0.002367633 0.074479403
## WinterRain -0.001674299 0.003236852

# Compare with previous models
confint(modWine1)
##                  2.5 %      97.5 % 
## (Intercept) -1.834844e+01 13.6632391095
## WinterRain   1.153872e-04  0.0021910509
## AGST         4.106337e-01  0.8182146540
## HarvestRain -5.577203e-03 -0.0020974232
## Age          -1.072931e-01  0.1348317795
## FrancePop   -2.858849e-04  0.0002416171
confint(modWine2)
##                  2.5 %      97.5 % 
## (Intercept) -7.1524497573 -0.150690903
## WinterRain   0.0001670449  0.002166393
## AGST         0.4190113907  0.813771726
## HarvestRain -0.0055353098 -0.002185890
## Age          0.0089852800  0.038710748
confint(modWine3)
##                  2.5 %      97.5 % 
## (Intercept) 4.746010626 7.220074676
## Age         0.007702664 0.064409106
## WinterRain -0.001030725 0.002593278

```

In `modWine3`, the 95% CI for  $\beta_0$  is (4.7460, 7.2201), for  $\beta_1$  is (0.0077, 0.0644), and for  $\beta_2$  is (-0.0010, 0.0026).

Therefore, we can say with a 95% confidence that *the coefficient of WinterRain is non significant* (0 is inside the CI). But inspecting the CI of  $\beta_2$  in `modWine2` we can see that *it is significant* for the model! How is this possible? The answer is that the presence of extra predictors affects the coefficient estimate, as we saw in Figure 2.2. Therefore, the precise statement to make is: **in the model `Price ~ Age + WinterRain`**, with  $\alpha = 0.05$ , the coefficient of `WinterRain` is non significant. Note that this **does not** mean that it will be always non significant: in `Price ~ Age + AGST + HarvestRain + WinterRain` it is.



Compute and interpret the CIs for the coefficients, at levels  $\alpha = 0.10, 0.05, 0.01$ , for the following regressions:

- `Price ~ WinterRain + HarvestRain + AGST (wine)`
- `AGST ~ Year + FrancePop (wine)`



For the `assumptions` dataset, do the following:

- Regression `y7 ~ x7`. Check that:
  - \* The intercept of is not significant for the regression at any reasonable level  $\alpha$ .
  - \* The slope is significant for any  $\alpha > 10^{-7}$ .
- Regression `y6 ~ x6`. Assume the linear model assumptions are verified.
  - \* Check that  $\hat{\beta}_0$  is significantly different from zero at any level  $\alpha$ .
  - \* For which  $\alpha = 0.10, 0.05, 0.01$  is  $\hat{\beta}_1$  significantly different from zero?

## 2.5 Prediction

The forecast of  $Y$  from  $\mathbf{X} = \mathbf{x}$  (this is,  $X_1 = x_1, \dots, X_p = x_p$ ) is approached by two different ways:

1. Inference on the **conditional mean** of  $Y$  given  $\mathbf{X} = \mathbf{x}$ ,  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ . This is a deterministic quantity, which equals  $\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$ .
2. Prediction of the **conditional response**  $Y|\mathbf{X} = \mathbf{x}$ . This is a random variable distributed as  $\mathcal{N}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p, \sigma^2)$ .

Similarities and differences in the prediction of the conditional mean  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$  and conditional response  $Y|\mathbf{X} = \mathbf{x}$ :

- *Similarities.* The estimate is the same,  $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_p x_p$ . Both CI are centered in  $\hat{y}$ .
- *Differences.*  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$  is deterministic and  $Y|\mathbf{X} = \mathbf{x}$  is random. The prediction of the latter is more noisy, because it has to take into account the variability of  $Y$ . Therefore, the variance is larger for the prediction of  $Y|\mathbf{X} = \mathbf{x}$  than for the prediction of  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ . This has a direct consequence on the length of the prediction intervals, which are longer for  $Y|\mathbf{X} = \mathbf{x}$  than for  $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ .

The inspection of the CIs for the conditional mean and conditional response in the simple linear model help to fix ideas:

- The  $100(1 - \alpha)\%$  CI for the *conditional mean*  $\beta_0 + \beta_1 x$  is

$$\left( \hat{y} \pm t_{n-2:\alpha/2} \sqrt{\frac{\hat{\sigma}^2}{n} \left( 1 + \frac{(x - \bar{x})^2}{s_x^2} \right)} \right). \quad (2.18)$$

- The  $100(1 - \alpha)\%$  CI for the *conditional response*  $Y|X = x$  is

$$\left( \hat{y} \pm t_{n-2:\alpha/2} \sqrt{\hat{\sigma}^2 + \frac{\hat{\sigma}^2}{n} \left( 1 + \frac{(x - \bar{x})^2}{s_x^2} \right)} \right). \quad (2.19)$$

Notice the dependence of both CIs in  $x$ ,  $n$ , and  $\sigma^2$ , each of them with a clear effect on the resulting length of the interval. Figure 2.5 helps visualize these concepts and the difference between CIs interactively.

Illustration of the CIs for the conditional mean and response. Note how the width of the CIs is influenced by  $x$ , especially for the conditional mean (the conditional response has a constant term affecting the width). Application also available here.

### 2.5.1 Case study application

The prediction and computation of prediction CIs can be done with `predict`. The objects required for `predict` are: first, an `lm` object; second, a `data.frame` containing the locations  $\mathbf{x} = (x_1, \dots, x_p)$  where we want to predict  $\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$ . The prediction is  $\hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_p x_p$  and the CIs returned are either (2.18) or (2.19).



It is mandatory to name the columns of the data frame with the same names of the predictors used in `lm`. Otherwise `predict` will generate an error, see below.

```
# Fit a linear model for the price on WinterRain, HarvestRain, and AGST
modWine4 <- lm(Price ~ WinterRain + HarvestRain + AGST, data = wine)
summary(modWine4)
##
## Call:
## lm(formula = Price ~ WinterRain + HarvestRain + AGST, data = wine)
##
## Residuals:
##      Min      1Q      Median      3Q      Max
## -0.62816 -0.17923  0.02274  0.21990  0.62859
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.9506001  1.9694011 -2.514  0.01940 *
## WinterRain    0.0012820  0.0005765  2.224  0.03628 *
## HarvestRain   -0.0036242  0.0009646 -3.757  0.00103 **
## AGST         0.7123192  0.1087676  6.549 1.11e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3436 on 23 degrees of freedom
## Multiple R-squared:  0.7407, Adjusted R-squared:  0.7069
## F-statistic: 21.9 on 3 and 23 DF,  p-value: 6.246e-07

# Data for which we want a prediction
# Important! You have to name the column with the predictor name!
weather <- data.frame(WinterRain = 500, HarvestRain = 123,
                       AGST = 18)
weatherBad <- data.frame(500, 123, 18)

# Prediction of the mean

# Prediction of the mean at 95% - the defaults
predict(modWine4, newdata = weather)
##           1
## 8.066342
```

```

predict(modWine4, newdata = weatherBad) # Error
## Error in eval(predvars, data, env): object 'WinterRain' not found

# Prediction of the mean with 95% confidence interval (the default)
# CI: (lwr, upr)
predict(modWine4, newdata = weather, interval = "confidence")
##       fit      lwr      upr
## 1 8.066342 7.714178 8.418507
predict(modWine4, newdata = weather, interval = "confidence", level = 0.95)
##       fit      lwr      upr
## 1 8.066342 7.714178 8.418507

# Other levels
predict(modWine4, newdata = weather, interval = "confidence", level = 0.90)
##       fit      lwr      upr
## 1 8.066342 7.774576 8.358108
predict(modWine4, newdata = weather, interval = "confidence", level = 0.99)
##       fit      lwr      upr
## 1 8.066342 7.588427 8.544258

# Prediction of the response

# Prediction of the mean at 95% - the defaults
predict(modWine4, newdata = weather)
##       1
## 1 8.066342

# Prediction of the response with 95% confidence interval
# CI: (lwr, upr)
predict(modWine4, newdata = weather, interval = "prediction")
##       fit      lwr      upr
## 1 8.066342 7.273176 8.859508
predict(modWine4, newdata = weather, interval = "prediction", level = 0.95)
##       fit      lwr      upr
## 1 8.066342 7.273176 8.859508

# Other levels
predict(modWine4, newdata = weather, interval = "prediction", level = 0.90)
##       fit      lwr      upr
## 1 8.066342 7.409208 8.723476
predict(modWine4, newdata = weather, interval = "prediction", level = 0.99)
##       fit      lwr      upr
## 1 8.066342 6.989951 9.142733

# Predictions for several values
weather2 <- data.frame(WinterRain = c(500, 200), HarvestRain = c(123, 200),
                        AGST = c(17, 18))
predict(modWine4, newdata = weather2, interval = "prediction")
##       fit      lwr      upr
## 1 7.354023 6.613835 8.094211
## 2 7.402691 6.533945 8.271437

```

For the `wine` dataset, do the following:

For the `wine` dataset, do the following:

- Regress `WinterRain` on `HarvestRain` and `AGST`. Name the fitted model `modExercise`.
- Compute the estimate for the conditional mean of `WinterRain` for `HarvestRain = 123.0` and `AGST = 16.15`. What is the CI at  $\alpha = 0.01$ ?
- Compute the estimate for the conditional response for `HarvestRain = 125.0` and `AGST = 15`. What is the CI at  $\alpha = 0.10$ ?
- Check that `modExercise$fitted.values` is the same as `predict(modExercise, newdata = data.frame(HarvestRain = wine$HarvestRain, AGST = wine$AGST))`. Why is this so?

## 2.6 ANOVA

The variance of the error,  $\sigma^2$ , plays a fundamental role in the inference for the model coefficients and prediction. 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).

An important fact to highlight prior to introducing the ANOVA decomposition is that  $\bar{Y} = \hat{Y}$ . This is an important result that can be checked if we use matrix notation. The ANOVA decomposition considers the following measures of variation related with the response:

- $\text{SST} := \sum_{i=1}^n (Y_i - \bar{Y})^2$ , the **total sum of squares**. This is the *total variation* of  $Y_1, \dots, Y_n$ , since  $\text{SST} = ns_y^2$ , where  $s_y^2$  is the sample variance of  $Y_1, \dots, Y_n$ .
- $\text{SSR} := \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$ , the **regression sum of squares**. This is the variation explained by the regression plane, that is, *the variation from  $\bar{Y}$  that is explained by the estimated conditional mean  $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \dots + \hat{\beta}_p X_{ip}$* .  $\text{SSR} = ns_{\hat{y}}^2$ , where  $s_{\hat{y}}^2$  is the sample variance of  $\hat{Y}_1, \dots, \hat{Y}_n$ .
- $\text{SSE} := \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$ , the **sum of squared errors**<sup>6</sup>. Is the variation around the conditional mean. Recall that  $\text{SSE} = \sum_{i=1}^n \hat{\varepsilon}_i^2 = (n - p - 1)\hat{\sigma}^2$ , where  $\hat{\sigma}^2$  is the sample variance of  $\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_n$ .

The ANOVA decomposition is:

$$\underbrace{\text{SST}}_{\text{Variation of } Y'_i \text{'s}} = \underbrace{\text{SSR}}_{\text{Variation of } \hat{Y}'_i \text{'s}} + \underbrace{\text{SSE}}_{\text{Variation of } \hat{\varepsilon}'_i \text{'s}} \quad (2.20)$$

or, equivalently (dividing by  $n$  in (2.20)),

$$\underbrace{s_y^2}_{\text{Variance of } Y'_i \text{'s}} = \underbrace{s_{\hat{y}}^2}_{\text{Variance of } \hat{Y}'_i \text{'s}} + \underbrace{(n - p - 1)/n \times \hat{\sigma}^2}_{\text{Variance of } \hat{\varepsilon}'_i \text{'s}}.$$

The graphical interpretation of (2.20) when  $p = 1$  is shown in Figures 2.7 and 2.6.

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

The ANOVA table summarizes the decomposition of the variance:

<sup>6</sup>SSE and RSS are two names for the same quantity (that appears in different contexts):  $\text{SSE} = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} - \dots - \hat{\beta}_p X_{ip})^2 = \text{RSS}(\hat{\beta})$ .

| Degrees of freedom    | Sum Squares | Mean Squares               | $F$ -value                                | $p$ -value |
|-----------------------|-------------|----------------------------|---|------------|
| Predictors $p$        | SSR         | $\frac{\text{SSR}}{p}$     | $\frac{\text{SSR}/p}{\text{SSE}/(n-p-1)}$ | $p$        |
| Residuals $n - p - 1$ | SSE         | $\frac{\text{SSE}}{n-p-1}$ |   |            |

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

$$H_0 : \beta_1 = \dots = \beta_p = 0 \quad \text{vs.} \quad H_1 : \beta_j \neq 0 \text{ for any } j,$$

that is, the hypothesis of *no linear dependence of  $Y$  on  $X_1, \dots, X_p$*  (the plane is completely flat, with no inclination). This is the so-called *F-test* and, if  $H_0$  is rejected, allows to conclude that **at least one  $\beta_j$  is significantly different from zero**. It happens that

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

where  $F_{p,n-p-1}$  represents the *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 SSR will be close to zero (little variation is explained by the regression model since  $\hat{\beta} \approx \mathbf{0}$ ).



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 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>7</sup>:

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

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

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

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

<sup>7</sup>More complex – included here just for clarification of the `anova`'s output.

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

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.

In order to compute the simplified ANOVA table, we need to rely on an ad-hoc *function*. The function takes as input a fitted `lm`:

```
# This function computes the simplified anova from a linear model
simpleAnova <- function(object, ...) {

  # Compute anova table
  tab <- anova(object, ...)

  # Obtain number of predictors
  p <- nrow(tab) - 1

  # Add predictors row
  predictorsRow <- colSums(tab[1:p, 1:2])
  predictorsRow <- c(predictorsRow, predictorsRow[2] / predictorsRow[1])

  # F-quantities
  Fval <- predictorsRow[3] / tab[p + 1, 3]
  pval <- pf(Fval, df1 = p, df2 = tab$Df[p + 1], lower.tail = FALSE)
  predictorsRow <- c(predictorsRow, Fval, pval)

  # Simplified table
  tab <- rbind(predictorsRow, tab[p + 1, ])
  row.names(tab)[1] <- "Predictors"
  return(tab)

}
```

### 2.6.1 Case study application

Let's compute the ANOVA decomposition of `modWine1` and `modWine2` to test the existence of linear dependence.

```
# Models
modWine1 <- lm(Price ~ ., data = wine)
modWine2 <- lm(Price ~ . - FrancePop, data = wine)

# Simplified table
simpleAnova(modWine1)
## Analysis of Variance Table
##
## Response: Price
##             Df Sum Sq Mean Sq F value    Pr(>F)
## Predictors  5 8.6671 1.73343 20.188 2.232e-07 ***
## Residuals 21 1.8032 0.08587
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
simpleAnova(modWine2)
```

```

## Analysis of Variance Table
##
## Response: Price
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Predictors  4 8.6645 2.16613   26.39 4.057e-08 ***
## Residuals  22 1.8058 0.08208
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# The null hypothesis of no linear dependence is emphatically rejected in
# both models

# R's ANOVA table - warning this is now what we saw in lessons
anova(modWine1)
## Analysis of Variance Table
##
## Response: Price
##           Df Sum Sq Mean Sq F value    Pr(>F)
## WinterRain   1 0.1905  0.1905  2.2184 0.1512427
## AGST         1 5.8989  5.8989 68.6990 4.645e-08 ***
## HarvestRain  1 1.6662  1.6662 19.4051 0.0002466 ***
## Age          1 0.9089  0.9089 10.5852 0.0038004 **
## FrancePop    1 0.0026  0.0026  0.0305 0.8631279
## Residuals   21 1.8032  0.0859
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```



Compute the ANOVA table for the regression `Price ~ WinterRain + AGST + HarvestRain + Age` in the `wine` dataset. Check that the  $p$ -value for the  $F$ -test given in `summary` and by `simpleAnova` are the same.



For the `y6 ~ x6` and `y7 ~ x7` in the `assumptions` dataset, compute their ANOVA tables. Check that the  $p$ -values of the  $t$ -test for  $\beta_1$  and the  $F$ -test are the same (any explanation of why this is so?).

## 2.7 Model fit

### 2.7.1 The $R^2$

The *coefficient of determination*  $R^2$  is closely related with the ANOVA decomposition. It is defined as

$$R^2 := \frac{\text{SSR}}{\text{SST}} = \frac{\text{SSR}}{\text{SSR} + \text{SSE}} = \frac{\text{SSR}}{\text{SSR} + (n - p - 1)\hat{\sigma}^2}.$$

$R^2$  measures the **proportion of variation** of the response variable  $Y$  that is **explained** by the predictors  $X_1, \dots, X_p$  through the regression. Intuitively,  $R^2$  measures the *tightness of the data cloud around the regression plane*. Check in Figure 2.6 how changing the value of  $\sigma^2$  (not  $\hat{\sigma}^2$ , but  $\hat{\sigma}^2$  is obviously dependent on  $\sigma^2$ ) affects the  $R^2$ .

$R^2$  is intimately related with the sample correlation coefficient. For example, if  $p = 1$ , then it can be seen (exercise below) that  $R^2 = r_{xy}^2$ . More importantly,  $R^2 = r_{\hat{y}\hat{y}}^2$  for any  $p$ , that is, the square of the sample correlation coefficient between  $Y_1, \dots, Y_n$  and  $\hat{Y}_1, \dots, \hat{Y}_n$  is  $R^2$ , a fact that is not immediately evident. Let's see check this fact when  $p = 1$  by relying on  $R^2 = r_{xy}^2$ . First, by the form of  $\hat{\beta}_0$  given in (2.3),

$$\begin{aligned}\hat{Y}_i &= \hat{\beta}_0 + \hat{\beta}_1 X_i \\ &= (\bar{Y} - \hat{\beta}_1 X_i) + \hat{\beta}_1 X_i \\ &= \bar{Y} + \hat{\beta}_1 (X_i - \bar{X}).\end{aligned}\tag{2.21}$$

Then, we replace (2.21) in

$$\begin{aligned}r_{\hat{y}\hat{y}}^2 &= \frac{s_{\hat{y}\hat{y}}^2}{s_y^2 s_{\hat{y}}^2} \\ &= \frac{\left(\sum_{i=1}^n (Y_i - \bar{Y})(\hat{Y}_i - \bar{Y})\right)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2 \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2} \\ &= \frac{\left(\sum_{i=1}^n (Y_i - \bar{Y})(\bar{Y} + \hat{\beta}_1(X_i - \bar{X}) - \bar{Y})\right)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2 \sum_{i=1}^n (\bar{Y} + \hat{\beta}_1(X_i - \bar{X}) - \bar{Y})^2} \\ &= r_{xy}^2.\end{aligned}$$

and as a consequence  $r_{\hat{y}\hat{y}}^2 = r_{xy}^2 = R^2$  when  $p = 1$ .



Show that  $R^2 = r_{xy}^2$  when  $p = 1$ . Hint: start from the definition of  $R^2$  and use (2.3) to arrive to  $r_{xy}^2$ .

Trusting the  $R^2$  blindly can lead to catastrophic conclusions, since the model may not be correct. Here are a couple of counterexamples of a linear regression performed in a data that clearly does not satisfy the assumptions discussed in Section 2.3, but despite that, still has a large  $R^2$ . As a consequence, **inference built on the validity of the assumptions** (which do not hold!) **will be problematic, no matter what is the value of  $R^2$** . For example, recall how biased the predictions will be for  $x = 0.35$  and  $x = 0.65$ !

```
# Simple linear model

# Create data that:
# 1) does not follow a linear model
# 2) the error is heteroskedastic
x <- seq(0.15, 1, 1 = 100)
set.seed(123456)
eps <- rnorm(n = 100, sd = 0.25 * x^2)
y <- 1 - 2 * x * (1 + 0.25 * sin(4 * pi * x)) + eps

# Great R^2!?
reg <- lm(y ~ x)
summary(reg)
##
## Call:
```

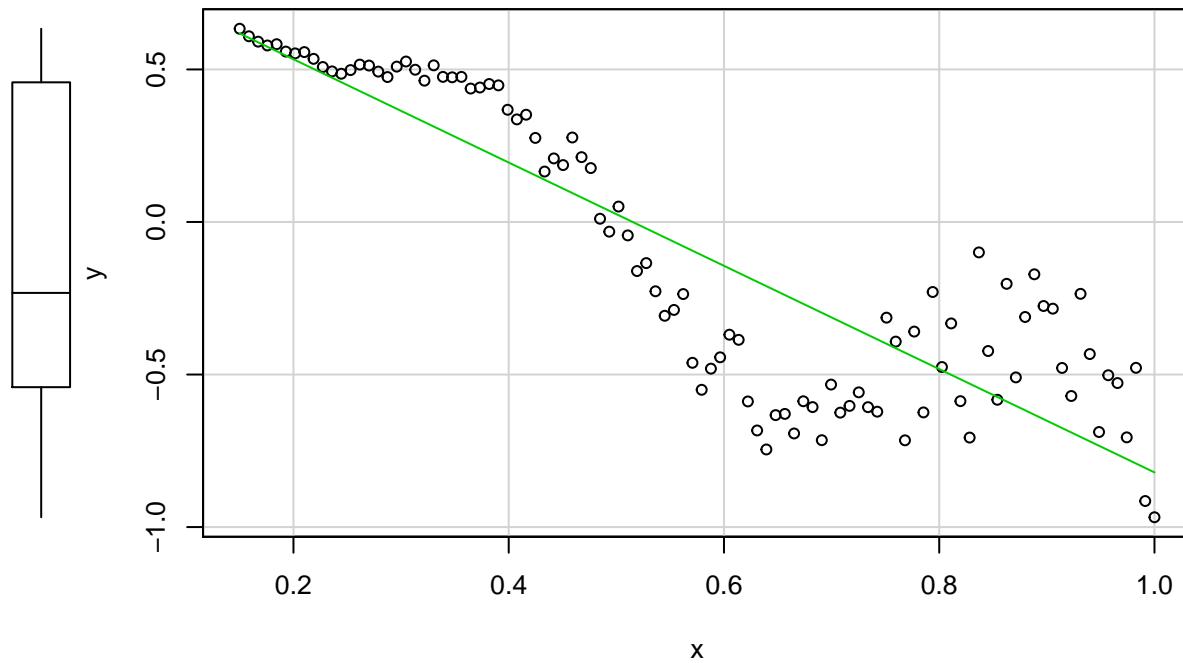
```

## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.53525 -0.18020  0.02811  0.16882  0.46896
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.87190   0.05860 14.88   <2e-16 ***
## x          -1.69268   0.09359 -18.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.232 on 98 degrees of freedom
## Multiple R-squared:  0.7695, Adjusted R-squared:  0.7671
## F-statistic: 327.1 on 1 and 98 DF,  p-value: < 2.2e-16

# scatterplot is a quick alternative to
# plot(x, y)
# abline(coef = reg$coef, col = 3)

# But prediction is obviously problematic
scatterplot(y ~ x, smooth = FALSE)

```



```

# Multiple linear model

# Create data that:
# 1) does not follow a linear model
# 2) the error is heteroskedastic

```

```

x1 <- seq(0.15, 1, l = 100)
set.seed(123456)
x2 <- runif(100, -3, 3)
eps <- rnorm(n = 100, sd = 0.25 * x1^2)
y <- 1 - 3 * x1 * (1 + 0.25 * sin(4 * pi * x1)) + 0.25 * cos(x2) + eps

# Great R^2!?
reg <- lm(y ~ x1 + x2)
summary(reg)
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##       Min     1Q Median     3Q    Max
## -0.78737 -0.20946  0.01031  0.19652  1.05351
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 0.788812  0.096418  8.181  1.1e-12 ***
## x1          -2.540073  0.154876 -16.401 < 2e-16 ***
## x2          0.002283  0.020954   0.109   0.913    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3754 on 97 degrees of freedom
## Multiple R-squared:  0.744,  Adjusted R-squared:  0.7388
## F-statistic:  141 on 2 and 97 DF,  p-value: < 2.2e-16

```

We can visualize the fit of the latter multiple linear model since we are in  $p = 2$ .

```

# But prediction is obviously problematic
scatter3d(y ~ x1 + x2, fit = "linear")

```



A large  $R^2$  means *nothing* in terms of inference if the **assumptions of the model do not hold**.  $R^2$  is the proportion of variance of  $Y$  explained by  $X_1, \dots, X_p$ , but, of course, *only when the linear model is correct*.



Remember that:

- $R^2$  does not measure the correctness of a linear model but its **usefulness**, assuming the model is correct.
- $R^2$  is the proportion of variance of  $Y$  explained by  $X_1, \dots, X_p$ , but, of course, *only when the linear model is correct*.

We finalize by pointing out a nice connection between the  $R^2$ , the ANOVA decomposition and the least squares estimator  $\hat{\beta}$ :



The ANOVA decomposition gives another interpretation of the least-squares estimates:  $\hat{\beta}$  **are the estimated coefficients that maximize the  $R^2$**  (among all the possible estimates we could think

about). To see this, recall that

$$R^2 = \frac{\text{SSR}}{\text{SST}} = \frac{\text{SST} - \text{SSE}}{\text{SST}} = \frac{\text{SST} - \text{RSS}(\hat{\beta})}{\text{SST}},$$

so if  $\text{RSS}(\hat{\beta}) = \min_{\beta \in \mathbb{R}^{p+1}} \text{RSS}(\beta)$ , then  $R^2$  is maximal for  $\hat{\beta}$ !

### 2.7.2 The $R_{\text{Adj}}^2$

As we saw, these are equivalent forms for  $R^2$ :

$$\begin{aligned} R^2 &:= \frac{\text{SSR}}{\text{SST}} = \frac{\text{SST} - \text{SSE}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}} \\ &= 1 - \frac{\hat{\sigma}^2}{\text{SST}} \times (n - p - 1). \end{aligned} \quad (2.22)$$

The SSE on the numerator always decreases as more predictors are added to the model, even if these are not significant. As a consequence, the  $R^2$  **always increases with  $p$** . Why is this so? Intuitively, because the complexity – hence the flexibility – of the model increases when we use more predictors to explain  $Y$ . Mathematically, because when  $p$  approaches  $n - 1$  the second term in (2.22) is reduced and, as a consequence,  $R^2$  grows.

The *adjusted*  $R^2$  is an important quantity specifically designed to cover this  $R^2$ 's flaw, which is ubiquitous in multiple linear regression. The purpose is to have a better tool for **comparing models without systematically favoring complexer models**. This alternative coefficient is defined as

$$\begin{aligned} R_{\text{Adj}}^2 &= 1 - \frac{\text{SSE}/(n - p - 1)}{\text{SST}/(n - 1)} = 1 - \frac{\text{SSE}}{\text{SST}} \times \frac{n - 1}{n - p - 1} \\ &= 1 - \frac{\hat{\sigma}^2}{\text{SST}} \times (n - 1). \end{aligned} \quad (2.23)$$

The  $R_{\text{Adj}}^2$  is independent of  $p$ , at least explicitly. If  $p = 1$  then  $R_{\text{Adj}}^2$  is *almost*  $R^2$  (practically identical if  $n$  is large). Both (2.22) and (2.23) are quite similar except for the last factor, which in the former does not depend on  $p$ . Therefore, (2.23) will only increase if  $\hat{\sigma}^2$  is reduced with  $p$  – in other words, if the new variables contribute in the reduction of variability around the regression plane.

The different behavior between  $R^2$  and  $R_{\text{Adj}}^2$  can be visualized by a small simulation. Suppose that we generate a random dataset, with  $n = 200$  observations of a response  $Y$  and two predictors  $X_1, X_2$ . This is, the sample  $\{(X_{i1}, X_{i2}, Y_i)\}_{i=1}^n$  with

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, 1).$$

To this data, we add 196 *garbage* predictors that are completely independent from  $Y$ . Therefore, we end up with  $p = 198$  predictors. Now we compute the  $R^2(j)$  and  $R_{\text{Adj}}^2(j)$  for the models

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_j X_j + \varepsilon,$$

with  $j = 1, \dots, p$  and we plot them as the curves  $(j, R^2(j))$  and  $(j, R_{\text{Adj}}^2(j))$ . Since  $R^2$  and  $R_{\text{Adj}}^2$  are **random variables**, we repeat the procedure 100 times to have a measure of the variability.

Figure 2.8 contains the results of this experiment. 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,  $R_{\text{Adj}}^2$  only increases

in the first two variables and then is flat on average, but it has a huge variability when  $p$  approaches  $n - 2$ . This is a consequence of the explosive variance of  $\hat{\sigma}^2$  in that *degenerate case*. The experiment demonstrates that  $R_{\text{Adj}}^2$  is more adequate than the  $R^2$  for evaluating the fit of a multiple linear regression.

An example of a simulated dataset considered in the experiment of Figure 2.8:

```
# Generate data
p <- 198
n <- 200
set.seed(3456732)
beta <- c(0.5, -0.5, rep(0, p - 2))
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
Y <- drop(X %*% beta + rnorm(n, sd = 3))
data <- data.frame(y = Y, x = X)

# Regression on the two meaningful predictors
summary(lm(y ~ x.1 + x.2, data = data))

# Adding 20 garbage variables
# R^2 increases and adjusted R^2 decreases
summary(lm(y ~ X[, 1:22], data = data))
```



The  $R_{\text{Adj}}^2$  no longer measures the proportion of variation of  $Y$  explained by the regression, but the result of *correcting this proportion by the number of predictors employed*. As a consequence of this,  $R_{\text{Adj}}^2 \leq 1$  but **it can be negative!**

The next code illustrates a situation where we have two predictors completely independent from the response. The fitted model has a negative  $R_{\text{Adj}}^2$ .

```
# Three independent variables
set.seed(234599)
x1 <- rnorm(100)
x2 <- rnorm(100)
y <- 1 + rnorm(100)

# Negative adjusted R^2
summary(lm(y ~ x1 + x2))
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##      Min      1Q      Median      3Q      Max
## -3.5081 -0.5021 -0.0191  0.5286  2.4750
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.97024   0.10399  9.330 3.75e-15 ***
## x1          0.09003   0.10300  0.874   0.384    
## x2         -0.05253   0.11090 -0.474   0.637    
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.034 on 97 degrees of freedom
```

```
## Multiple R-squared:  0.009797,  Adjusted R-squared:  -0.01062
## F-statistic: 0.4799 on 2 and 97 DF,  p-value: 0.6203
```



For the previous example, construct more predictors ( $x_3, x_4, \dots$ ) that are independent from  $y$  and check that when the predictors are added to the model, the  $R_{\text{Adj}}^2$  decreases and the  $R^2$  increases.

### 2.7.3 Case study application

Coming back to the case study, we have studied so far three models:

```
# Fit models
modWine1 <- lm(Price ~ ., data = wine)
modWine2 <- lm(Price ~ . - FrancePop, data = wine)
modWine3 <- lm(Price ~ Age + WinterRain, data = wine)

# Summaries
summary(modWine1)
##
## Call:
## lm(formula = Price ~ ., data = wine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46541 -0.24133  0.00413  0.18974  0.52495
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.343e+00 7.697e+00 -0.304 0.76384
## WinterRain   1.153e-03 4.991e-04  2.311 0.03109 *
## AGST        6.144e-01 9.799e-02  6.270 3.22e-06 ***
## HarvestRain -3.837e-03 8.366e-04 -4.587 0.00016 ***
## Age         1.377e-02 5.821e-02  0.237 0.81531
## FrancePop   -2.213e-05 1.268e-04 -0.175 0.86313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.293 on 21 degrees of freedom
## Multiple R-squared:  0.8278, Adjusted R-squared:  0.7868
## F-statistic: 20.19 on 5 and 21 DF,  p-value: 2.232e-07
summary(modWine2)
##
## Call:
## lm(formula = Price ~ . - FrancePop, data = wine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46024 -0.23862  0.01347  0.18601  0.53443
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.6515703 1.6880876 -2.163 0.04167 *
```

```

## WinterRain  0.0011667  0.0004820  2.420  0.02421 *
## AGST        0.6163916  0.0951747  6.476  1.63e-06 ***
## HarvestRain -0.0038606  0.0008075 -4.781  8.97e-05 ***
## Age         0.0238480  0.0071667  3.328  0.00305 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2865 on 22 degrees of freedom
## Multiple R-squared:  0.8275, Adjusted R-squared:  0.7962
## F-statistic: 26.39 on 4 and 22 DF,  p-value: 4.057e-08
summary(modWine3)
##
## Call:
## lm(formula = Price ~ Age + WinterRain, data = wine)
##
## Residuals:
##       Min     1Q   Median     3Q    Max
## -0.88964 -0.51421 -0.00066  0.43103  1.06897
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 5.9830427  0.5993667  9.982 5.09e-10 ***
## Age         0.0360559  0.0137377  2.625  0.0149 *  
## WinterRain  0.0007813  0.0008780  0.890  0.3824    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5769 on 24 degrees of freedom
## Multiple R-squared:  0.2371, Adjusted R-squared:  0.1736
## F-statistic:  3.73 on 2 and 24 DF,  p-value: 0.03884

```

modWine2 is the model with the largest  $R^2_{\text{Adj}}$ . It is a model that explains the 82.75% of the variability in a non-redundant way and with all its coefficients significant. Therefore, we have a formula for effectively explaining and predicting the quality of a vintage (answers Q1). Prediction and, importantly, quantification of the uncertainty in the prediction, can be done with `predict`.

Furthermore, the interpretation of `modWine2` agrees with well-known facts in viticulture that make perfect sense (Q2):

- Higher temperatures are associated with better quality (higher priced) wine.
- Rain before the growing season is good for the wine quality, but during harvest is bad.
- The quality of the wine improves with the age.

Although these were known facts, keep in mind that the this model allows to *quantify the effect of each variable on the wine quality* and provides us with a precise way of *predicting the quality of future vintages*.

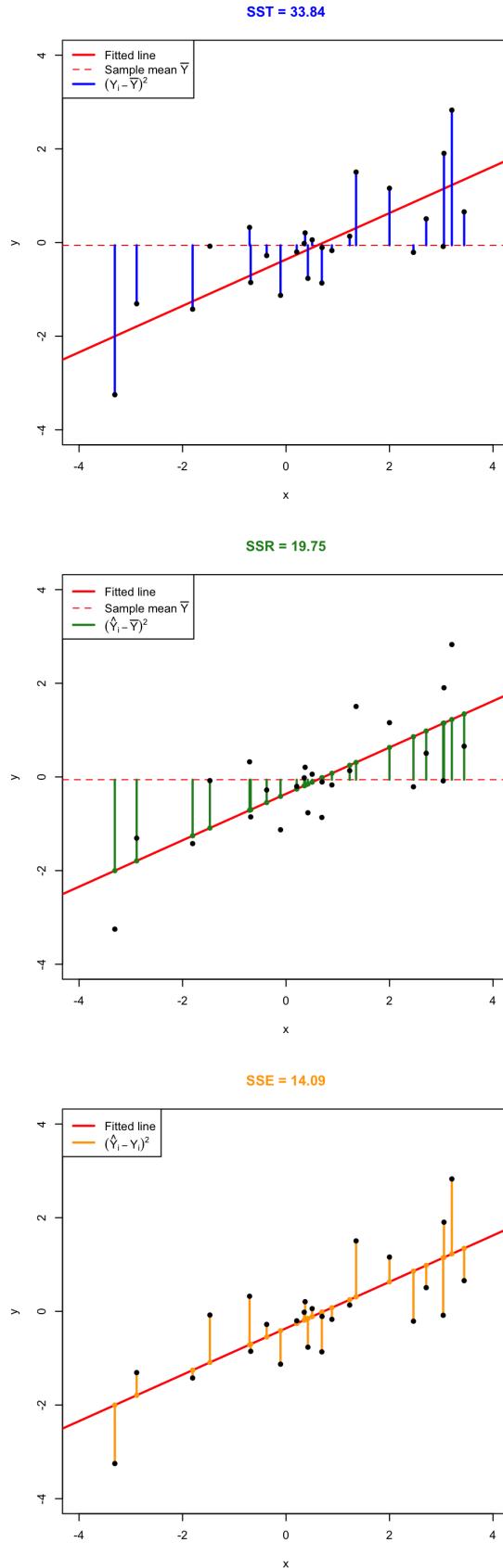


Figure 2.7: Visualization of the ANOVA decomposition. SST measures the variation of  $Y_1, \dots, Y_n$  with respect to  $\bar{Y}$ . SST measures the variation with respect to the conditional means,  $\hat{\beta}_0 + \hat{\beta}_1 X_i$ . SSE collects the variation of the residuals.

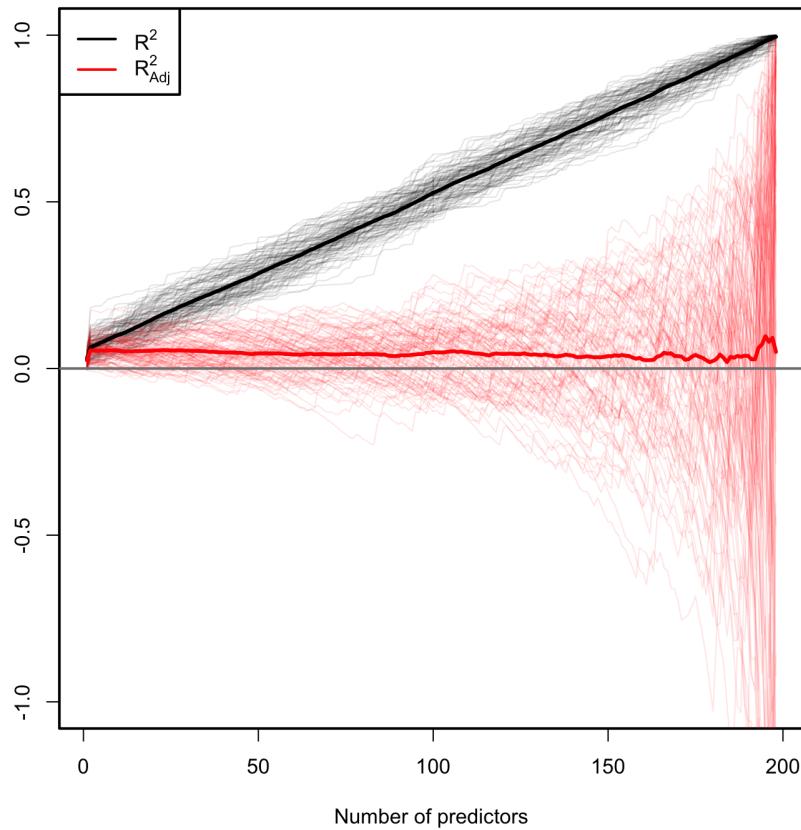


Figure 2.8: Comparison of  $R^2$  and  $R^2_{\text{Adj}}$  for  $n = 200$  and  $p$  ranging from 1 to 198.  $M = 100$  datasets were simulated with **only the first two** predictors being significant. The thicker curves are the mean of each color's curves.

# Chapter 3

## Linear models II: model selection, extensions, and diagnostics

Given the response  $Y$  and the predictors  $X_1, \dots, X_p$ , many linear models can be built for predicting and explaining  $Y$ . In this chapter we will see how to address the problem of selecting the *best subset of predictors*  $X_1, \dots, X_p$  for explaining  $Y$ . Among others, we will also see how to extend the linear model to account for nonlinear relations between  $Y$  and  $X_1, \dots, X_p$ , and how to check whether the assumptions of the model are realistic in practice.

### 3.1 Case study: *Housing values in Boston*

This case study is motivated by Harrison and Rubinfeld (1978), who proposed an *hedonic model* for determining the willingness of house buyers to pay for clean air. An hedonic model is a model that decomposes the price of an item into separate components that determine its price. For example, an hedonic model for the price of a house may decompose its price into the house characteristics, the kind of neighborhood, and the location. The study of Harrison and Rubinfeld (1978) employed data from the Boston metropolitan area, containing 560 suburbs and 14 variables. The **Boston** dataset is available through the file **Boston.xlsx** file (download) and through the dataset **Boston** in the **MASS** package.

The description of the related variables can be found in **?Boston** and Harrison and Rubinfeld (1978)<sup>1</sup>, but we summarize here the most important ones as they appear in **Boston**. They are aggregated into five topics:

- *Dependent* variable: **medv**, the median value of owner-occupied homes (in thousands of dollars).
- *Structural* variables indicating the house characteristics: **rm** (average number of rooms “in owner units”) and **age** (proportion of owner-occupied units built prior to 1940).
- *Neighborhood* variables: **crim** (crime rate), **zn** (proportion of residential areas), **indus** (proportion of non-retail business area), **chas** (river limitation), **tax** (cost of public services in each community), **ptratio** (pupil-teacher ratio), **black** (variable  $1000(B - 0.63)^2$ , where  $B$  is the black proportion of population – low and high values of  $B$  increase housing prices) and **lstat** (percent of lower status of the population).
- *Accesibility* variables: **dis** (distances to five Boston employment centers) and **rad** (accessibility to radial highways – larger index denotes better accessibility).
- *Air pollution* variable: **nox**, the annual concentration of nitrogen oxide (in parts per ten million).

We begin by importing the data:

---

<sup>1</sup>But be aware of the changes in units for **medv**, **black**, **lstat**, and **nox**.

```
# Read data
library(readxl)
Boston <- read_excel(path = "Boston.xlsx", sheet = 1, col_names = TRUE)
```

A summary of the data is shown below:

```
summary(Boston)
##      crim             zn             indus            chas
##  Min.   : 0.00632   Min.   : 0.00   Min.   : 0.46   Min.   :0.00000
##  1st Qu.: 0.08204   1st Qu.: 0.00   1st Qu.: 5.19   1st Qu.:0.00000
##  Median : 0.25651   Median : 0.00   Median : 9.69   Median :0.00000
##  Mean   : 3.61352   Mean   :11.36   Mean   :11.14   Mean   :0.06917
##  3rd Qu.: 3.67708   3rd Qu.:12.50   3rd Qu.:18.10   3rd Qu.:0.00000
##  Max.   :88.97620   Max.   :100.00  Max.   :27.74   Max.   :1.00000
##      nox              rm              age              dis
##  Min.   :0.3850   Min.   :3.561   Min.   : 2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.:45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median :77.50   Median : 3.207
##  Mean   :0.5547   Mean   :6.285   Mean   :68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.:94.08   3rd Qu.: 5.188
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00  Max.   :12.127
##      rad              tax              ptratio            black
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   : 0.32
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
##      lstat             medv
##  Min.   : 1.73   Min.   : 5.00
##  1st Qu.: 6.95   1st Qu.:17.02
##  Median :11.36   Median :21.20
##  Mean   :12.65   Mean   :22.53
##  3rd Qu.:16.95   3rd Qu.:25.00
##  Max.   :37.97   Max.   :50.00
```

The two goals of this case study are:

- Q1. *Quantify the influence of the predictor variables in the housing prices.*
- Q2. *Obtain the “best possible” model for decomposing the housing prices and interpret it.*

We begin by making an exploratory analysis of the data with a matrix scatterplot. Since the number of variables is high, we opt to plot only five variables: `crim`, `dis`, `medv`, `nox`, and `rm`. Each of them represents the five topics in which variables were classified.

```
scatterplotMatrix(~ crim + dis + medv + nox + rm, reg.line = lm, smooth = FALSE,
                  spread = FALSE, span = 0.5, ellipse = FALSE, levels = c(.5, .9),
                  id.n = 0, diagonal = 'density', data = Boston)
```

The diagonal panels are showing an estimate of the unknown density of each variable. Note the peculiar distribution of `crim`, very concentrated at zero, and the asymmetry in `medv`, with a second mode associated to the most expensive properties. Inspecting the individual panels, it is clear that some nonlinearity exists in the data and that some predictors are going to be more important than others (recall that we have plotted just a subset of all the predictors).

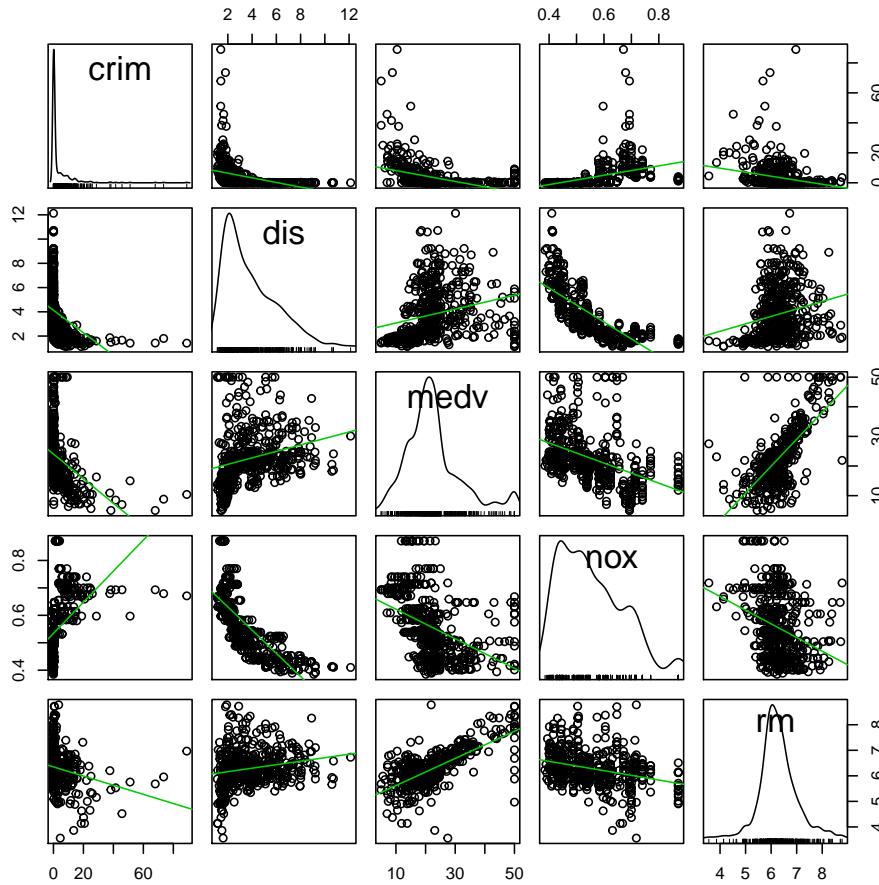


Figure 3.1: Scatterplot matrix for `crim`, `dis`, `medv`, `nox`, and `rm` from the Boston dataset.

## 3.2 Model selection

In Chapter 2 we briefly saw that **the inclusion of more predictors is not for free**: there is a price to pay in terms of more variability in the coefficients estimates, harder interpretation, and possible inclusion of highly-dependent predictors. Indeed, there is a **maximum number of predictors**  $p$  that can be considered in a linear model for a sample size  $n$ :  $p \leq n - 2$ . Or equivalently, there is a **minimum sample size**  $n$  required for fitting a model with  $p$  predictors:  $n \geq p + 2$ .

The interpretation of this fact is simple if we think on the geometry for  $p = 1$  and  $p = 2$ :

- If  $p = 1$ , we need at least  $n = 2$  points to uniquely fit a line. However, this line gives no information on the vertical variation around it and hence  $\hat{\sigma}^2$  can not be estimated (applying its formula, we would have  $\hat{\sigma}^2 = \infty$ ). Therefore we need at least  $n = 3$  points, or in other words  $n \geq p + 2 = 3$ .
- If  $p = 2$ , we need at least  $n = 3$  points to uniquely fit a plane. But this plane gives no information on the variation of the data around it and hence  $\hat{\sigma}^2$  cannot be estimated. Therefore we need  $n \geq p + 2 = 4$ .

Another interpretation is the following:

The fitting of a linear model with  $p$  predictors involves the estimation the  $p+2$  parameters  $(\beta, \sigma^2)$  from  $n$  data points. The closer  $p+2$  and  $n$  are, the more variable the estimates  $(\hat{\beta}, \hat{\sigma}^2)$  will be, since less information is available for estimating each one. In the limit case  $n = p + 2$ , each sample point determines a parameter estimate.

The *degrees of freedom*  $n - p - 1$  quantify the increase in the variability of  $(\hat{\beta}, \hat{\sigma}^2)$  when  $n - p - 1$  decreases. For example:

- $t_{n-p-1;\alpha/2}$  appears in (2.16) and influences the length of the CIs for  $\beta_j$ , see (2.17). It also influences the length of the CIs for the prediction. As Figure 3.2 shows, when the degrees of freedom  $n - p - 1$  decrease,  $t_{n-p-1;\alpha/2}$  increases, thus the intervals become wider.
- $\hat{\sigma}^2 = \frac{1}{n-p-1} \sum_{i=1}^n \hat{\varepsilon}_i^2$  influences the  $R^2$  and  $R_{\text{Adj}}^2$ . If no relevant variables are added to the model then  $\sum_{i=1}^n \hat{\varepsilon}_i^2$  will not change substantially. However, the reducing factor  $\frac{1}{n-p-1}$  will decrease as  $p$  augments, inflating  $\hat{\sigma}^2$  and its variance. This is exactly what happened in Figure 2.8.

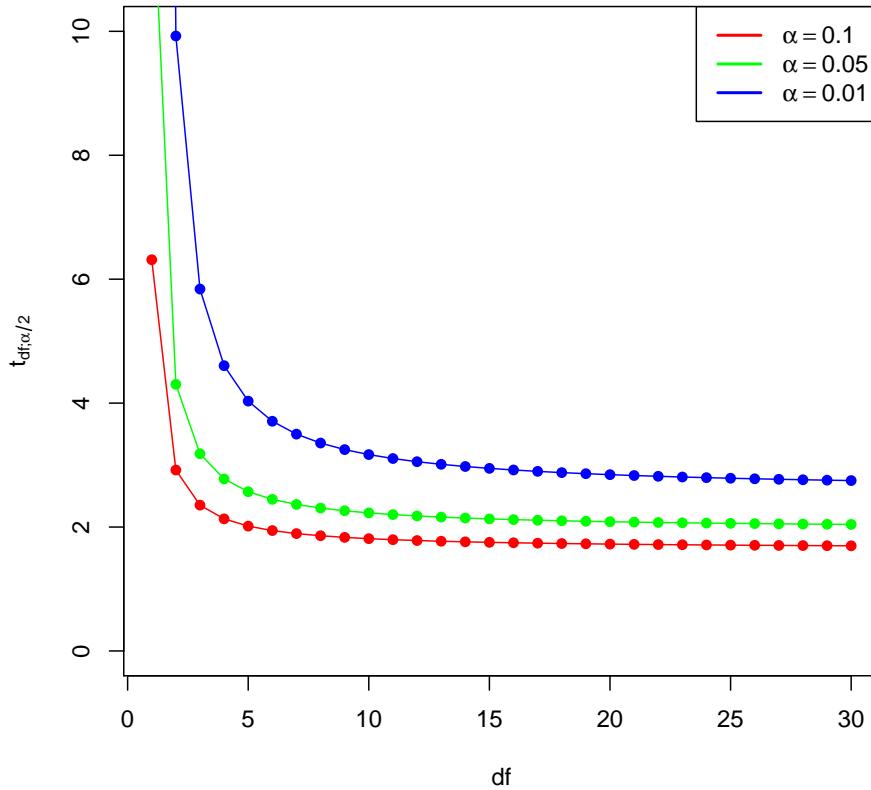


Figure 3.2: Effect of  $df = n - p - 1$  in  $t_{df;\alpha/2}$  for  $\alpha = 0.10, 0.05, 0.01$ .

Now that we have shed more light on the problem of having an excess of predictors, we turn the focus on **selecting the most adequate predictors for a multiple regression model**. This is a challenging task without a unique solution, and what is worse, without a method that is guaranteed to work in all the cases. However, there is a well-established procedure that usually gives good results: the **stepwise model selection**. Its principle is to compare multiple linear regression models with different predictors (and, of course, with the same responses).

Before introducing the method, we need to understand what is an **information criterion**. An information criterion balances the fitness of a model with the number of predictors employed. Hence, it determines objectively the best model as the one that *minimizes the information criterion*. Two common criteria are the *Bayesian Information Criterion* (BIC) and the *Akaike Information Criterion* (AIC). Both are based on a **balance between the model fitness and its complexity**:

$$\text{BIC}(\text{model}) = \underbrace{-2\ell(\text{model})}_{\text{Model fitness}} + \underbrace{\text{npar}(\text{model}) \times \log n}_{\text{Complexity}}, \quad (3.1)$$

where  $\ell(\text{model})$  is the *log-likelihood of the model* (how well the model fits the data) and  $\text{npar}(\text{model})$  is the number of parameters of the model,  $p + 2$  in the case of a multiple linear regression model with  $p$  predictors.

The AIC replaces  $\log n$  by 2 in (3.1) so, compared with BIC, it **penalizes the more complex models less** (recall that  $\log n > 2$  if  $n \geq 8$ ). This is one of the reasons why BIC is preferred by some practitioners for model comparison. Also, because it is *consistent* in selecting the true model: if enough data is provided, the BIC is guaranteed to select the data-generating model among a list of candidate models.

The BIC and AIC can be computed in R through the functions `BIC` and `AIC`. They take a model as the input.

```
# Two models with different predictors
mod1 <- lm(medv ~ age + crim, data = Boston)
mod2 <- lm(medv ~ age + crim + lstat, data = Boston)

# BICs
BIC(mod1)
## [1] 3581.893
BIC(mod2) # Smaller -> better
## [1] 3300.841

# AICs
AIC(mod1)
## [1] 3564.987
AIC(mod2) # Smaller -> better
## [1] 3279.708

# Check the summaries
summary(mod1)
##
## Call:
## lm(formula = medv ~ age + crim, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.940  -4.991  -2.420   2.110  32.033
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 29.80067   0.97078 30.698 < 2e-16 ***
## age         -0.08955   0.01378 -6.499 1.95e-10 ***
## crim        -0.31182   0.04510 -6.914 1.43e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.157 on 503 degrees of freedom
## Multiple R-squared:  0.2166, Adjusted R-squared:  0.2134
## F-statistic: 69.52 on 2 and 503 DF,  p-value: < 2.2e-16
summary(mod2)
##
## Call:
## lm(formula = medv ~ age + crim + lstat, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.133  -3.848  -1.380   1.970  23.644
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```

## (Intercept) 32.82804    0.74774 43.903 < 2e-16 ***
## age         0.03765    0.01225  3.074  0.00223 **
## crim        -0.08262   0.03594 -2.299  0.02193 *
## lstat        -0.99409   0.05075 -19.587 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.147 on 502 degrees of freedom
## Multiple R-squared:  0.5559, Adjusted R-squared:  0.5533
## F-statistic: 209.5 on 3 and 502 DF,  p-value: < 2.2e-16

```

Let's go back to the selection of predictors. If we have  $p$  predictors, a naive procedure would be to check *all the possible* models that can be constructed with them and then select the best one in terms of BIC/AIC. This is the so-called *best subset selection*. The problem is that there are  $2^{p+1}$  possible models! Fortunately, the `stepAIC` procedure helps us navigating this ocean of models by iteratively adding useful predictors and removing the non important ones. The function takes as input a *model employing all the available predictors*. Let's see how it works with the already studied `wine` dataset.

```

# Load data - notice that "Year" is also included
wine <- read.csv(file = "wine.csv", header = TRUE)

```

`stepAIC` takes the argument `k` as 2 (default) or  $\log n$ , where  $n$  is the sample size. With `k = 2` it uses the AIC criterion and with `k = log(n)` it considers the BIC.

```

# Full model
mod <- lm(Price ~ ., data = wine)

# With BIC
modBIC <- stepAIC(mod, k = log(nrow(wine)))
## Start:  AIC=-53.29
## Price ~ Year + WinterRain + AGST + HarvestRain + Age + FrancePop
##
## 
## Step:  AIC=-53.29
## Price ~ Year + WinterRain + AGST + HarvestRain + FrancePop
##
##           Df Sum of Sq   RSS   AIC
## - FrancePop  1   0.0026 1.8058 -56.551
## - Year       1   0.0048 1.8080 -56.519
## <none>          1.8032 -53.295
## - WinterRain 1   0.4585 2.2617 -50.473
## - HarvestRain 1   1.8063 3.6095 -37.852
## - AGST        1   3.3756 5.1788 -28.105
##
## Step:  AIC=-56.55
## Price ~ Year + WinterRain + AGST + HarvestRain
##
##           Df Sum of Sq   RSS   AIC
## <none>          1.8058 -56.551
## - WinterRain  1   0.4809 2.2867 -53.473
## - Year       1   0.9089 2.7147 -48.840
## - HarvestRain 1   1.8760 3.6818 -40.612
## - AGST        1   3.4428 5.2486 -31.039
summary(modBIC)
##
```

```

## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain,
##      data = wine)
##
## Residuals:
##      Min      1Q  Median      3Q     Max
## -0.46024 -0.23862  0.01347  0.18601  0.53443
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 43.6390418 14.6939240  2.970  0.00707 ** 
## Year        -0.0238480  0.0071667 -3.328  0.00305 ** 
## WinterRain   0.0011667  0.0004820  2.420  0.02421 *  
## AGST         0.6163916  0.0951747  6.476 1.63e-06 *** 
## HarvestRain  -0.0038606  0.0008075 -4.781 8.97e-05 *** 
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2865 on 22 degrees of freedom
## Multiple R-squared:  0.8275, Adjusted R-squared:  0.7962 
## F-statistic: 26.39 on 4 and 22 DF,  p-value: 4.057e-08

# With AIC
modAIC <- stepAIC(mod, k = 2)
## Start:  AIC=-61.07
## Price ~ Year + WinterRain + AGST + HarvestRain + Age + FrancePop
##
##
## Step:  AIC=-61.07
## Price ~ Year + WinterRain + AGST + HarvestRain + FrancePop
##
##             Df Sum of Sq    RSS     AIC
## - FrancePop  1   0.0026  1.8058 -63.031
## - Year       1   0.0048  1.8080 -62.998
## <none>          1.8032 -61.070
## - WinterRain 1   0.4585  2.2617 -56.952
## - HarvestRain 1   1.8063  3.6095 -44.331
## - AGST        1   3.3756  5.1788 -34.584
##
## Step:  AIC=-63.03
## Price ~ Year + WinterRain + AGST + HarvestRain
##
##             Df Sum of Sq    RSS     AIC
## <none>          1.8058 -63.031
## - WinterRain  1   0.4809  2.2867 -58.656
## - Year        1   0.9089  2.7147 -54.023
## - HarvestRain 1   1.8760  3.6818 -45.796
## - AGST        1   3.4428  5.2486 -36.222
summary(modAIC)
##
## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain,
##      data = wine)

```

```

## 
## Residuals:
##      Min      1Q  Median      3Q     Max 
## -0.46024 -0.23862  0.01347  0.18601  0.53443 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 43.6390418 14.6939240  2.970  0.00707 ** 
## Year        -0.0238480  0.0071667 -3.328  0.00305 ** 
## WinterRain   0.0011667  0.0004820  2.420  0.02421 *  
## AGST         0.6163916  0.0951747  6.476 1.63e-06 *** 
## HarvestRain  -0.0038606  0.0008075 -4.781 8.97e-05 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.2865 on 22 degrees of freedom 
## Multiple R-squared:  0.8275, Adjusted R-squared:  0.7962 
## F-statistic: 26.39 on 4 and 22 DF,  p-value: 4.057e-08

```

Note that the selected models `modBIC` and `modAIC` are equivalent to the `modWine2` we selected in Section 2.7.3 as the best model. This is an illustration that the model selected by `stepAIC` is often a good starting point for further additions or deletions of predictors.



When applying `stepAIC` for BIC/AIC, different final models might be selected depending on the choice of `direction`. This is the interpretation:

- "backward": starts from the full model, *removes* predictors sequentially.
- "forward": starts from the simplest model, *adds* predictors sequentially.
- "both" (default): combination of the above.

The **advice** is to try several of these methods and retain the one with minimum BIC/AIC. Set `trace = 0` to omit lengthy outputs of information of the search procedure.

Further options with `stepAIC`:

```

# Different search directions and omitting the trace,
# gives only the final model
modAICFor <- stepAIC(mod, trace = 0, direction = "forward")
modAICBack <- stepAIC(mod, trace = 0, direction = "backward")
modAICFor
## 
## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain +
##     Age + FrancePop, data = wine)
## 
## Coefficients:
## (Intercept)          Year      WinterRain          AGST      HarvestRain
## 2.496e+01   -1.377e-02    1.153e-03    6.144e-01   -3.837e-03
##     Age      FrancePop
##     NA      -2.213e-05
modAICBack
## 
## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain,

```

```
##      data = wine)
##
## Coefficients:
## (Intercept)      Year  WinterRain       AGST  HarvestRain
## 43.639042    -0.023848     0.001167     0.616392    -0.003861
```



stepAIC assumes that no NA's (missing values) are present in the data. It is advised to remove the missing values in the data before. Their presence might lead to errors. To do so, employ `data = na.omit(dataset)` in the call to `lm` (if your dataset is `dataset`). Also, see Appendix F.

We conclude by highlighting a caveat on the use of the BIC and AIC: they are constructed assuming that the sample size  $n$  is much larger than the number of parameters in the model ( $p + 2$ ). Therefore, they will work reasonably well if  $n \gg p + 2$ , but if this is not true they may favor unrealistic complex models. An illustration of this phenomena is Figure 3.3, which is the BIC/AIC version of Figure 2.8 for the experiment done in Section 2.6. The BIC and AIC curves tend to have local minimums close to  $p = 2$  and then increase. But when  $p + 2$  gets close to  $n$ , they quickly drop down. Note also how the BIC penalizes the complexity more than the AIC, which is more flat.

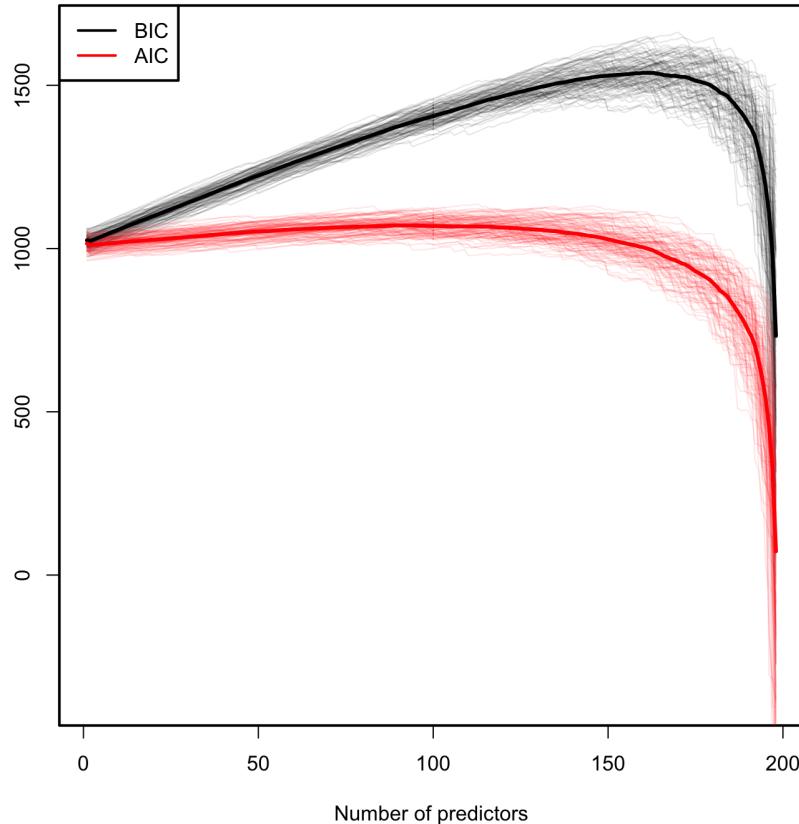


Figure 3.3: Comparison of BIC and AIC for  $n = 200$  and  $p$  ranging from 1 to 198.  $M = 100$  datasets were simulated with **only the first two** predictors being significant. The thicker curves are the mean of each color's curves.

### 3.2.1 Case study application

We want to build a linear model for predicting and explaining `medv`. There are a good number of predictors and some of them might be of little use for predicting `medv`. However, there is no clear intuition of which predictors will yield better explanations of `medv` with the information at hand. Therefore, we can start by doing a linear model on *all* the predictors:

```
modHouse <- lm(medv ~ ., data = Boston)
summary(modHouse)
##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595 -2.730 -0.518  1.777 26.199
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.646e+01 5.103e+00 7.144 3.28e-12 ***
## crim        -1.080e-01 3.286e-02 -3.287 0.001087 **
## zn          4.642e-02 1.373e-02 3.382 0.000778 ***
## indus       2.056e-02 6.150e-02 0.334 0.738288
## chas        2.687e+00 8.616e-01 3.118 0.001925 **
## nox         -1.777e+01 3.820e+00 -4.651 4.25e-06 ***
## rm          3.810e+00 4.179e-01 9.116 < 2e-16 ***
## age         6.922e-04 1.321e-02 0.052 0.958229
## dis         -1.476e+00 1.995e-01 -7.398 6.01e-13 ***
## rad         3.060e-01 6.635e-02 4.613 5.07e-06 ***
## tax         -1.233e-02 3.760e-03 -3.280 0.001112 **
## ptratio     -9.527e-01 1.308e-01 -7.283 1.31e-12 ***
## black       9.312e-03 2.686e-03 3.467 0.000573 ***
## lstat      -5.248e-01 5.072e-02 -10.347 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

There are a couple of non significant variables, but so far the model has an  $R^2 = 0.74$  and the fitted coefficients are sensible with what would be expected. For example, `crim`, `tax`, `ptratio`, and `nox` have negative effects on `medv`, while `rm`, `rad`, and `chas` have positive. However, the non significant coefficients are not significantly affecting the model, but only adding artificial noise and decreasing the overall accuracy of the coefficient estimates.

Let's polish the previous model a little bit. Instead of manually removing each non significant variable to reduce the complexity, we employ `stepAIC` for selecting a candidate *best model*:

```
# Best models
modBIC <- stepAIC(modHouse, k = log(nrow(Boston)))
## Start:  AIC=1648.81
## medv ~ crim + zn + indus + chas + nox + rm + age + dis + rad +
##       tax + ptratio + black + lstat
##
##          Df Sum of Sq   RSS   AIC
```

```

## - age      1     0.06 11079 1642.6
## - indus    1     2.52 11081 1642.7
## <none>          11079 1648.8
## - chas     1    218.97 11298 1652.5
## - tax      1    242.26 11321 1653.5
## - crim     1    243.22 11322 1653.6
## - zn       1    257.49 11336 1654.2
## - black    1    270.63 11349 1654.8
## - rad      1    479.15 11558 1664.0
## - nox     1    487.16 11566 1664.4
## - ptratio   1   1194.23 12273 1694.4
## - dis      1   1232.41 12311 1696.0
## - rm       1   1871.32 12950 1721.6
## - lstat    1   2410.84 13490 1742.2
##
## Step:  AIC=1642.59
## medv ~ crim + zn + indus + chas + nox + rm + dis + rad + tax +
##       ptratio + black + lstat
##
##           Df Sum of Sq   RSS   AIC
## - indus    1     2.52 11081 1636.5
## <none>          11079 1642.6
## - chas     1    219.91 11299 1646.3
## - tax      1    242.24 11321 1647.3
## - crim     1    243.20 11322 1647.3
## - zn       1    260.32 11339 1648.1
## - black    1    272.26 11351 1648.7
## - rad      1    481.09 11560 1657.9
## - nox     1    520.87 11600 1659.6
## - ptratio   1   1200.23 12279 1688.4
## - dis      1   1352.26 12431 1694.6
## - rm       1   1959.55 13038 1718.8
## - lstat    1   2718.88 13798 1747.4
##
## Step:  AIC=1636.48
## medv ~ crim + zn + chas + nox + rm + dis + rad + tax + ptratio +
##       black + lstat
##
##           Df Sum of Sq   RSS   AIC
## <none>          11081 1636.5
## - chas     1    227.21 11309 1640.5
## - crim     1    245.37 11327 1641.3
## - zn       1    257.82 11339 1641.9
## - black    1    270.82 11352 1642.5
## - tax      1    273.62 11355 1642.6
## - rad      1    500.92 11582 1652.6
## - nox     1    541.91 11623 1654.4
## - ptratio   1   1206.45 12288 1682.5
## - dis      1   1448.94 12530 1692.4
## - rm       1   1963.66 13045 1712.8
## - lstat    1   2723.48 13805 1741.5
modAIC <- stepAIC(modHouse, trace = 0, k = 2)

```

```

# Comparison
compareCoefs(modBIC, modAIC)
##
## Call:
## 1: lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad + tax +
##       ptratio + black + lstat, data = Boston)
## 2: lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad + tax +
##       ptratio + black + lstat, data = Boston)
##           Est. 1      SE 1      Est. 2      SE 2
## (Intercept) 36.34115  5.06749  36.34115  5.06749
## crim        -0.10841  0.03278  -0.10841  0.03278
## zn          0.04584  0.01352  0.04584  0.01352
## chas        2.71872  0.85424  2.71872  0.85424
## nox        -17.37602 3.53524 -17.37602 3.53524
## rm          3.80158  0.40632  3.80158  0.40632
## dis         -1.49271  0.18573  -1.49271  0.18573
## rad          0.29961  0.06340  0.29961  0.06340
## tax         -0.01178  0.00337  -0.01178  0.00337
## ptratio     -0.94652  0.12907  -0.94652  0.12907
## black        0.00929  0.00267  0.00929  0.00267
## lstat       -0.52255  0.04742  -0.52255  0.04742
summary(modBIC)
##
## Call:
## lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad +
##      tax + ptratio + black + lstat, data = Boston)
##
## Residuals:
##    Min      1Q  Median      3Q      Max
## -15.5984 -2.7386 -0.5046  1.7273 26.2373
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.341145  5.067492  7.171 2.73e-12 ***
## crim        -0.108413  0.032779 -3.307 0.001010 **  
## zn          0.045845  0.013523  3.390 0.000754 *** 
## chas        2.718716  0.854240  3.183 0.001551 **  
## nox        -17.376023 3.535243 -4.915 1.21e-06 ***
## rm          3.801579  0.406316  9.356 < 2e-16 ***
## dis         -1.492711  0.185731 -8.037 6.84e-15 ***
## rad          0.299608  0.063402  4.726 3.00e-06 ***
## tax         -0.011778  0.003372 -3.493 0.000521 *** 
## ptratio     -0.946525  0.129066 -7.334 9.24e-13 ***
## black        0.009291  0.002674  3.475 0.000557 *** 
## lstat       -0.522553  0.047424 -11.019 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.736 on 494 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7348
## F-statistic: 128.2 on 11 and 494 DF,  p-value: < 2.2e-16

# Confidence intervals

```

```
confint(modBIC)
##                  2.5 %      97.5 %
## (Intercept) 26.384649126 46.29764088
## crim        -0.172817670 -0.04400902
## zn          0.019275889  0.07241397
## chas        1.040324913  4.39710769
## nox        -24.321990312 -10.43005655
## rm          3.003258393  4.59989929
## dis        -1.857631161 -1.12779176
## rad          0.175037411  0.42417950
## tax        -0.018403857 -0.00515209
## ptratio     -1.200109823 -0.69293932
## black        0.004037216  0.01454447
## lstat       -0.615731781 -0.42937513
```

Note how the  $R^2_{\text{Adj}}$  has slightly increased with respect to the full model and how all the predictors are significant. Note also that `modBIC` and `modAIC` are the same.

We have quantified the influence of the predictor variables in the housing prices (Q1) and we can conclude that, in the final model (Q2) and with confidence level  $\alpha = 0.05$ :

- `chas`, `age`, `rad`, and `black` have a **significantly positive** influence on `medv`.
- `nox`, `dis`, `tax`, `pratio`, and `lstat` have a **significantly negative** influence on `medv`.



The MASS functions `addterm` and `dropterm` allow to add and remove all individual predictors to a given model, and inform the BICs / AICs of the possible combinations. Check that: - `modBIC` can not be improved in terms of BIC by removing predictors. Use `dropterm(modBIC, k = log(nobs(modBIC)))` for that. - `modBIC` can not be improved in terms of BIC by adding predictors. Use `addterm(modBIC, scope = lm(medv ~ ., data = Boston), k = log(nobs(modBIC)))` for that. `scope` must specify the maximal model or formula<sup>2</sup> to compare additions with.

### 3.3 Use of qualitative predictors

An important situation not covered so far is how to deal with *qualitative*, and not *quantitative*, predictors. 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.

The previous interpretation translates easily to the linear model. Assume that the dummy variable  $D$  is

available together with other predictors  $X_1, \dots, X_p$ . Then:

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p, D = d] = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \beta_{p+1} D.$$

The coefficient associated to  $D$  is easily interpretable.  $\beta_{p+1}$  is the increment in mean of  $Y$  associated to changing  $D = 0$  (reference) to  $D = 1$ , while the rest of the predictors are fixed. Or in other words,  $\beta_{p+1}$  is the increment in mean of  $Y$  associated to changing the level of the categorical variable from  $a$  to  $b$ .

R does the dummmification 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  $D_1 = 0, D_2 = 1$ . The interpretation of the regression models with the presence of  $D_1$  and  $D_2$  is very similar to the one before. For example, for the linear model, the coefficient associated to  $D_1$  gives the increment in mean of  $Y$  when the category of  $C$  changes from  $a$  to  $b$ . The coefficient for  $D_2$  gives the increment in mean of  $Y$  when it changes from  $a$  to  $c$ .

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 dummmification automatically for you if it detects that a factor variable is present in the regression model. Let's see an example with the `iris` dataset.

```
# iris dataset - factors in the last column
summary(iris)
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##   Median :5.800   Median :3.000   Median :4.350   Median :1.300
##   Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##   Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##           Species
##   setosa    :50
##   versicolor:50
##   virginica :50
##
##

# Summary of a linear model
mod1 <- lm(Sepal.Length ~ ., data = iris)
summary(mod1)
##
## Call:
## lm(formula = Sepal.Length ~ ., data = iris)
##
## Residuals:
##       Min     1Q   Median     3Q    Max
## -0.79424 -0.21874  0.00899  0.20255  0.73103
```

```

## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)           2.17127   0.27979   7.760 1.43e-12 ***
## Sepal.Width            0.49589   0.08607   5.761 4.87e-08 ***
## Petal.Length           0.82924   0.06853  12.101 < 2e-16 ***
## Petal.Width            -0.31516   0.15120  -2.084  0.03889 *  
## Speciesversicolor     -0.72356   0.24017  -3.013  0.00306 ** 
## Speciesvirginica      -1.02350   0.33373  -3.067  0.00258 ** 
## ---                  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.3068 on 144 degrees of freedom
## Multiple R-squared:  0.8673, Adjusted R-squared:  0.8627 
## F-statistic: 188.3 on 5 and 144 DF,  p-value: < 2.2e-16
# Speciesversicolor (D1) coefficient: -0.72356. The average increment of
# Sepal.Length when the species is versicolor instead of setosa (reference)
# Speciesvirginica (D2) coefficient: -1.02350. The average increment of
# Sepal.Length when the species is virginica instead of setosa (reference)
# Both dummy variables are significant

# How to set a different level as reference (versicolor)
iris$Species <- relevel(iris$Species, ref = "versicolor")

# Same estimates except for the dummy coefficients
mod2 <- lm(Sepal.Length ~ ., data = iris)
summary(mod2)
## 
## Call:
## lm(formula = Sepal.Length ~ ., data = iris)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.79424 -0.21874  0.00899  0.20255  0.73103
## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)           1.44770   0.28149   5.143 8.68e-07 ***
## Sepal.Width            0.49589   0.08607   5.761 4.87e-08 ***
## Petal.Length           0.82924   0.06853  12.101 < 2e-16 ***
## Petal.Width            -0.31516   0.15120  -2.084  0.03889 *  
## Speciessetosa          0.72356   0.24017   3.013  0.00306 ** 
## Speciesvirginica      -0.29994   0.11898  -2.521  0.01280 * 
## ---                  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.3068 on 144 degrees of freedom
## Multiple R-squared:  0.8673, Adjusted R-squared:  0.8627 
## F-statistic: 188.3 on 5 and 144 DF,  p-value: < 2.2e-16
# Speciessetosa (D1) coefficient: 0.72356. The average increment of
# Sepal.Length when the species is setosa instead of versicolor (reference)
# Speciesvirginica (D2) coefficient: -0.29994. The average increment of
# Sepal.Length when the species is virginica instead of versicolor (reference)

```

```

# Both dummy variables are significant

# Coefficients of the model
confint(mod2)
##                2.5 %    97.5 %
## (Intercept) 0.8913266 2.00408209
## Sepal.Width 0.3257653 0.66601260
## Petal.Length 0.6937939 0.96469395
## Petal.Width -0.6140049 -0.01630542
## Speciesversicolor 0.2488500 1.19827390
## Speciesvirginica -0.5351144 -0.06475727
# The coefficients of Speciesversicolor and Speciesvirginica are significantly negative

# Show the dummy variables employed for encoding a factor
contrasts(iris$Species)
##           setosa virginica
## versicolor      0      0
## setosa          1      0
## virginica       0      1
iris$Species <- relevel(iris$Species, ref = "setosa")
contrasts(iris$Species)
##           versicolor virginica
## setosa          0      0
## versicolor       1      0
## virginica        0      1

```



It may happen that one dummy variable, say  $D_1$ , is not significant, while other dummy variables, say  $D_2$ , are significant. For example, this happens in the example above at level  $\alpha = 0.01$ .



**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 consider 1, 1.5, and 1.75 instead?

The right way of dealing with categorical variables in regression is to set the variable as a factor and let R do the dummification internally.

### 3.3.1 Case study application

Let's see what the dummy variables are in the `Boston` dataset and what effect they have on `medv`.

```

# Load the Boston dataset
library(MASS)
data(Boston)

# Structure of the data

```

```

str(Boston)
## 'data.frame': 506 obs. of 14 variables:
## $ crim : num 0.00632 0.02731 0.02729 0.03237 0.06905 ...
## $ zn : num 18 0 0 0 0 12.5 12.5 12.5 12.5 ...
## $ indus : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 ...
## $ chas : int 0 0 0 0 0 0 0 0 0 ...
## $ nox : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 ...
## $ rm : num 6.58 6.42 7.18 7 7.15 ...
## $ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
## $ dis : num 4.09 4.97 4.97 6.06 6.06 ...
## $ rad : int 1 2 2 3 3 3 5 5 5 ...
## $ tax : num 296 242 242 222 222 311 311 311 311 ...
## $ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 ...
## $ black : num 397 397 393 395 397 ...
## $ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
## $ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
# chas is a dummy variable measuring if the suburb is close to the river (1)
# or not (0). In this case it is not codified as a factor but as a 0 or 1
# (so it is already dummyfied)

# Summary of a linear model
mod <- lm(medv ~ chas + crim, data = Boston)
summary(mod)
##
## Call:
## lm(formula = medv ~ chas + crim, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.540 -5.421 -1.878  2.575 30.134
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 23.61403   0.41862  56.409 < 2e-16 ***
## chas        5.57772   1.46926   3.796 0.000165 ***
## crim       -0.40598   0.04339  -9.358 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.373 on 503 degrees of freedom
## Multiple R-squared:  0.1744, Adjusted R-squared:  0.1712
## F-statistic: 53.14 on 2 and 503 DF,  p-value: < 2.2e-16
# The coefficient associated to chas is 5.57772. That means that if the suburb
# is close to the river, the mean of medv increases in 5.57772 units for
# the same house and neighborhood conditions
# chas is significant (the presence of the river adds a valuable information
# for explaining medv)

# Summary of the best model in terms of BIC
summary(modBIC)
##
## Call:
## lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad +

```

```

##      tax + ptratio + black + lstat, data = Boston)
##
## Residuals:
##      Min      1Q  Median      3Q     Max
## -15.5984 -2.7386 -0.5046  1.7273 26.2373
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.341145  5.067492  7.171 2.73e-12 ***
## crim        -0.108413  0.032779 -3.307 0.001010 **  
## zn          0.045845  0.013523  3.390 0.000754 ***  
## chas        2.718716  0.854240  3.183 0.001551 **  
## nox        -17.376023 3.535243 -4.915 1.21e-06 ***  
## rm          3.801579  0.406316  9.356 < 2e-16 ***  
## dis        -1.492711  0.185731 -8.037 6.84e-15 ***  
## rad         0.299608  0.063402  4.726 3.00e-06 ***  
## tax        -0.011778  0.003372 -3.493 0.000521 ***  
## ptratio     -0.946525  0.129066 -7.334 9.24e-13 ***  
## black       0.009291  0.002674  3.475 0.000557 ***  
## lstat      -0.522553  0.047424 -11.019 < 2e-16 ***  
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.736 on 494 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7348 
## F-statistic: 128.2 on 11 and 494 DF,  p-value: < 2.2e-16
# The coefficient associated to chas is 2.71871. If the suburb is close to
# the river, the mean of medv increases in 2.71871 units
# chas is significant as well in the presence of more predictors

```

We will see how to mix dummy and quantitative predictors in Section 3.4.3.

## 3.4 Nonlinear relationships

### 3.4.1 Transformations in the simple linear model

The linear model is termed *linear* not because the regression curve is a plane, but because *the effects of the parameters are linear*. Indeed, the predictor  $X$  may exhibit a nonlinear effect on the response  $Y$  and still be a linear model! For example, the following models can be transformed into simple linear models:

1.  $Y = \beta_0 + \beta_1 X^2 + \varepsilon$
2.  $Y = \beta_0 + \beta_1 \log(X) + \varepsilon$
3.  $Y = \beta_0 + \beta_1 (X^3 - \log(|X|) + 2^X) + \varepsilon$

The trick is to work with the transformed predictors ( $X^2, \log(X), \dots$ ), instead of with the original predictor  $X$ . Then, rather than working with the sample  $(X_1, Y_1), \dots, (X_n, Y_n)$ , we consider the transformed sample  $(\tilde{X}_1, Y_1), \dots, (\tilde{X}_n, Y_n)$  with (for the above examples):

1.  $\tilde{X}_i = X_i^2, i = 1, \dots, n.$
2.  $\tilde{X}_i = \log(X_i), i = 1, \dots, n.$
3.  $\tilde{X}_i = X_i^3 - \log(|X_i|) + 2^{X_i}, i = 1, \dots, n.$

An example of this simple but powerful trick is given as follows. The left panel of Figure 3.4 shows the scatterplot for some data  $y$  and  $x$ , together with its fitted regression line. Clearly, the data does not follow

a linear pattern, but a nonlinear one, similar to a parabola  $y = x^2$ . Hence,  $y$  might be better explained by the *square* of  $x$ ,  $x^2$ , rather than by  $x$ . Indeed, if we plot  $y$  against  $x^2$  in the right panel of Figure 3.4, we can see that the relation of  $y$  and  $x^2$  is now linear!

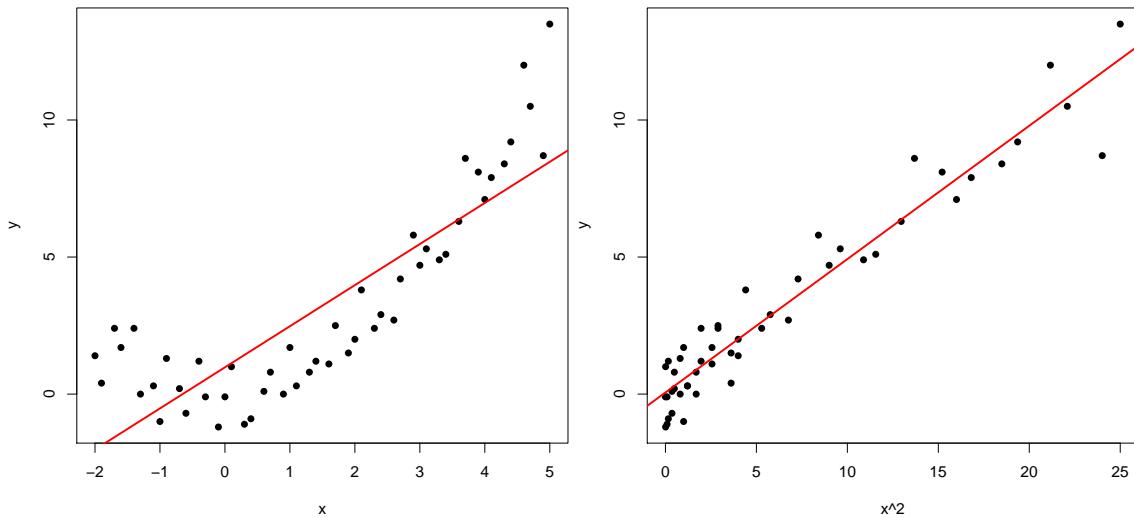


Figure 3.4: Left: quadratic pattern when plotting  $Y$  against  $X$ . Right: linearized pattern when plotting  $Y$  against  $X^2$ . In red, the fitted regression line.

In conclusion, with a simple trick we have drastically increased the explanation of the response. However, there is a catch: knowing which transformation is required in order to linearise the relation between response and the predictor is a kind of art which requires a good eye. One first approach is to consider one of the usual transformations, which are displayed in Figure 3.5, depending on the pattern of the data. Figure 3.4.1 illustrates how to choose an adequate transformation for linearizing nonlinear data patterns.

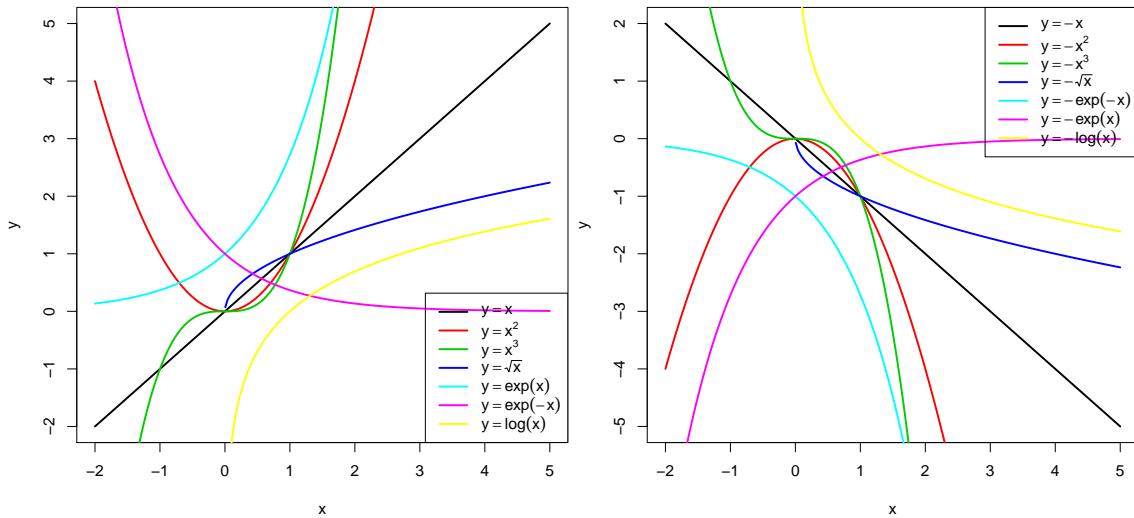


Figure 3.5: Some common nonlinear transformations and their negative counterparts. Recall the domain of definition of each transformation.

Illustration of the choice of the nonlinear transformation. Application also available here.



If you apply a nonlinear transformation, namely  $f$ , and fit the linear model  $Y = \beta_0 + \beta_1 f(X) + \varepsilon$ ,

then there is no point in also fitting the model resulting from the negative transformation  $-f$ . The model with  $-f$  is exactly the same as the one with  $f$  but with the sign of  $\beta_1$  flipped!

As a rule of thumb, use Figure 3.5 with the transformations to compare it with the data pattern, then choose the most similar curve, and finally apply the corresponding function with **positive sign**.

Let's see how we can compute transformations of our predictors and perform a linear regression with them. The data for Figure 3.4 is:

```

# Data
x <- c(-2, -1.9, -1.7, -1.6, -1.4, -1.3, -1.1, -1, -0.9, -0.7, -0.6,
      -0.4, -0.3, -0.1, 0, 0.1, 0.3, 0.4, 0.6, 0.7, 0.9, 1, 1.1, 1.3,
      1.4, 1.6, 1.7, 1.9, 2, 2.1, 2.3, 2.4, 2.6, 2.7, 2.9, 3, 3.1,
      3.3, 3.4, 3.6, 3.7, 3.9, 4, 4.1, 4.3, 4.4, 4.6, 4.7, 4.9, 5)
y <- c(1.4, 0.4, 2.4, 1.7, 2.4, 0, 0.3, -1, 1.3, 0.2, -0.7, 1.2, -0.1,
      -1.2, -0.1, 1, -1.1, -0.9, 0.1, 0.8, 0, 1.7, 0.3, 0.8, 1.2, 1.1,
      2.5, 1.5, 2, 3.8, 2.4, 2.9, 2.7, 4.2, 5.8, 4.7, 5.3, 4.9, 5.1,
      6.3, 8.6, 8.1, 7.1, 7.9, 8.4, 9.2, 12, 10.5, 8.7, 13.5)

# Data frame (a matrix with column names)
nonLinear <- data.frame(x = x, y = y)

# We create a new column inside nonLinear, called x2, that contains the
# new variable x^2
nonLinear$x2 <- nonLinear$x^2
# If you wish to remove it
# nonLinear$x2 <- NULL

# Regressions
mod1 <- lm(y ~ x, data = nonLinear)
mod2 <- lm(y ~ x2, data = nonLinear)
summary(mod1)
##
## Call:
## lm(formula = y ~ x, data = nonLinear)
## 
## Residuals:
##      Min      1Q  Median      3Q     Max 
## -2.5268 -1.7513 -0.4017  0.9750  5.0265 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.9771     0.3506   2.787   0.0076 ***
## x            1.4993     0.1374  10.911 1.35e-14 ***
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.005 on 48 degrees of freedom
## Multiple R-squared:  0.7126, Adjusted R-squared:  0.7067 
## F-statistic: 119 on 1 and 48 DF,  p-value: 1.353e-14
summary(mod2)
##
## Call:
```

```

## lm(formula = y ~ x2, data = nonLinear)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.0418 -0.5523 -0.1465  0.6286  1.8797
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.05891   0.18462   0.319   0.751
## x2          0.48659   0.01891  25.725  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9728 on 48 degrees of freedom
## Multiple R-squared:  0.9324, Adjusted R-squared:  0.931
## F-statistic: 661.8 on 1 and 48 DF,  p-value: < 2.2e-16
# mod2 has a larger R^2. Also notice the intercept is not significative

```



A fast way of performing and summarizing the quadratic fit is

```
summary(lm(y ~ I(x^2), data = nonLinear))
```

The `I()` function wrapping `x^2` is fundamental when applying arithmetic operations in the predictor. The symbols `+`, `\*`, `^`, ... have **different meaning** when inputted in a formula, so is required to use `I()` to indicate that they must be interpreted in their arithmetic meaning and that the result of the expression denotes a new predictor variable. For example, use `I((x - 1)^3 - log(3 \* x))` if you want to apply the transformation  $(x - 1)^3 - \log(3 * x)$ .



Load the dataset `assumptions.RData`. We are going to work with the regressions  $y2 \sim x2$ ,  $y3 \sim x3$ ,  $y8 \sim x8$ , and  $y9 \sim x9$ , in order to identify which transformation of Figure 3.5 gives the best fit. (For the purpose of illustration, we do not care if the assumptions are respected.) For these, do the following:

- Find the transformation that yields the largest  $R^2$ .
- Compare the original and the transformed linear models.

Some hints:

- $y2 \sim x2$  has a negative dependence, so look at the right panel of Figure 3.4.1.
- $y3 \sim x3$  seems to have just a subtle nonlinearity... Will it be worth it to attempt a transformation?
- For  $y9 \sim x9$ , try with also with `exp(-abs(x9))`, `log(abs(x9))`, and `2^abs(x9)`.

### 3.4.2 Polynomial transformations

A powerful nonlinear extension of the linear model are **polynomial models**. These are constructed from replacing each predictor  $X_j$  by the set of monomials  $(X_j, X_j^2, \dots, X_j^k)$  that are formed from  $X_j$ . In the case with a single predictor  $X$ , we have the  $k$ -th order polynomial fit:

$$Y = \beta_0 + \beta_1 X + \dots + \beta_k X^k + \varepsilon.$$

With this approach, a highly flexible model is produced, as it was shown in Figure 1.3. The creation of such models can be automated by the use of the function `poly`, which for the observations  $(X_1, \dots, X_n)$  of  $X$

creates the matrices

$$\begin{pmatrix} X_1 & X_1^2 & \dots & X_1^k \\ \vdots & \vdots & \ddots & \vdots \\ X_n & X_n^2 & \dots & X_n^k \end{pmatrix} \text{ or } \begin{pmatrix} p_1(X_1) & p_2(X_1) & \dots & p_k(X_1) \\ \vdots & \vdots & \ddots & \vdots \\ p_1(X_n) & p_2(X_n) & \dots & p_k(X_n) \end{pmatrix},$$

where  $p_1, \dots, p_k$  are *orthogonal*<sup>3</sup> polynomials of orders  $1, \dots, k$ , respectively.

Let's see a couple of examples:

```
x1 <- 1:5
poly(x = x1, degree = 2, raw = TRUE) # (X, X^2)
##      1 2
## [1,] 1 1
## [2,] 2 4
## [3,] 3 9
## [4,] 4 16
## [5,] 5 25
## attr(),"degree")
## [1] 1 2
## attr(),"class")
## [1] "poly"    "matrix"
poly(x = x1, degree = 2) # By default, it employs orthogonal polynomials
##      1 2
## [1,] -0.6324555  0.5345225
## [2,] -0.3162278 -0.2672612
## [3,]  0.0000000 -0.5345225
## [4,]  0.3162278 -0.2672612
## [5,]  0.6324555  0.5345225
## attr(),"coefs")
## attr(),"coefs")$alpha
## [1] 3 3
##
## attr(),"coefs")$norm2
## [1] 1 5 10 14
##
## attr(),"degree")
## [1] 1 2
## attr(),"class")
## [1] "poly"    "matrix"
```

These matrices can now be used as inputs in the predictor side of `lm`. Let's see this in an example.

```
# Data containing speed (mph) and stopping distance (ft) of cars from 1920
data(cars)
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)")

# Fit a linear model of dist ~ speed
mod1 <- lm(dist ~ speed, data = cars)
abline(coef = mod1$coefficients, col = 2)

# Quadratic
mod2 <- lm(dist ~ poly(speed, degree = 2), data = cars)
# The fit is not a line, we must look for an alternative approach
```

<sup>3</sup>In theory this means that  $\int p_j(x)p_l(x)dx = 0$  for  $j \neq l$ , which in practice translates to  $\sum_{i=1}^n p_j(X_i)p_l(X_i) = 0$ , thus uncorrelation.

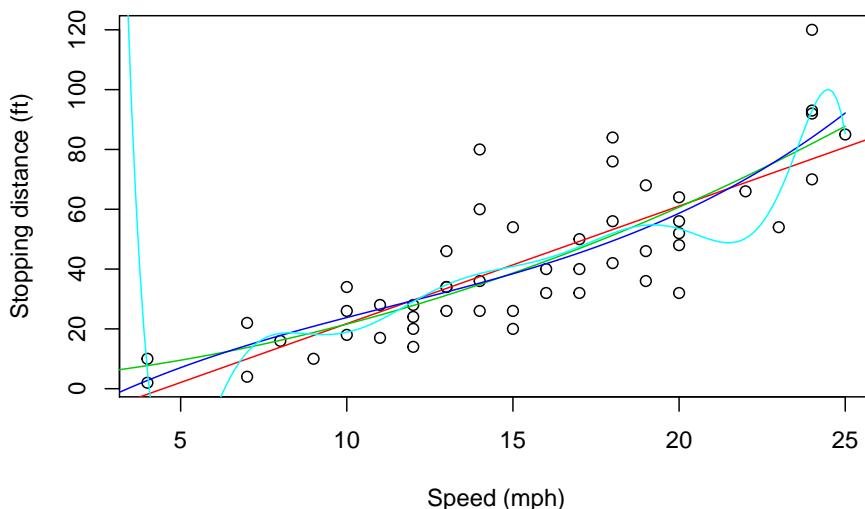
```

d <- seq(0, 25, length.out = 200)
lines(d, predict(mod2, new = data.frame(speed = d)), col = 3)

# Cubic
mod3 <- lm(dist ~ poly(speed, degree = 3), data = cars)
lines(d, predict(mod3, new = data.frame(speed = d)), col = 4)

# 10th order - overfitting
mod10 <- lm(dist ~ poly(speed, degree = 10), data = cars)
lines(d, predict(mod10, new = data.frame(speed = d)), col = 5)

```

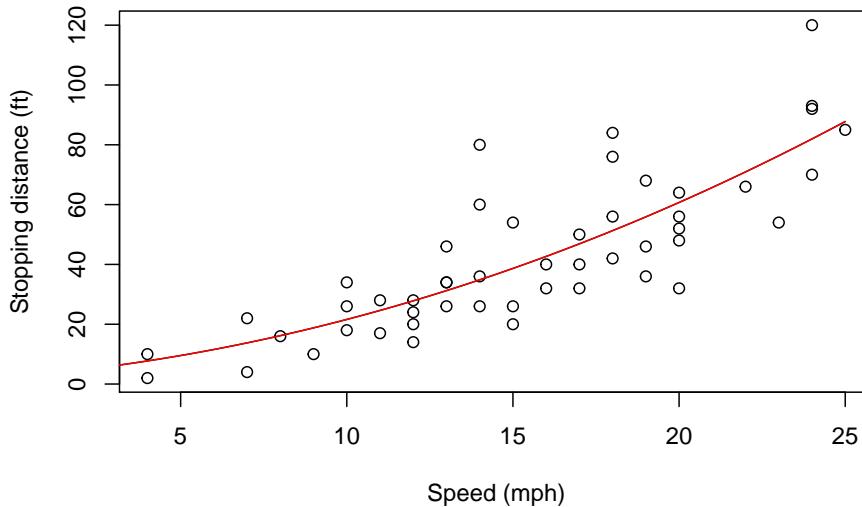


```

# BICs - the linear model is better!
BIC(mod1, mod2, mod3, mod10)
##      df      BIC
## mod1  3 424.8929
## mod2  4 426.4202
## mod3  5 429.4451
## mod10 12 450.3523

# poly computes by default orthogonal polynomials. These are not
# X^1, X^2, ..., X^p but combinations of them such that the polynomials are
# orthogonal. 'Raw' polynomials are possible with raw = TRUE. They give the
# same fit, but the coefficient estimates are different.
mod2Raw <- lm(dist ~ poly(speed, degree = 2, raw = TRUE), data = cars)
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)")
lines(d, predict(mod2, new = data.frame(speed = d)), col = 1)
lines(d, predict(mod2Raw, new = data.frame(speed = d)), col = 2)

```



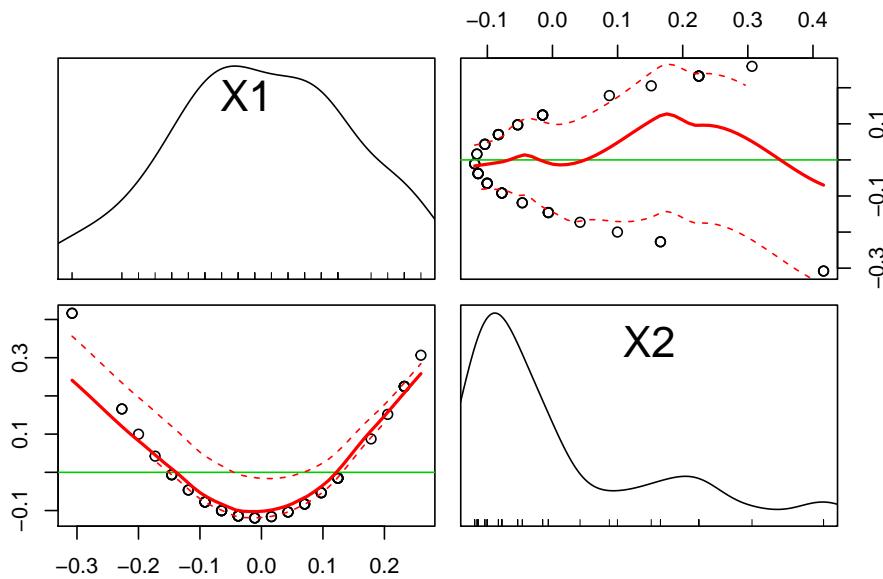
```

# However: different coefficient estimates, but same R^2. How is this possible?
summary(mod2)
##
## Call:
## lm(formula = dist ~ poly(speed, degree = 2), data = cars)
##
## Residuals:
##    Min     1Q   Median     3Q    Max
## -28.720 -9.184 -3.188  4.628 45.152
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                 42.980    2.146  20.026 < 2e-16 ***
## poly(speed, degree = 2)1 145.552    15.176   9.591 1.21e-12 ***
## poly(speed, degree = 2)2   22.996    15.176   1.515   0.136    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.18 on 47 degrees of freedom
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532 
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12
summary(mod2Raw)
##
## Call:
## lm(formula = dist ~ poly(speed, degree = 2, raw = TRUE), data = cars)
##
## Residuals:
##    Min     1Q   Median     3Q    Max
## -28.720 -9.184 -3.188  4.628 45.152
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                 2.47014   14.81716   0.167   0.868  
## poly(speed, degree = 2, raw = TRUE)1 0.91329    2.03422   0.449   0.656  
## poly(speed, degree = 2, raw = TRUE)2  0.09996    0.06597   1.515   0.136  
## ---
## Residual standard error: 15.18 on 47 degrees of freedom

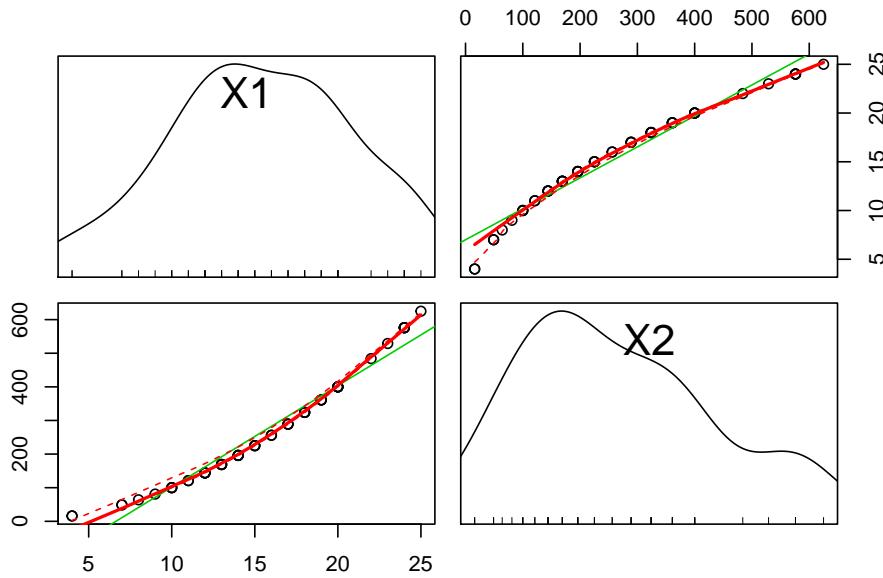
```

```
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12

# Because the predictors in mod2Raw are highly related between them, and the ones in
# mod2 are uncorrelated between them!
scatterplotMatrix(mod2$model[, -1])
```



```
scatterplotMatrix(mod2Raw$model[, -1])
```



```
cor(mod2$model[, -1])
##          1      2
## 1 1.000000e+00 4.686464e-17
## 2 4.686464e-17 1.000000e+00

cor(mod2Raw$model[, -1])
##          1      2
## 1 1.000000 0.9794765
## 2 0.9794765 1.0000000
```



The use of **orthogonal polynomials** is advised for high order polynomial fits, since it avoids numerical instabilities and excessive linear dependencies between the predictors. However, the coefficients of orthogonal polynomials are less interpretable than the ones from raw polynomials.

### 3.4.3 Interactions

When two or more predictors  $X_1$  and  $X_2$  are present, it may be of interest to explore the **interaction** between them  $X_1X_2$ . This is a new variable that positively (negatively) affects the response  $Y$  when both  $X_1$  and  $X_2$  are positive or negative at the same time (at different times):

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon$$

The coefficient  $\beta_3$  in  $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon$  can be interpreted as the **increment of the effect of the predictor  $X_1$  on the mean of  $Y$  for a unit increment in  $X_2$**  (the roles of  $X_1$  and  $X_2$  in can be interchanged). Significance testing on these coefficients can be carried out as usual.

The way of adding these interactions in `lm` is through `:` and `*`. The operator `:` only adds the term  $X_1 X_2$  and `*` adds  $X_1$ ,  $X_2$ , and  $X_1 X_2$ . Let's see an example in the `Boston` dataset.

```
# Interaction between lstat and age
summary(lm(medv ~ lstat + lstat:age, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat + lstat:age, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.815  -4.039  -1.335   2.086  27.491
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.041514   0.691334  52.133 < 2e-16 ***
## lstat       -1.388161   0.126911 -10.938 < 2e-16 ***
## lstat:age    0.004103   0.001133   3.621 0.000324 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.142 on 503 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.554 
## F-statistic: 314.6 on 2 and 503 DF,  p-value: < 2.2e-16
# For a unit increment in age, the effect of lstat in the response
# increases positively by 0.004103 units, shifting from -1.388161 to -1.384058
# Thus, the fact that age increases makes lstat is affecting less
# negatively medv. Note that the same interpretation does NOT hold if we
# switching the roles of age and lstat because age is not present as a sole
# predictor!

# First order interaction
summary(lm(medv ~ lstat * age, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
```

```

## 
## Residuals:
##   Min     1Q  Median     3Q     Max
## -15.806 -4.045 -1.333  2.085 27.552
##
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.0885359  1.4698355 24.553 < 2e-16 ***
## lstat        -1.3921168  0.1674555 -8.313 8.78e-16 ***
## age          -0.0007209  0.0198792 -0.036  0.9711    
## lstat:age    0.0041560  0.0018518  2.244  0.0252 *  
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.149 on 502 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.5531 
## F-statistic: 209.3 on 3 and 502 DF,  p-value: < 2.2e-16

# Second order interaction
summary(lm(medv ~ lstat * age * indus, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat * age * indus, data = Boston)
##
## Residuals:
##   Min     1Q  Median     3Q     Max
## -15.1549 -3.6437 -0.8427  2.1991 24.8751
##
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 46.103752  2.891173 15.946 < 2e-16 ***
## lstat        -2.641475  0.372223 -7.096 4.43e-12 ***
## age          -0.042300  0.041668 -1.015  0.31051    
## indus        -1.849829  0.380252 -4.865 1.54e-06 ***
## lstat:age    0.014249  0.004437  3.211  0.00141 ** 
## lstat:indus  0.177418  0.037647  4.713 3.18e-06 ***
## age:indus   0.014332  0.004386  3.268  0.00116 ** 
## lstat:age:indus -0.001621  0.000408 -3.973 8.14e-05 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.929 on 498 degrees of freedom
## Multiple R-squared:  0.5901, Adjusted R-squared:  0.5844 
## F-statistic: 102.4 on 7 and 498 DF,  p-value: < 2.2e-16

```

Stepwise regression can also be done with interaction terms. `stepAIC` supports interaction terms, but their inclusion must be asked for in the `scope` argument. By default, `scope` considers the largest model in which to perform stepwise regression as the formula of the model in `object`, the first argument. In order to set the largest model to search for the best subset of predictors as the one that contains first-order interactions, we proceed as follows:

```

# Include first-order interactions in the search for the best model in
# terms of BIC, not just single predictors
modIntBIC <- stepAIC(object = lm(medv ~ ., data = Boston), scope = medv ~ .^2,

```

```

k = log(nobs(modBIC)), trace = 0
summary(modIntBIC)
##
## Call:
## lm(formula = medv ~ crim + indus + chas + nox + rm + age + dis +
##      rad + tax + ptratio + black + lstat + rm:lstat + rad:lstat +
##      rm:rad + dis:rad + black:lstat + dis:ptratio + crim:chas +
##      chas:nox + chas:rm + chas:ptratio + rm:ptratio + age:black +
##      indus:dis + indus:lstat + crim:rm + crim:lstat, data = Boston)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -8.5845 -1.6797 -0.3157  1.5433 19.4311
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -9.673e+01  1.350e+01 -7.167 2.93e-12 ***
## crim        -1.454e+00  3.147e-01 -4.620 4.95e-06 ***
## indus        7.647e-01  1.237e-01  6.182 1.36e-09 ***
## chas         6.341e+01  1.115e+01  5.687 2.26e-08 ***
## nox         -1.691e+01  3.020e+00 -5.598 3.67e-08 ***
## rm          1.946e+01  1.730e+00 11.250 < 2e-16 ***
## age         2.233e-01  5.898e-02  3.786 0.000172 ***
## dis         -2.462e+00  6.776e-01 -3.634 0.000309 ***
## rad          3.461e+00  3.109e-01 11.132 < 2e-16 ***
## tax          -1.401e-02  2.536e-03 -5.522 5.52e-08 ***
## ptratio      1.207e+00  7.085e-01  1.704 0.089111 .
## black        7.946e-02  1.262e-02  6.298 6.87e-10 ***
## lstat        2.939e+00  2.707e-01 10.857 < 2e-16 ***
## rm:lstat    -3.793e-01  3.592e-02 -10.559 < 2e-16 ***
## rad:lstat   -4.804e-02  4.465e-03 -10.760 < 2e-16 ***
## rm:rad       -3.490e-01  4.370e-02 -7.986 1.05e-14 ***
## dis:rad     -9.236e-02  2.603e-02 -3.548 0.000427 ***
## black:lstat -8.337e-04  3.355e-04 -2.485 0.013292 *  
## dis:ptratio  1.371e-01  3.719e-02  3.686 0.000254 ***
## crim:chas   2.544e+00  3.813e-01  6.672 7.01e-11 ***
## chas:nox    -3.706e+01  6.202e+00 -5.976 4.48e-09 ***
## chas:rm     -3.774e+00  7.402e-01 -5.099 4.94e-07 ***
## chas:ptratio -1.185e+00  3.701e-01 -3.203 0.001451 ** 
## rm:ptratio   -3.792e-01  1.067e-01 -3.555 0.000415 ***
## age:black    -7.107e-04  1.552e-04 -4.578 5.99e-06 ***
## indus:dis    -1.316e-01  2.533e-02 -5.197 3.00e-07 ***
## indus:lstat  -2.580e-02  5.204e-03 -4.959 9.88e-07 ***
## crim:rm      1.605e-01  4.001e-02  4.011 7.00e-05 ***
## crim:lstat   1.511e-02  4.954e-03  3.051 0.002408 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.045 on 477 degrees of freedom
## Multiple R-squared:  0.8964, Adjusted R-squared:  0.8904
## F-statistic: 147.5 on 28 and 477 DF,  p-value: < 2.2e-16

# There is no improvement by removing terms in modIntBIC

```

```

dropterm(modIntBIC, k = log(nobs(modIntBIC)), sorted = TRUE)
## Single term deletions
##
## Model:
## medv ~ crim + indus + chas + nox + rm + age + dis + rad + tax +
##      ptratio + black + lstat + rm:lstat + rad:lstat + rm:rad +
##      dis:rad + black:lstat + dis:ptratio + crim:chas + chas:nox +
##      chas:rm + chas:ptratio + rm:ptratio + age:black + indus:dis +
##      indus:lstat + crim:rm + crim:lstat
##          Df Sum of Sq    RSS    AIC
## <none>          4423.7 1277.7
## black:lstat    1    57.28 4481.0 1278.0
## crim:lstat    1    86.33 4510.1 1281.2
## chas:ptratio   1    95.15 4518.9 1282.2
## dis:rad        1   116.73 4540.5 1284.6
## rm:ptratio     1   117.23 4541.0 1284.7
## dis:ptratio    1   126.00 4549.7 1285.7
## crim:rm        1   149.24 4573.0 1288.2
## age:black      1   194.40 4618.1 1293.2
## indus:lstat   1   228.05 4651.8 1296.9
## chas:rm        1   241.11 4664.8 1298.3
## indus:dis      1   250.51 4674.2 1299.3
## tax            1   282.77 4706.5 1302.8
## chas:nox       1   331.19 4754.9 1308.0
## crim:chas      1   412.86 4836.6 1316.6
## rm:rad          1   591.45 5015.2 1335.0
## rm:lstat       1  1033.93 5457.7 1377.7
## rad:lstat      1  1073.80 5497.5 1381.4

# Neither by including other terms interactions
addterm(modIntBIC, scope = lm(medv ~ .^2, data = Boston),
        k = log(nobs(modIntBIC)), sorted = TRUE)
## Single term additions
##
## Model:
## medv ~ crim + indus + chas + nox + rm + age + dis + rad + tax +
##      ptratio + black + lstat + rm:lstat + rad:lstat + rm:rad +
##      dis:rad + black:lstat + dis:ptratio + crim:chas + chas:nox +
##      chas:rm + chas:ptratio + rm:ptratio + age:black + indus:dis +
##      indus:lstat + crim:rm + crim:lstat
##          Df Sum of Sq    RSS    AIC
## <none>          4423.7 1277.7
## nox:age         1    52.205 4371.5 1277.9
## chas:lstat     1    50.231 4373.5 1278.1
## crim:nox       1    50.002 4373.7 1278.2
## indus:tax       1    46.182 4377.6 1278.6
## nox:rad         1    42.822 4380.9 1279.0
## tax:ptratio    1    37.105 4386.6 1279.6
## age:lstat      1    29.825 4393.9 1280.5
## rm:tax          1    27.221 4396.5 1280.8
## nox:rm          1    25.099 4398.6 1281.0
## nox:ptratio    1    17.994 4405.7 1281.8
## rm:age          1    16.956 4406.8 1282.0

```

```

## crim:black      1  15.566 4408.2 1282.1
## dis:tax        1  13.336 4410.4 1282.4
## dis:lstat      1  10.944 4412.8 1282.7
## rm:black       1   9.909 4413.8 1282.8
## rm:dis         1   9.312 4414.4 1282.8
## crim:indus    1   8.458 4415.3 1282.9
## tax:lstat      1   7.891 4415.8 1283.0
## ptratio:black  1   7.769 4416.0 1283.0
## rad:black      1   7.327 4416.4 1283.1
## age:ptratio   1   6.857 4416.9 1283.1
## age:tax        1   5.785 4417.9 1283.2
## nox:dis        1   5.727 4418.0 1283.2
## age:dis        1   5.618 4418.1 1283.3
## nox:tax        1   5.579 4418.2 1283.3
## crim:dis       1   5.376 4418.4 1283.3
## tax:black      1   4.867 4418.9 1283.3
## indus:age      1   4.554 4419.2 1283.4
## indus:rm        1   4.089 4419.6 1283.4
## indus:ptratio  1   4.082 4419.6 1283.4
## zn             1   3.919 4419.8 1283.5
## chas:tax       1   3.918 4419.8 1283.5
## rad:tax        1   3.155 4420.6 1283.5
## age:rad         1   3.085 4420.6 1283.5
## nox:black      1   2.939 4420.8 1283.6
## ptratio:lstat  1   2.469 4421.3 1283.6
## indus:chas     1   2.359 4421.4 1283.6
## chas:black     1   1.940 4421.8 1283.7
## indus:nox      1   1.440 4422.3 1283.7
## indus:black    1   1.177 4422.6 1283.8
## chas:rad       1   0.757 4423.0 1283.8
## chas:age       1   0.757 4423.0 1283.8
## crim:rad       1   0.678 4423.1 1283.8
## nox:lstat     1   0.607 4423.1 1283.8
## rad:ptratio   1   0.567 4423.2 1283.8
## crim:age       1   0.348 4423.4 1283.9
## indus:rad      1   0.219 4423.5 1283.9
## dis:black      1   0.077 4423.7 1283.9
## crim:ptratio  1   0.019 4423.7 1283.9
## crim:tax       1   0.004 4423.7 1283.9
## chas:dis       1   0.004 4423.7 1283.9

```



A fast way of accounting interactions between predictors is to use the `^` operator in `lm`:

- `lm(y ~ (x1 + x2 + x3)^2)` equals `lm(y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3)`. Higher powers like `lm(y ~ (x1 + x2 + x3)^3)` serve to include up to second-order interactions like `x1:x2:x3`.
- It is possible to regress on all the predictors and the first order interactions using `lm(y ~ .^2)`.
- Further flexibility in `lm` is possible, e.g. removing a particular interaction with `lm(y ~ .^2 - x1:x2)` or forcing the intercept to be zero with `lm(y ~ 0 + .^2)`.

Interactions are also possible with categorical variables. For example, for one predictor  $X$  and one dummy variable  $D$  encoding a factor with two levels, we have seven possible linear models stemming from how we

want to combine  $X$  and  $D$ :

1. *Predictor and no dummy variable.* Then we have the usual simple linear model:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

2. *Predictor and dummy variable.* Then  $D$  affects the intercept of the linear fit, which is different for each group:

$$Y = \beta_0 + \beta_1 X + \beta_2 D + \varepsilon = \begin{cases} \beta_0 + \beta_1 X + \varepsilon, & \text{if } D = 0, \\ (\beta_0 + \beta_2) + \beta_1 X + \varepsilon, & \text{if } D = 1. \end{cases}$$

3. *Predictor and dummy variable, with interaction.* Then  $D$  affects the intercept *and* the slope of the linear fit, and both are different for each group:

$$Y = \beta_0 + \beta_1 X + \beta_2 D + \beta_3 (X \cdot D) + \varepsilon = \begin{cases} \beta_0 + \beta_1 X + \varepsilon, & \text{if } D = 0, \\ (\beta_0 + \beta_2) + (\beta_1 + \beta_3) X + \varepsilon, & \text{if } D = 1. \end{cases}$$

4. *Predictor and interaction with dummy variable.* Then  $D$  affects only the slope of the linear fit, which is different for each group:

$$Y = \beta_0 + \beta_1 X + \beta_2 (X \cdot D) + \varepsilon = \begin{cases} \beta_0 + \beta_1 X + \varepsilon, & \text{if } D = 0, \\ \beta_0 + (\beta_1 + \beta_2) X + \varepsilon, & \text{if } D = 1. \end{cases}$$

5. *Dummy variable and no predictor.* Then  $D$  controls the intercept of a constant fit, depending on each group:

$$Y = \beta_0 + \beta_1 D + \varepsilon = \begin{cases} \beta_0 + \varepsilon, & \text{if } D = 0, \\ (\beta_0 + \beta_1) + \varepsilon, & \text{if } D = 1. \end{cases}$$

6. *Dummy variable and interaction with predictor.* Then  $D$  adds the predictor  $X$  for one group and affects the intercept, which is different for each group:

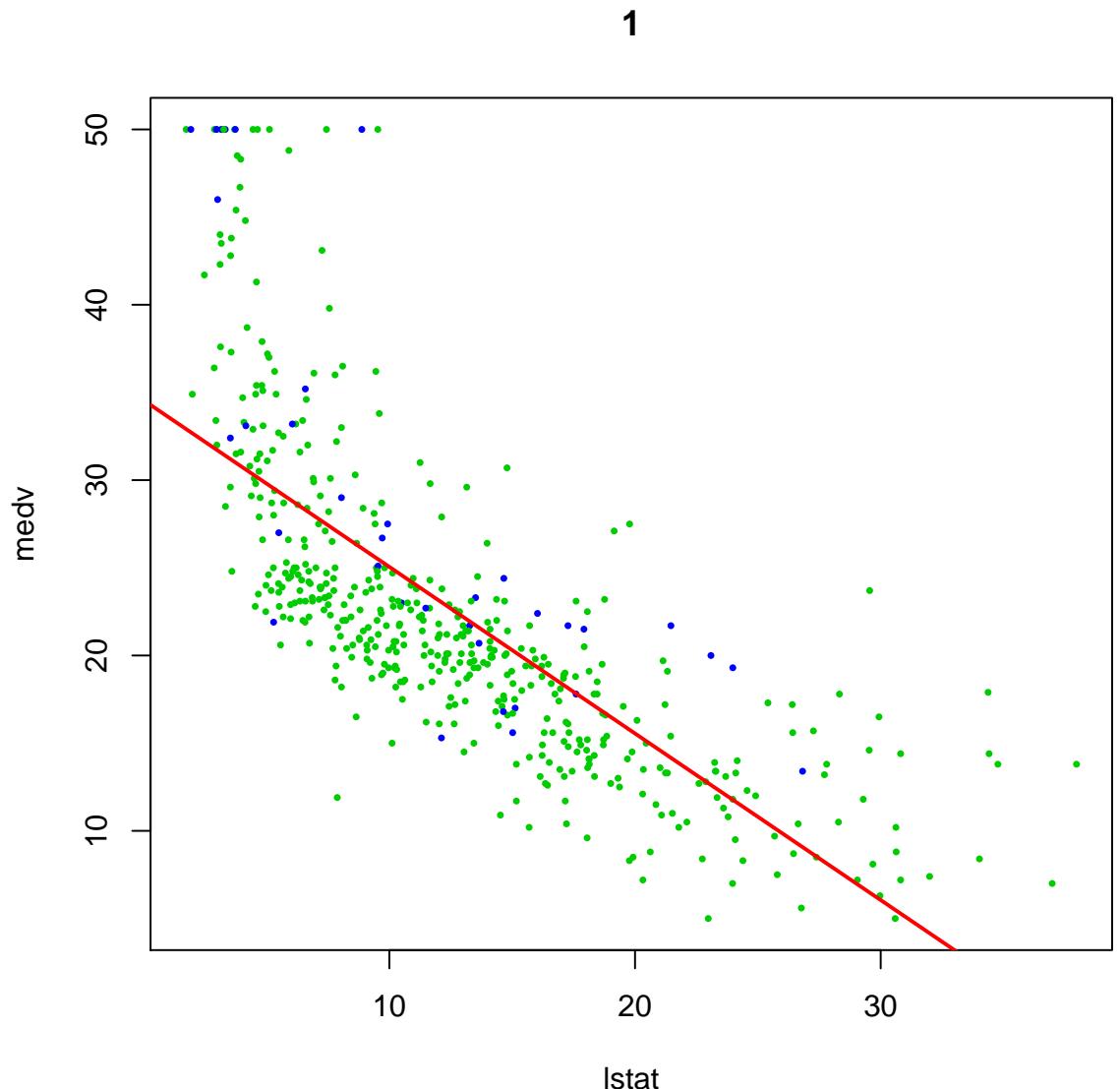
$$Y = \beta_0 + \beta_1 D + \beta_2 (X \cdot D) + \varepsilon = \begin{cases} \beta_0 + \varepsilon, & \text{if } D = 0, \\ (\beta_0 + \beta_1) + \beta_2 X + \varepsilon, & \text{if } D = 1. \end{cases}$$

7. *Interaction of dummy and predictor.* Then  $D$  the predictor  $X$  for one group and both intercepts are the same:

$$Y = \beta_0 + \beta_1 (X \cdot D) + \varepsilon = \begin{cases} \beta_0 + \varepsilon, & \text{if } D = 0, \\ \beta_0 + \beta_1 X + \varepsilon, & \text{if } D = 1. \end{cases}$$

Let's see the visualization of these seven possibilities.

```
# 1. No dummy variable
(mod1 <- lm(medv ~ lstat, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Coefficients:
## (Intercept)      lstat
##      34.55      -0.95
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
     main = "1")
abline(coef = mod1$coefficients, col = 2, lwd = 2)
```



```

# The rest of plots
par(mfrow = c(2, 3))

# 2. Dummy variable
(mod2 <- lm(medv ~ lstat + chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat + chas, data = Boston)
##
## Coefficients:
## (Intercept)      lstat          chas
##       34.0941     -0.9406      4.9200
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
     main = "2")
abline(a = mod2$coefficients[1], b = mod2$coefficients[2], col = 3, lwd = 2)
abline(a = mod2$coefficients[1] + mod2$coefficients[3],
       b = mod2$coefficients[2], col = 4, lwd = 2)

```

```

# 3. Dummy variable, with interaction
(mod3 <- lm(medv ~ lstat * chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat * chas, data = Boston)
##
## Coefficients:
## (Intercept)      lstat          chas      lstat:chas
##      33.7672     -0.9150      9.8251     -0.4329
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
     main = "3")
abline(a = mod3$coefficients[1], b = mod3$coefficients[2], col = 3, lwd = 2)
abline(a = mod3$coefficients[1] + mod3$coefficients[3],
       b = mod3$coefficients[2] + mod3$coefficients[4], col = 4, lwd = 2)

# 4. Dummy variable only present in interaction
(mod4 <- lm(medv ~ lstat + lstat:chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat + lstat:chas, data = Boston)
##
## Coefficients:
## (Intercept)      lstat      lstat:chas
##      34.4893     -0.9580      0.2128
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
     main = "4")
abline(a = mod4$coefficients[1], b = mod4$coefficients[2], col = 3, lwd = 2)
abline(a = mod4$coefficients[1],
       b = mod4$coefficients[2] + mod4$coefficients[3], col = 4, lwd = 2)

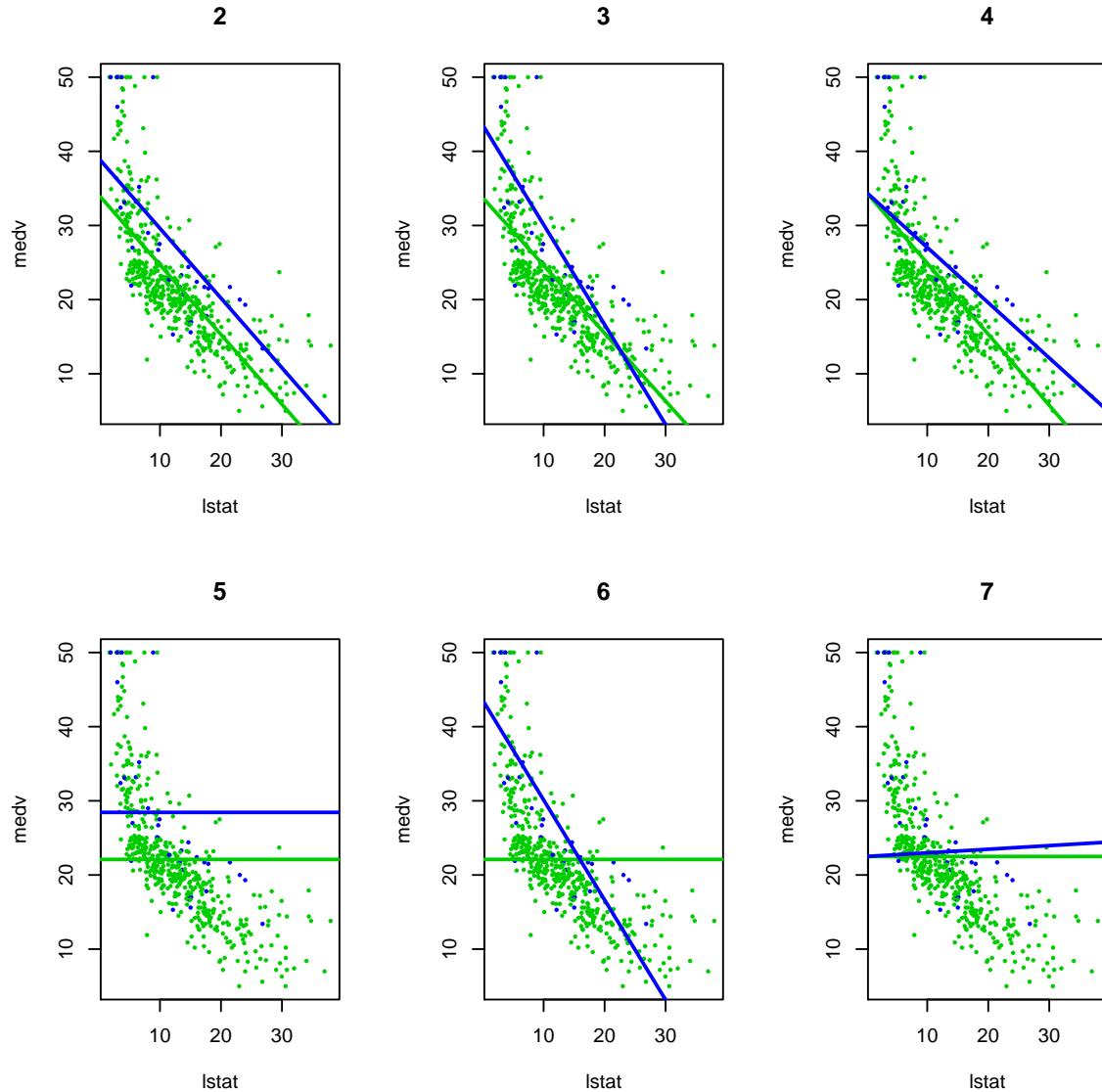
# 5. Dummy variable and no predictor
(mod5 <- lm(medv ~ chas, data = Boston))
##
## Call:
## lm(formula = medv ~ chas, data = Boston)
##
## Coefficients:
## (Intercept)      chas
##      22.094       6.346
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
     main = "5")
abline(a = mod5$coefficients[1], b = 0, col = 3, lwd = 2)
abline(a = mod5$coefficients[1] + mod5$coefficients[2], b = 0, col = 4, lwd = 2)

# 6. Dummy variable. Interaction in the intercept and slope
(mod6 <- lm(medv ~ chas + lstat:chas, data = Boston))
##
## Call:
## lm(formula = medv ~ chas + lstat:chas, data = Boston)
##
## Coefficients:
## (Intercept)      chas      chas:lstat
##      22.094      21.498     -1.348

```

```
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
      main = "6")
abline(a = mod6$coefficients[1], b = 0, col = 3, lwd = 2)
abline(a = mod6$coefficients[1] + mod6$coefficients[2],
       b = mod6$coefficients[3], col = 4, lwd = 2)

# 7. Dummy variable. Interaction in the intercept and slope
(mod7 <- lm(medv ~ lstat:chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat:chas, data = Boston)
##
## Coefficients:
## (Intercept)    lstat:chas
##      22.49484      0.04882
plot(medv ~ lstat, data = Boston, col = chas + 3, pch = 16, cex = 0.5,
      main = "7")
abline(a = mod7$coefficients[1], b = 0, col = 3, lwd = 2)
abline(a = mod7$coefficients[1], b = mod7$coefficients[2], col = 4, lwd = 2)
```



From the above illustration, it is clear that the effect of adding a dummy variable is to **simultaneously fit two linear models (with varying flexibility) to the two groups of data encoded by the dummy variable**, and merge this simultaneous fit within a single linear model. We can check this in more detail using the `subset` option of `lm`:

```
# Model using a dummy variable in the full dataset
lm(medv ~ lstat + chas + lstat:chas, data = Boston)
##
## Call:
## lm(formula = medv ~ lstat + chas + lstat:chas, data = Boston)
## 
## Coefficients:
## (Intercept)      lstat          chas      lstat:chas
##       33.7672     -0.9150      9.8251     -0.4329

# Individual model for the group with chas == 0
lm(medv ~ lstat, data = Boston, subset = chas == 0)
##
## Call:
```

```

## lm(formula = medv ~ lstat, data = Boston, subset = chas == 0)
##
## Coefficients:
## (Intercept)      lstat
##      33.767      -0.915
# Notice that the intercept and lstat coefficient are the same as before

# Individual model for the group with chas == 1
lm(medv ~ lstat, data = Boston, subset = chas == 1)
##
## Call:
## lm(formula = medv ~ lstat, data = Boston, subset = chas == 1)
##
## Coefficients:
## (Intercept)      lstat
##      43.592      -1.348
# Notice that the intercept and lstat coefficient equal the ones from the
# joint model, plus the specific terms associated to chas

```

This discussion can be extended to the situation where we have a factor with several levels, and hence more dummy variables. Such as for example in the `iris` dataset, where there are three groups. The next code shows how three group-specific linear regressions are modeled together by means of two dummy variables:

```

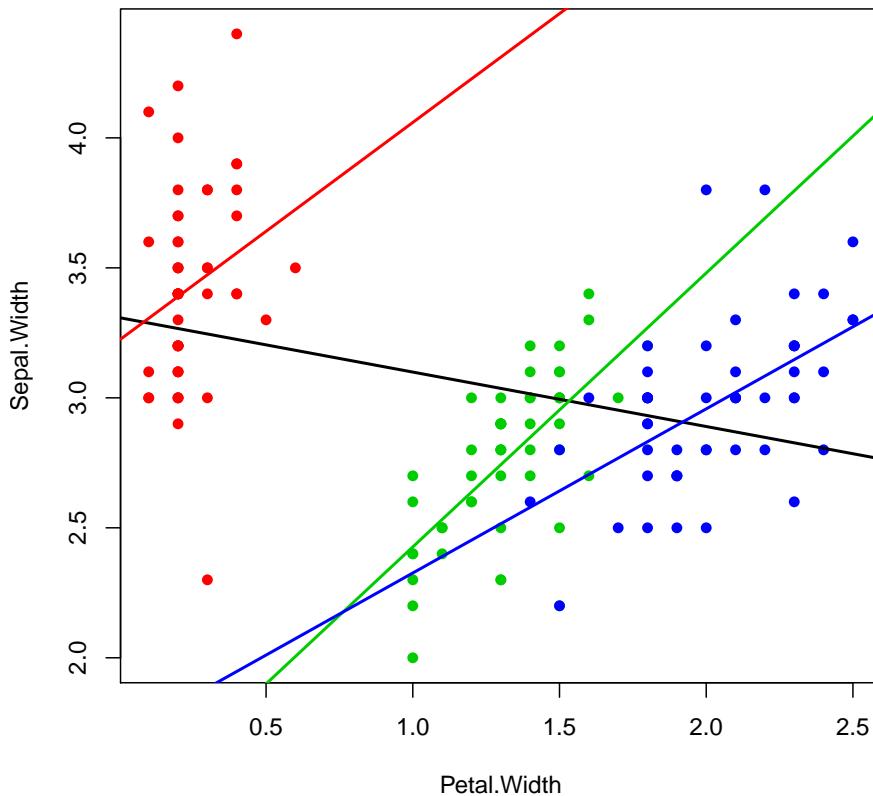
# Does not take into account the groups in the data
(modIris <- lm(Sepal.Width ~ Petal.Width, data = iris))
##
## Call:
## lm(formula = Sepal.Width ~ Petal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Petal.Width
##      3.3084      -0.2094

# Adding interactions with the groups
(modIrisSpecies <- lm(Sepal.Width ~ Petal.Width * Species, data = iris))
##
## Call:
## lm(formula = Sepal.Width ~ Petal.Width * Species, data = iris)
##
## Coefficients:
##                   (Intercept)          Petal.Width
##                         3.2221                  0.8372
## Speciesversicolor          Speciesvirginica
##                  -1.8492                  -1.5273
## Petal.Width:Speciesversicolor  Petal.Width:Speciesvirginica
##                         0.2165                  -0.2058

# Joint regression line shows negative correlation, but each group
# regression line shows a positive correlation
plot(Sepal.Width ~ Petal.Width, data = iris, col = as.integer(Species) + 1,
      pch = 16)
abline(a = modIris$coefficients[1], b = modIris$coefficients[2], lwd = 2)
abline(a = modIrisSpecies$coefficients[1], b = modIrisSpecies$coefficients[2],
       col = 2, lwd = 2)

```

```
abline(a = modIrisSpecies$coefficients[1] + modIrisSpecies$coefficients[3] ,
       b = modIrisSpecies$coefficients[2] + modIrisSpecies$coefficients[5] ,
       col = 3, lwd = 2)
abline(a = modIrisSpecies$coefficients[1] + modIrisSpecies$coefficients[4] ,
       b = modIrisSpecies$coefficients[2] + modIrisSpecies$coefficients[6] ,
       col = 4, lwd = 2)
```



The last scatterplot is an illustration of the **Simpson's paradox**. The simplest case of the paradox arises in simple linear regression, when there are two or more well-defined groups in the data such that:

1. Within each group, there is a clear and common correlation pattern between the response and the predictor.
2. When the groups are aggregated, the response and the predictor exhibit an *opposite* correlation pattern.

### 3.4.4 Case study application



The model employed in Harrison and Rubinfeld (1978) is different from the `modBIC` model. In the paper, several nonlinear transformations of the predictors and the response are done to improve the linear fit. Also, different units are used for `medv`, `black`, `lstat`, and `nox`. The authors considered these variables:

- Response: `log(1000 * medv)`

- *Linear predictors*: `age`, `black / 1000` (this variable corresponds to their  $(B - 0.63)^2$ ), `tax`, `ptratio`, `crim`, `zn`, `indus`, and `chas`.
- *Nonlinear predictors*: `rm^2`, `log(dis)`, `log(rad)`, `log(lstat / 100)`, and `(10 * nox)^2`.

Do the following:

1. Check if the model with such predictors corresponds to the one in the first column, Table VII, page 100 of Harrison and Rubinfeld (1978) (open-access paper available [here](#)). To do so, save this model as `modelHarrison` and summarize it.
2. Make a `stepAIC` selection of the variables in `modelHarrison` (use BIC) and save it as `modelHarrisonSel`. Summarize the fit.
3. Which model has a larger  $R^2$ ? And adjusted  $R^2$ ? Which is simpler and has more significant coefficients?

## 3.5 Model diagnostics

As we saw in Section 2.3, checking the assumptions of the multiple linear model through the visualization of the data becomes tricky even when  $p = 2$ . To solve this issue, a series of *diagnostic tools* have been designed in order to evaluate graphically and systematically the validity of the assumptions.

We will illustrate them in the `wine` dataset, which you can download as an `.RData` ([here](#)).

```
load("wine.RData")
mod <- lm(Price ~ Age + AGST + HarvestRain + WinterRain, data = wine)
summary(mod)
```



When one assumption fails, it is likely that this failure will affect other assumptions. For example, if linearity fails, then most likely homoscedasticity and normality will fail also. The key point is to identify the **root cause** of the assumptions failure in order to try to find a patch.

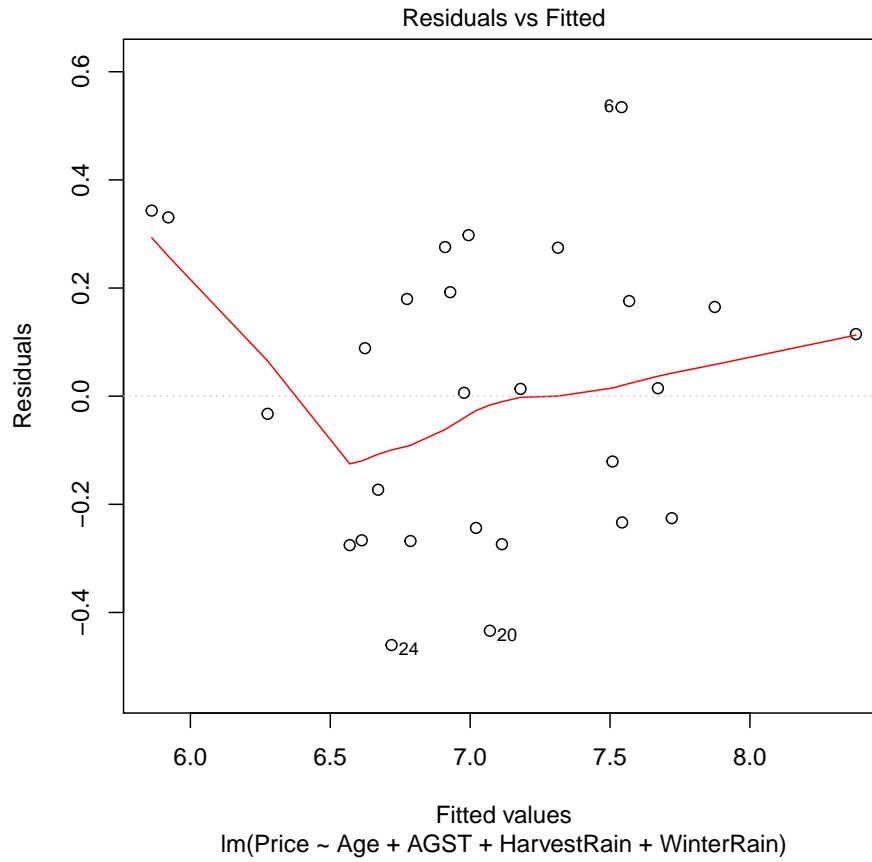
### 3.5.1 Linearity

Linearity between the response  $Y$  and the predictors  $X_1, \dots, X_p$  is the building block of the linear model. If this assumption fails, *i.e.*, if there is a nonlinear trend linking  $Y$  and at least one of the predictors  $X_1, \dots, X_p$  in a significant way, then all the conclusions we might extract from the analysis are suspected to be flawed. Therefore it is a **key assumption**.

#### How to check it

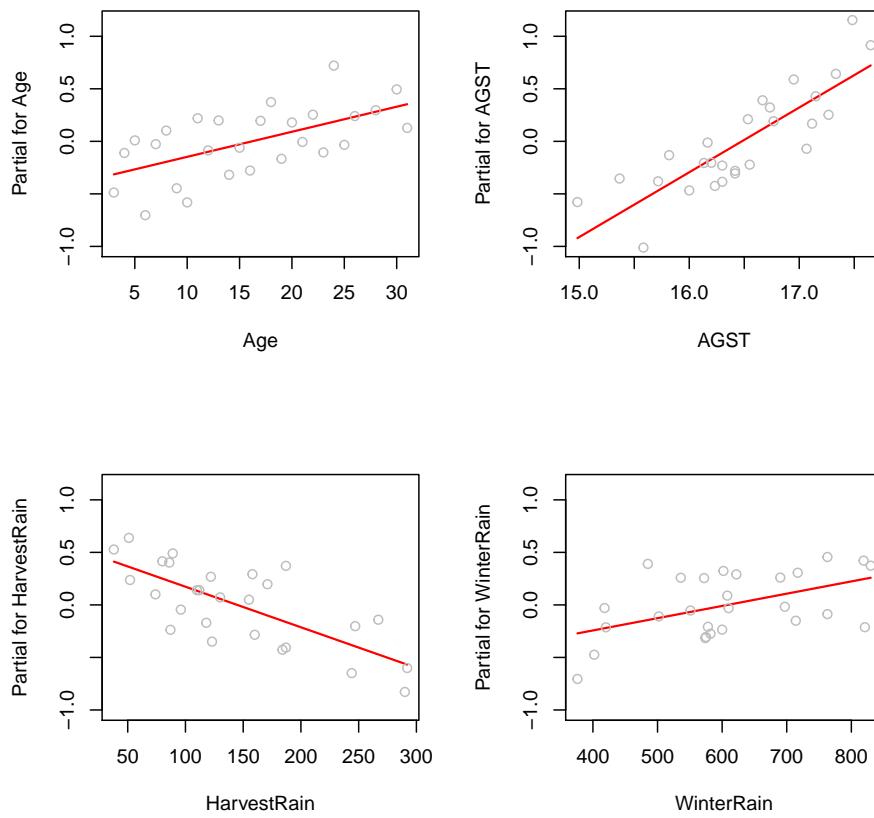
The so-called **residuals vs. fitted values plot** is the scatterplot of  $(\hat{Y}_i, \hat{\varepsilon}_i)$ ,  $i = 1, \dots, n$ , and is a very useful for detecting linearity departures using a single graphical device. Under linearity, we expect that there is *no trend in the residuals*  $\hat{\varepsilon}_i$  with respect to  $\hat{Y}_i$ . For example:

```
plot(mod, 1)
```



**Under linearity, we expect the red line** (a flexible fit of the mean of the residuals) **to capture no trend**. If nonlinearities are observed, it is worth to plot the *regression terms* of the model. These are the  $p$  scatterplots  $(X_{ij}, Y_i)$ ,  $i = 1, \dots, n$ , that are accompanied by the regression lines  $y = \hat{\beta}_0 + \hat{\beta}_j x$  (important:  $\hat{\beta}_j$ ,  $j = 1, \dots, p$ , come from the *multiple* linear fit that gives  $\hat{\beta}$ , not from individual *simple* linear regressions). They help with detecting which predictor is having nonlinear effects on  $Y$ .

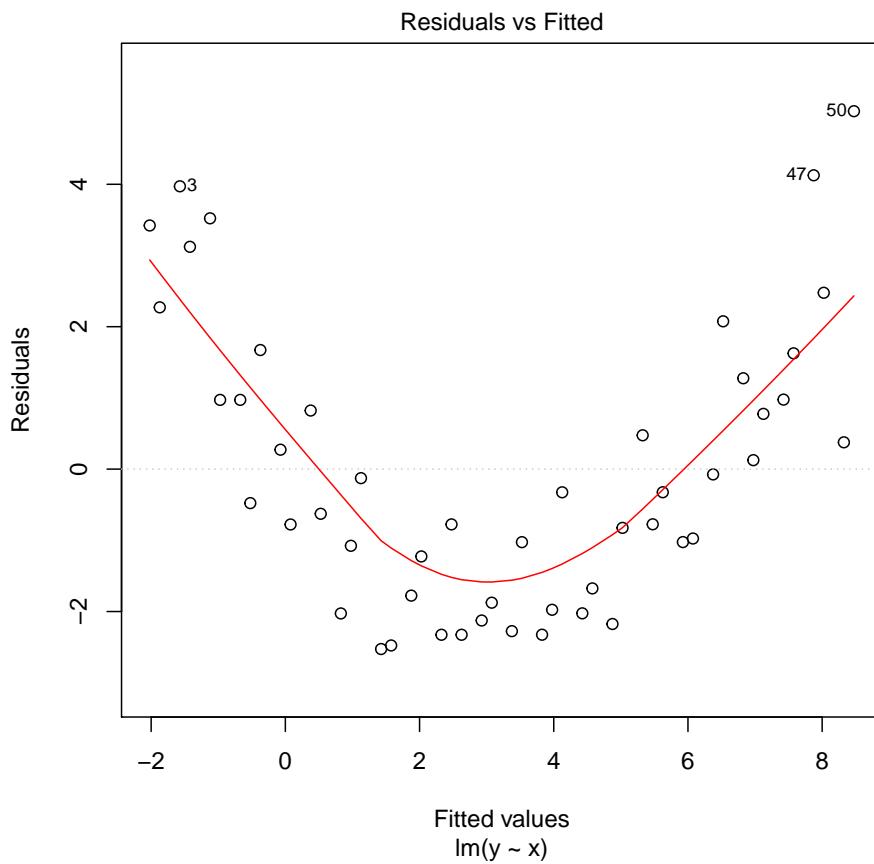
```
par(mfrow = c(2, 2)) # We have 4 predictors
termplot(mod, partial.resid = TRUE)
```

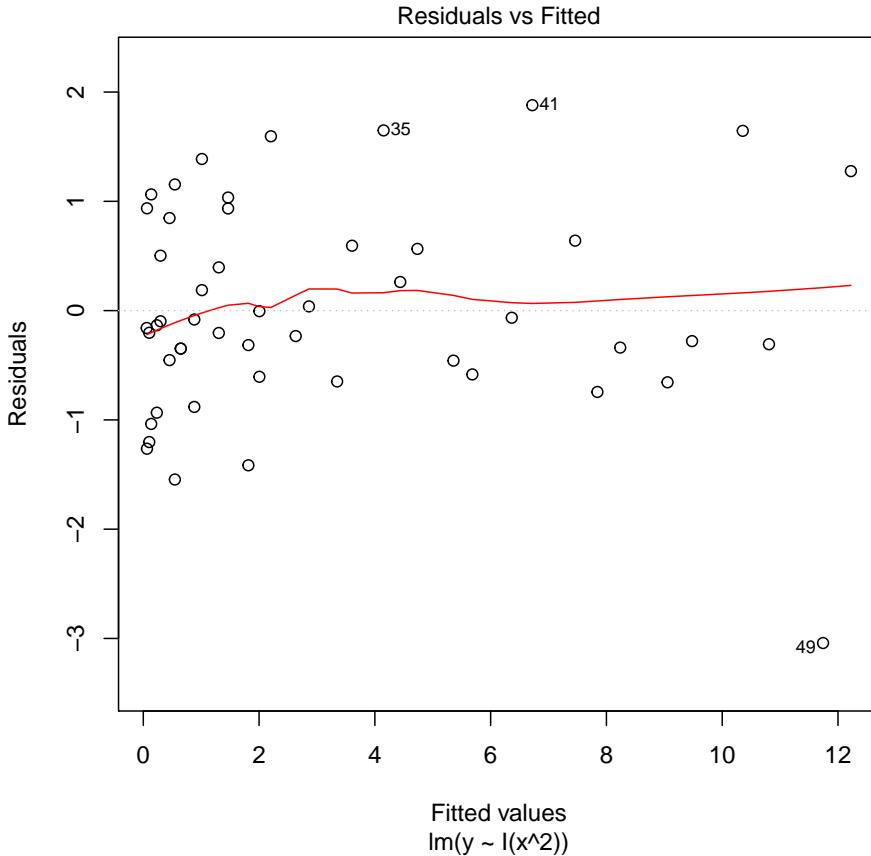


### What to do if fails

Using an adequate nonlinear transformation for the problematic predictors or adding interaction terms, as we saw in Section 3.4, might be helpful. Alternatively, consider a nonlinear transformation  $f$  for the response  $Y$  (of course, at the price of predicting  $f(Y)$  rather than  $Y$ ), as we will see in the case study of Section 3.5.7. Let's see the transformation of predictors in the example that motivated Section 3.4.

```
plot(lm(y ~ x, data = nonLinear), 1) # Nonlinear
```





### 3.5.2 Normality

The normality in the errors  $\varepsilon_i$ ,  $i = 1, \dots, n$ , allows us to make *exact* inference in the linear model, in the sense that the distribution of  $\hat{\beta}$  given in (2.11) is exact for  $n$ , is not asymptotic with  $n \rightarrow \infty$  or approximate. If normality does not hold, then the inference we did (confidence intervals for  $\beta_j$ , hypothesis testing, confidence intervals for prediction) is to be *somehow* suspected. Why just *somehow*? Roughly speaking, the reason is that the central limit theorem will make  $\hat{\beta}$  *asymptotically* normal, even if the errors are not. However, the speed of this asymptotic convergence greatly depends on how non normal is the distribution of the errors.

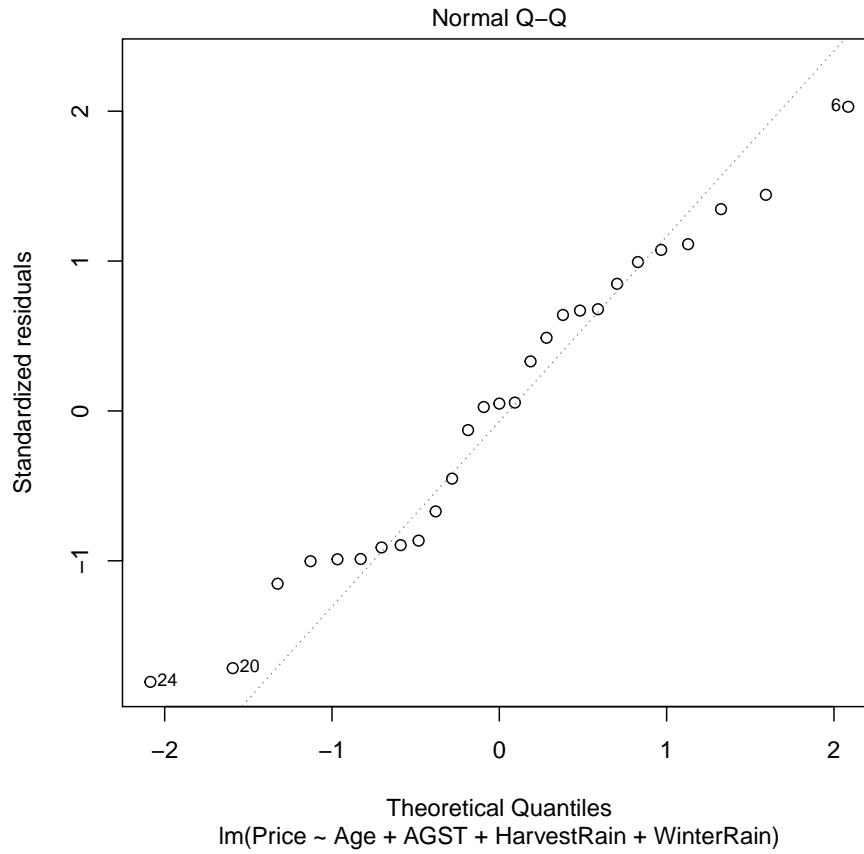
As a rule of thumb: **non severe<sup>4</sup> departures from normality yield valid (asymptotic) inference for relatively large sample sizes  $n$** . Therefore, the failure of normality is less problematic than other assumptions.

#### How to check it

The **QQ-plot** (Theoretical Quantile vs. Empirical Quantile) allows to check if the *standardized* residuals follow a  $\mathcal{N}(0, 1)$ . What it does is compare the theoretical quantiles of a  $\mathcal{N}(0, 1)$  with the quantiles of the sample of standardized residuals.

```
plot(mod, 2)
```

<sup>4</sup>Distributions that are not heavy tailed, not heavy multimodal, and not heavy skewed.



Under normality, we expect the points to align with the diagonal line, which represents the ideal position of the points if they were sampled from a  $\mathcal{N}(0, 1)$ . It is usual to have departures from the diagonal in the extremes than in the center, even under normality, although these departures are more evident if the data is non normal.

There are formal tests to check the null hypothesis of normality in our residuals:

```
# Shapiro-Wilks test of normality
shapiro.test(mod$residuals) # Allows up to 5000 observations
##
##  Shapiro-Wilk normality test
##
##  data:  mod$residuals
##  W = 0.95903, p-value = 0.3512
# We do not reject normality

# Lilliefors test - an adaptation of Kolmogorov-Smirnov test
library(nortest)
lillie.test(mod$residuals)
##
##  Lilliefors (Kolmogorov-Smirnov) normality test
##
##  data:  mod$residuals
##  D = 0.13739, p-value = 0.2125
# We do not reject normality
```

### What to do if fails

Patching non-normality is not easy and most of the time requires the consideration of other models, like the ones to be seen in Chapter 5. One possibility is to transform  $Y$  by means of a Box-Cox transformation of the form  $\frac{Y^\lambda - 1}{\lambda}$ ,  $\lambda \neq 0$  or  $\log(Y)$ , of course at the price of modelling the transformed response rather than  $Y$ . It is also possible to patch it if it is a consequence of the failure of linearity or homoscedasticity, which translates the problem into fixing those assumptions.

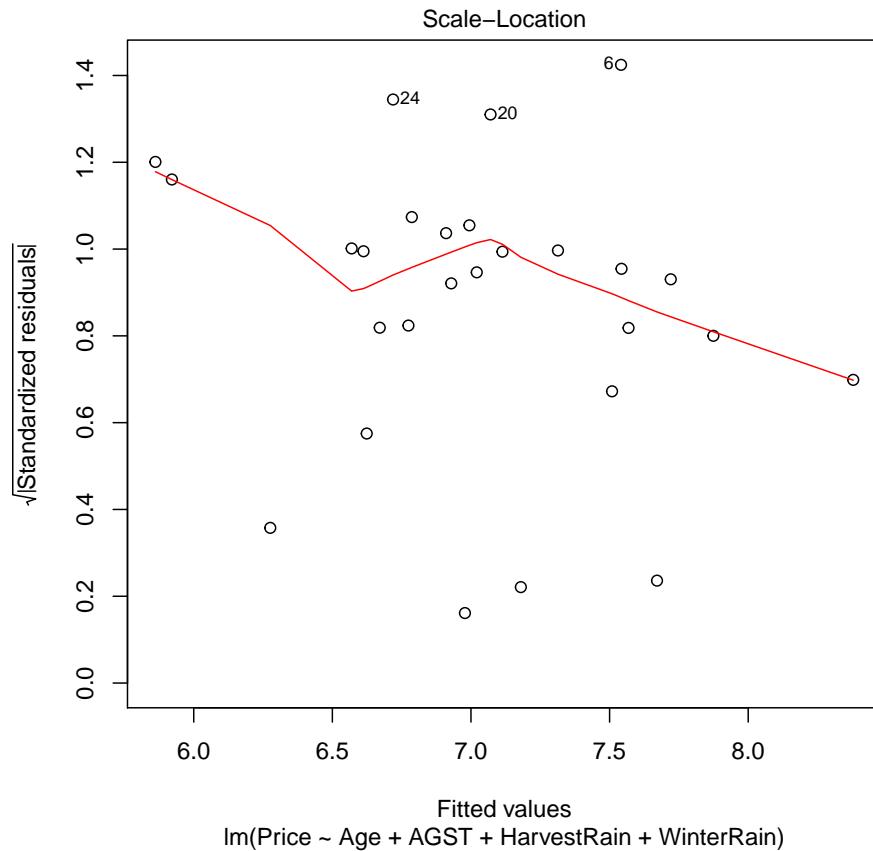
### 3.5.3 Homoscedasticity

The constant-variance assumption of the errors is also key for obtaining the inferential results we saw. For example, if the assumption does not hold, then the confidence intervals for prediction will be wider or shorter than the *adequate* (meaning that the confidence is not respected).

#### How to check it

Heteroskedasticity can be detected by looking into irregular *vertical* dispersion patterns in the residuals vs. fitted values plot. However, it is simpler to use the **scale-location plot**, where the standardized residuals are transformed by a square root (of its absolute value) to look only for deviations in the positive axis.

```
plot(mod, 3)
```



Under homoscedasticity, we expect the red line to show no trend. If there are consistent nonlinear patterns, then there is evidence of heteroskedasticity.

There are formal tests to check the null hypothesis of homoscedasticity in our residuals. For example:

```
# Breusch-Pagan test
ncvTest(mod)
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 0.3254683    Df = 1    p = 0.5683398
# We do not reject homoscedasticity
```



The Breusch-Pagan test checks homoscedasticity against a nonconstant, *linearly increasing with respect to the predictors*, variance in the residuals. This fact means that the test can be *fooled* by a nonlinear pattern in the variance of the residuals that results in a flat plane fit (e.g., a quadratic pattern). It is advised then to **check the scale-location plot in addition to performing the Breusch-Pagan test** in order to identify evident non constant variances driven by possibly tricky nonlinearities. The following code illustrates this with two examples:

```
# Heteroskedastic models
set.seed(123456)
x <- rnorm(100)
y1 <- 1 + 2 * x + rnorm(100, sd = x^2)
y2 <- 1 + 2 * x + rnorm(100, sd = 1 + x * (x > 0))
modHet1 <- lm(y1 ~ x)
modHet2 <- lm(y2 ~ x)

# Heteroskedasticity not detected
ncvTest(modHet1)
plot(modHet1, 3)

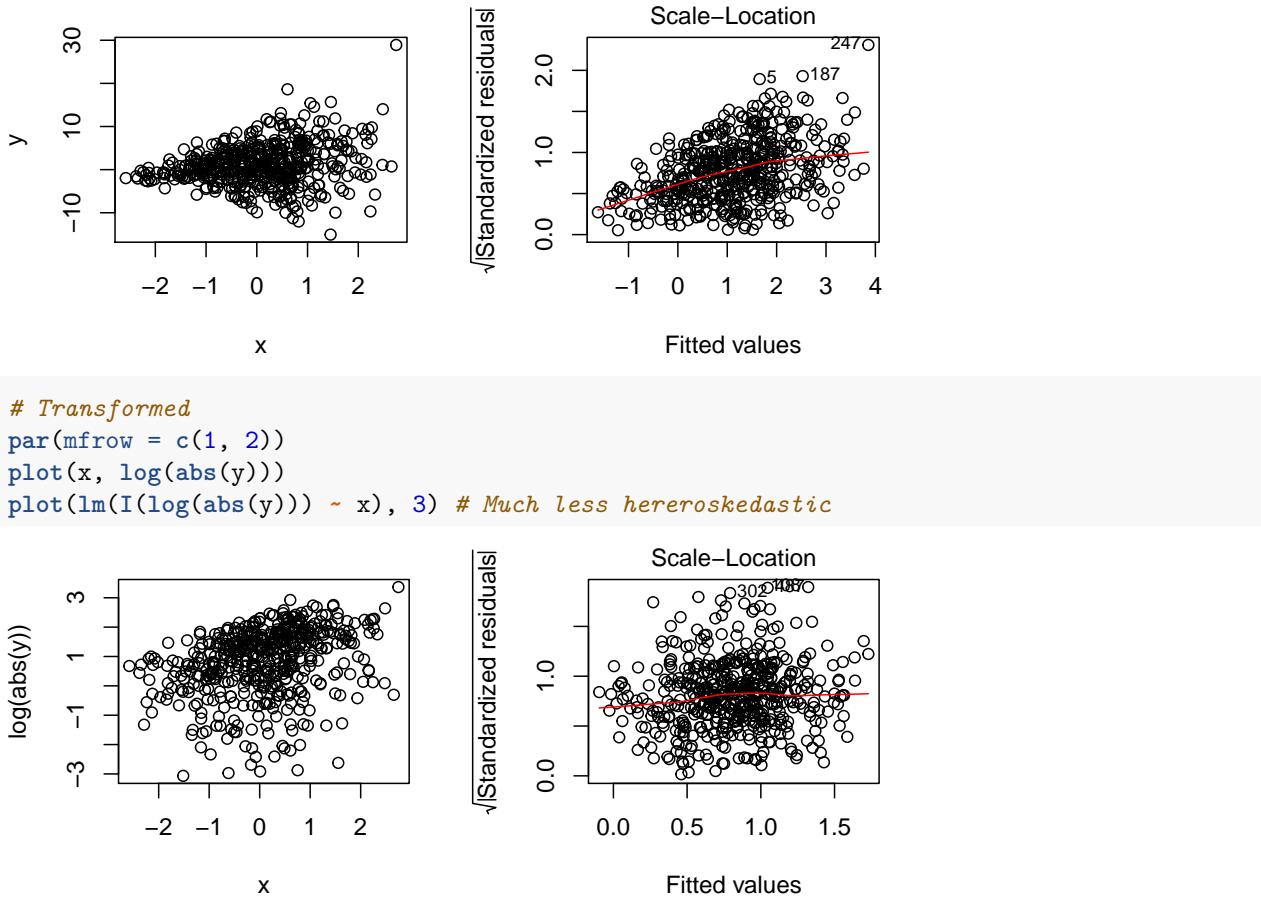
# Heteroskedasticity correctly detected
ncvTest(modHet2)
plot(modHet2, 3)
```

### What to do if fails

Using a nonlinear transformation *for the response Y* may help control the variance. Typical choices are  $\log Y$  and  $\sqrt{Y}$ , which reduce the scale of the larger responses and leads to a reduction of heteroskedasticity. Let's see a quick example of this.

```
# Artificial data with heteroskedasticity
set.seed(12345)
x <- rnorm(500)
e <- rnorm(500, sd = sqrt(0.1 + 2 * (x + 3)^2))
y <- 1 + x + e

# Original
par(mfrow = c(1, 2))
plot(x, y)
plot(lm(y ~ x), 3) # Very heteroskedastic
```



### 3.5.4 Independence

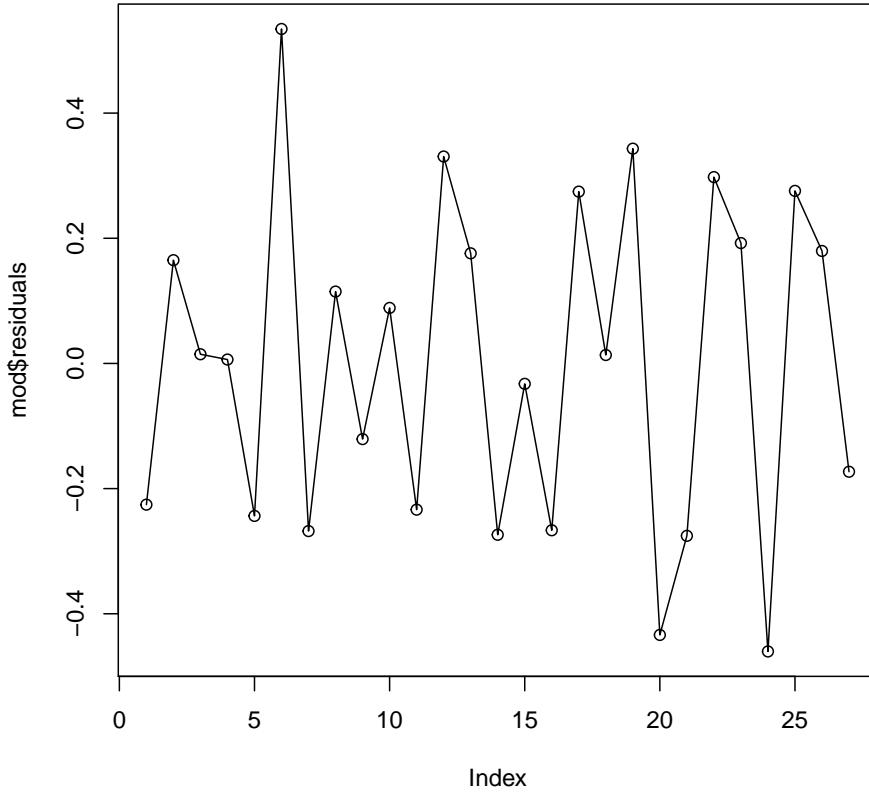
Independence is also a key assumption: it guarantees that the amount of information that we have on the relationship between  $Y$  and  $X_1, \dots, X_p$  with  $n$  observations is *maximal*. **If there is dependence, then information is repeated**, and as a consequence the variability of the estimates will be larger. For example, our 95% confidence intervals will be smaller than the *adequate*, meaning that they will not contain with a 95% confidence the unknown parameter, but with a lower confidence (say 80%).

An extreme case is the following: suppose we have two samples of sizes  $n$  and  $2n$ , where the  $2n$  sample contains the elements of the  $n$  sample twice. The information in both samples is the same, and so are the estimates for the coefficients  $\beta$ . Yet in the  $2n$  sample the length of the confidence intervals is  $C\sqrt{2n}^{-1}$ , whereas in the  $n$  sample they have length  $C\sqrt{n}^{-1}$ . A reduction by a factor of  $\sqrt{2}$  in the confidence interval has happened, but we have the same information! This will give us a wrong sense of confidence in our model, and the root of the evil was the dependence between observations.

#### How to check it

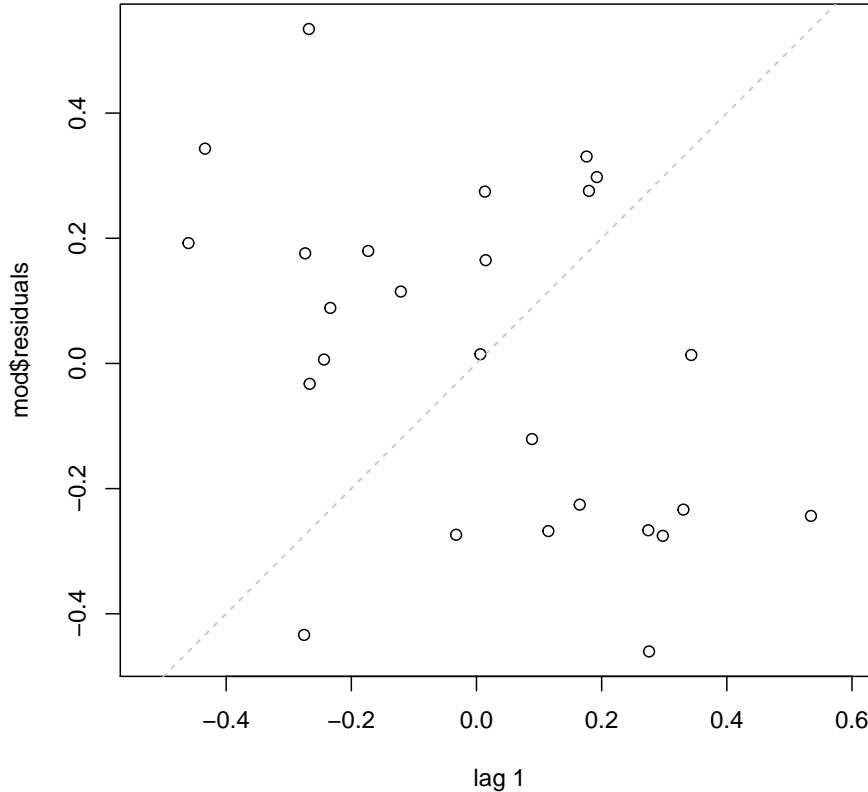
The set of possible dependence structures is immense, and there is no simple way of checking all of them. Usually what it is examined is the presence of *autocorrelation*, which appears when there is some kind of time dependence in the measurement of observations. The **serial plot of the residuals** allows to detect time trends in them.

```
plot(mod$residuals, type = "o")
```



Under uncorrelation, we expect the series to show no *tracking* of the residuals. That is, that the closer observations do not take similar values, but rather change without any kind of distinguishable pattern. Tracking is associated to positive autocorrelation, but negative, manifested as alternating small-large or positive-negative residuals, is also possible. The lagged plots of  $(\hat{\varepsilon}_{i-l}, \hat{\varepsilon}_i)$ ,  $i = l+1, \dots, n$ , obtained through `lag.plot`, allow us to detect both kinds of autocorrelations for a given lag  $l$ . Under independence, we expect no trend in such plot. Here is an example:

```
lag.plot(mod$residuals, lags = 1, do.lines = FALSE)
```



```
# No serious serial trend, but some negative autocorrelation is appreciated
cor(mod$residuals[-1], mod$residuals[-length(mod$residuals)])
## [1] -0.4267776
```

There are also formal tests for testing for the absence of autocorrelation (null hypothesis):

```
# Durbin-Watson test
durbinWatsonTest(mod)
##   lag Autocorrelation D-W Statistic p-value
##   1    -0.4160168     2.787261  0.058
## Alternative hypothesis: rho != 0
# Does not reject at alpha = 0.05
```

### What to do if fails

Little can be done if there is dependence in the data, once this has been collected. We must rely on the family of statistical models meant to deal with serial dependence: *time series*.

However, there is a simple trick worth mentioning. If the observations of the response  $Y$ , say  $Y_1, Y_2, \dots, Y_n$ , present serial dependence, a differentiation of the sample that yields  $Y_1 - Y_2, Y_2 - Y_3, \dots, Y_{n-1} - Y_n$  may lead to independent observations. These are called the *innovations* of the series of  $Y$ .



Load the dataset `assumptions3D.RData`(`download`) and compute the regressions  $y.3 \sim x1.3 + x2.3$ ,  $y.4 \sim x1.4 + x2.4$ ,  $y.5 \sim x1.5 + x2.5$ , and  $y.8 \sim x1.8 + x2.8$ . Use the presented diagnostic tools to test the assumptions of the linear model and look out for possible problems.

### 3.5.5 Multicollinearity

A common problem that arises in multiple linear regression is **multicollinearity**. This is the situation when two or more predictors are highly *linearly* related between them. Multicollinearity has important effects on the fit of the model:

- It **reduces the precision of the estimates**. As a consequence, signs of fitted coefficients may be reversed and valuable predictors may appear as non significant.
- It is **difficult to determine how each of the highly related predictors affects the response**, since one masks the other. Also, this may result in numerical instabilities because  $\mathbf{X}'\mathbf{X}$  will be close to being singular.



Intuitively, multicollinearity can be visualized as a card (fitting plane) that is held on its opposite corners and that spins on its diagonal (where the data is concentrated). Then very different planes will fit the data almost equally well, which results in a large variability of the selected plane.

An approach to detect multicollinearity is to compute the correlation matrix between the predictors by `cor`

```
cor(wine)
##                Year      Price  WinterRain      AGST HarvestRain
## Year      1.00000000 -0.4604087  0.05118354 -0.29488335 -0.05884976
## Price     -0.46040873  1.0000000  0.13488004  0.66752483 -0.50718463
## WinterRain  0.05118354  0.1348800  1.00000000 -0.32113230 -0.26798907
## AGST      -0.29488335  0.6675248 -0.32113230  1.00000000 -0.02708361
## HarvestRain -0.05884976 -0.5071846 -0.26798907 -0.02708361  1.00000000
## Age       -1.00000000  0.4604087 -0.05118354  0.29488335  0.05884976
## FrancePop  0.99227908 -0.4810720  0.02945091 -0.30126148 -0.03201463
##                Age      FrancePop
## Year      -1.00000000  0.99227908
## Price      0.46040873 -0.48107195
## WinterRain -0.05118354  0.02945091
## AGST       0.29488335 -0.30126148
## HarvestRain  0.05884976 -0.03201463
## Age        1.00000000 -0.99227908
## FrancePop -0.99227908  1.00000000
```

Here we can see what we already knew from Section 2.1, that `Age` and `Year` are perfectly linearly related and that `Age` and `FrancePop` are highly linearly related. Then one approach will be to directly remove one of the highly-correlated predictors.

However, **is not enough to inspect pair by pair correlations in order to get rid of multicollinearity**. It is possible to build counterexamples that show non suspicious pairwise correlations but problematic more complex linear relations that remain hidden. Here is one:

```
# Create predictors with multicollinearity: x4 depends on the rest
set.seed(45678)
x1 <- rnorm(100)
x2 <- 0.5 * x1 + rnorm(100)
x3 <- 0.5 * x2 + rnorm(100)
x4 <- -x1 + x2 + rnorm(100, sd = 0.25)

# Response
y <- 1 + 0.5 * x1 + 2 * x2 - 3 * x3 - x4 + rnorm(100)
data <- data.frame(x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)
```

```
# Correlations - none seems suspicious
cor(data)
##          x1          x2          x3          x4          y
## x1  1.0000000  0.38254782  0.2142011 -0.5261464  0.31194689
## x2  0.3825478  1.00000000  0.5167341  0.5673174 -0.04428223
## x3  0.2142011  0.51673408  1.0000000  0.2500123 -0.77482655
## x4 -0.5261464  0.56731738  0.2500123  1.0000000 -0.28677304
## y   0.3119469 -0.04428223 -0.7748265 -0.2867730  1.00000000
```

A better approach to detect multicollinearity is to compute the **Variance Inflation Factor** (VIF) of each coefficient  $\hat{\beta}_j$ . This is a *measure of how linearly dependent is  $X_j$  with the rest of predictors* and is defined as

$$\text{VIF}(\hat{\beta}_j) = \frac{1}{1 - R_{X_j|X_{-j}}^2}$$

where  $R_{X_j|X_{-j}}^2$  represents the  $R^2$  from the regression of  $X_j$  into the remaining predictors. The next simple rule of thumb gives direct insight into which predictors are multicollinear:

- VIF close to 1: absence of multicollinearity.
- **VIF larger than 5 or 10: problematic amount of multicollinearity.** Advised to remove the predictor with largest VIF.

VIF is computed with the `vif` function and takes as an argument a linear model. We continue with the previous example.

```
# Abnormal variance inflation factors: largest for x4, we remove it
modMultiCo <- lm(y ~ x1 + x2 + x3 + x4)
vif(modMultiCo)
##          x1          x2          x3          x4
## 26.361444 29.726498  1.416156 33.293983

# Without x4
modClean <- lm(y ~ x1 + x2 + x3)

# Comparison
compareCoefs(modMultiCo, modClean)
##
## Call:
## 1: lm(formula = y ~ x1 + x2 + x3 + x4)
## 2: lm(formula = y ~ x1 + x2 + x3)
##          Est. 1    SE 1  Est. 2    SE 2
## (Intercept) 1.062  0.103  1.058  0.103
## x1          0.922  0.551  1.450  0.116
## x2          1.640  0.546  1.119  0.124
## x3         -3.165  0.109 -3.145  0.107
## x4         -0.529  0.541
confint(modMultiCo)
##             2.5 %    97.5 %
## (Intercept) 0.8568419 1.2674705
## x1          -0.1719777 2.0167093
## x2          0.5556394 2.7240952
## x3         -3.3806727 -2.9496676
## x4         -1.6030032  0.5446479
confint(modClean)
##             2.5 %    97.5 %
## (Intercept) 0.8526681 1.262753
```

```

## x1          1.2188737  1.680188
## x2          0.8739264  1.364981
## x3         -3.3564513 -2.933473

# Summaries
summary(modMultiCo)
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x4)
##
## Residuals:
##    Min     1Q  Median     3Q    Max
## -1.9762 -0.6663  0.1195  0.6217  2.5568
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.0622    0.1034 10.270 < 2e-16 ***
## x1          0.9224    0.5512  1.673  0.09756 .
## x2          1.6399    0.5461  3.003  0.00342 **
## x3         -3.1652    0.1086 -29.158 < 2e-16 ***
## x4          -0.5292    0.5409 -0.978  0.33040
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.028 on 95 degrees of freedom
## Multiple R-squared:  0.9144, Adjusted R-squared:  0.9108
## F-statistic: 253.7 on 4 and 95 DF,  p-value: < 2.2e-16
summary(modClean)
##
## Call:
## lm(formula = y ~ x1 + x2 + x3)
##
## Residuals:
##    Min     1Q  Median     3Q    Max
## -1.91297 -0.66622  0.07889  0.65819  2.62737
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.0577    0.1033 10.24 < 2e-16 ***
## x1          1.4495    0.1162 12.47 < 2e-16 ***
## x2          1.1195    0.1237  9.05 1.63e-14 ***
## x3         -3.1450    0.1065 -29.52 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.028 on 96 degrees of freedom
## Multiple R-squared:  0.9135, Adjusted R-squared:  0.9108
## F-statistic: 338 on 3 and 96 DF,  p-value: < 2.2e-16

# Variance inflation factors are normal
vif(modClean)
##      x1      x2      x3
## 1.171942 1.525501 1.364878

```



Note that multicollinearity is another instance of the **model correctness vs. usefulness**. A model with multicollinearity might be perfectly valid in the sense of respecting the assumptions of the model. As we saw in Section 2.3, it does not matter whether the predictors are related or not. At least for the verification of the assumptions. But the model will be useless if the multicollinearity is high, since it can inflate the variability of the estimation without any kind of bound.

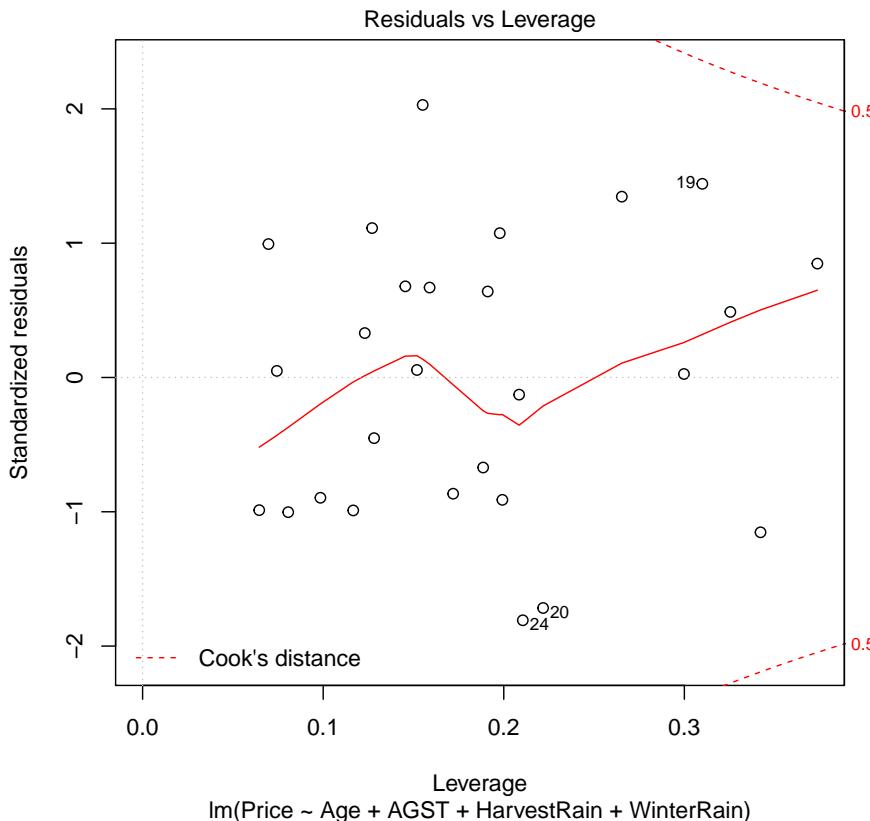
### 3.5.6 Outliers and high-leverage points

Outliers and high-leverage points are particular observations that have a large impact in the final linear model, either on the estimates or on the properties of the model.

- **Outliers** are the observations with a *response  $Y_i$  far away from the regression plane*. They typically do not affect the estimate of the plane, unless one of the predictors is also extreme (see next point). But they affect  $\hat{\sigma}$  and as a consequence the  $R^2$  of the model (drawing it down) and the confidence intervals (expanding them).
- **High-leverage points** are observations with an *extreme predictor  $X_{ij}$  located far away from the rest of points*. These observations are highly influential and may drive the fitting of the linear model. The reason is the squared distance in the RSS: an individual extremal point contributes a large portion of the RSS.

Both outliers and high-leverage points can be identified with the **residuals vs leverage plot**:

```
plot(mod, 5)
```



The rule of thumb for declaring outliers and high-leverage points are:

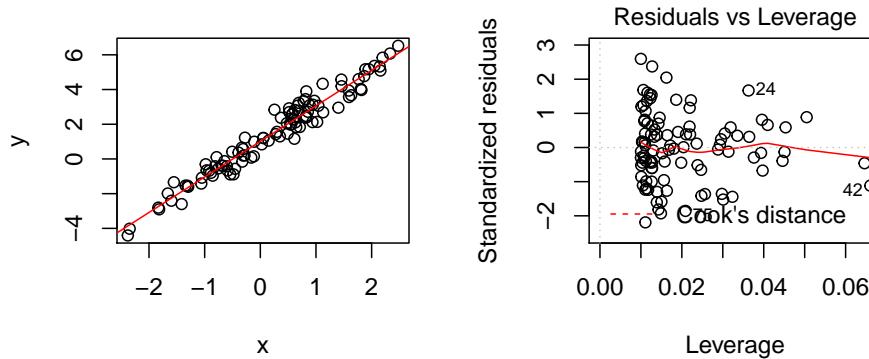
- If the standardized residual of an observation is larger than 3 in absolute value, then it may be an outlier.
- If the leverage statistic is *greatly exceeding*  $(p + 1)/n$ , then it may be suspected of having a high leverage.

Let's see an artificial example.

```
# Create data
set.seed(12345)
x <- rnorm(100)
e <- rnorm(100, sd = 0.5)
y <- 1 + 2 * x + e

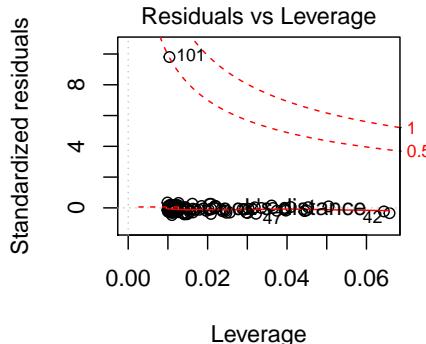
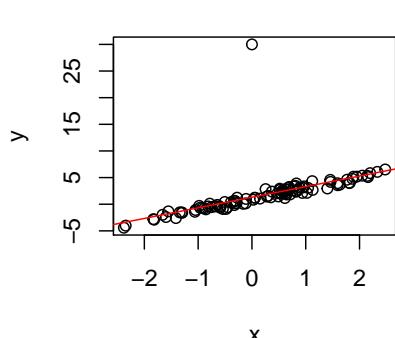
# Leverage expected value
2 / 101 # (p + 1) / n
## [1] 0.01980198

# Base model
m0 <- lm(y ~ x)
par(mfrow = c(1, 2))
plot(x, y)
abline(coef = m0$coefficients, col = 2)
plot(m0, 5)
```



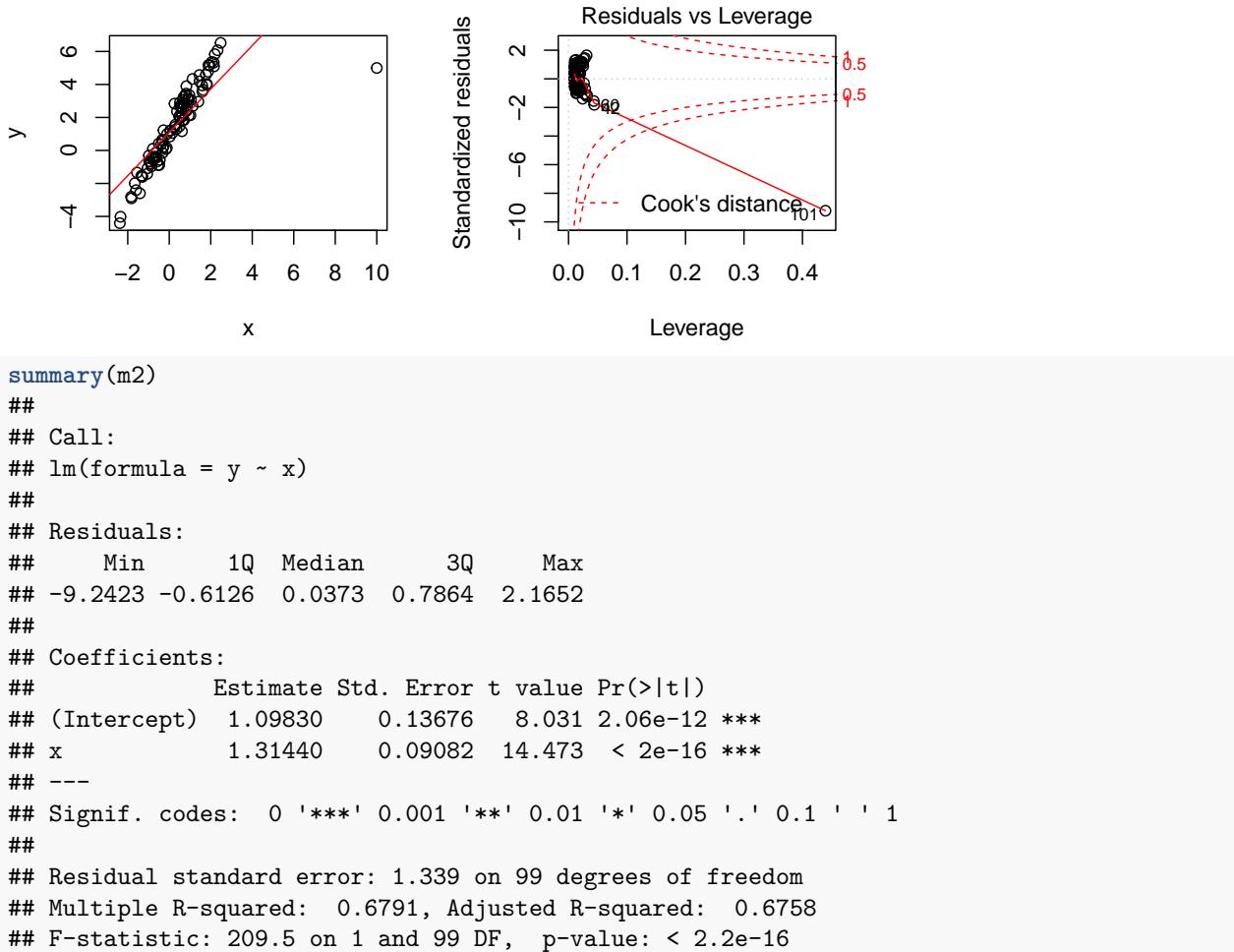
```
summary(m0)
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min      1Q      Median      3Q      Max
## -1.10174 -0.30139 -0.00557  0.30949  1.30485
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.01103   0.05176 19.53   <2e-16 ***
## x           2.04727   0.04557 44.93   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5054 on 98 degrees of freedom
## Multiple R-squared:  0.9537, Adjusted R-squared:  0.9532
## F-statistic: 2018 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
# Make an outlier
x[101] <- 0; y[101] <- 30
m1 <- lm(y ~ x)
par(mfrow = c(1, 2))
plot(x, y)
abline(coef = m1$coefficients, col = 2)
plot(m1, 5)
```



```
summary(m1)
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min      1Q  Median      3Q      Max
## -1.3676 -0.5730 -0.2955  0.0941 28.6881
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.3119    0.2997  4.377 2.98e-05 ***
## x           1.9901    0.2652  7.505 2.71e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.942 on 99 degrees of freedom
## Multiple R-squared:  0.3627, Adjusted R-squared:  0.3562
## F-statistic: 56.33 on 1 and 99 DF,  p-value: 2.708e-11

# Make a high-leverage point
x[101] <- 10; y[101] <- 5
m2 <- lm(y ~ x)
par(mfrow = c(1, 2))
plot(x, y)
abline(coef = m2$coefficients, col = 2)
plot(m2, 5)
```



The leverage statistic associated to the  $i$ -th datum corresponds to the  $i$ -th diagonal entry of the hat matrix  $\mathbf{H}$ :

$$h_i := \mathbf{H}_{ii} = (\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}')_{ii}$$

and it can be seen that  $\frac{1}{n} \leq h_i \leq 1$  and that the mean  $\bar{h} = \frac{1}{n} \sum_{i=1}^n h_i = \frac{p+1}{n}$ . This can be clearly seen in the case of simple linear regression, where the leverage statistic has the explicit form

$$h_i = \frac{1}{n} + \frac{(X_i - \bar{X})^2}{\sum_{j=1}^n (X_j - \bar{X})^2}.$$

Interestingly, this expression shows that the leverage statistic is directly dependent on the distance to the center of the predictor. A measure of how much  $h_i$  exceeds the expected value  $\bar{h}$  can be given if the predictors are assumed to be jointly normal. In this case,  $nh_i - 1 \sim \chi_p^2$  (Peña, 2002) and hence the  $i$ -th point is declared as a *potential high-leverage point* if  $h_i > \frac{\chi_{p;\alpha}^2 + 1}{n}$ , where  $\chi_{p;\alpha}^2$  is the  $\alpha$ -upper quantile of the  $\chi_p^2$  distribution and  $\alpha$  can be taken as 0.05 or 0.01.

The functions `influence`, `hat`, and `rstandard` allow to have a finer inspection of the leverage statistics

```
# Access leverage statistics
influence(model = m2, do.coef = FALSE)$hat
##      1          2          3          4          5          6
## 0.010174492 0.010523331 0.010837656 0.012812436 0.010222087 0.031373039
##      7          8          9         10         11         12
## 0.010283656 0.011658906 0.011704575 0.017222024 0.010866718 0.019923351
```

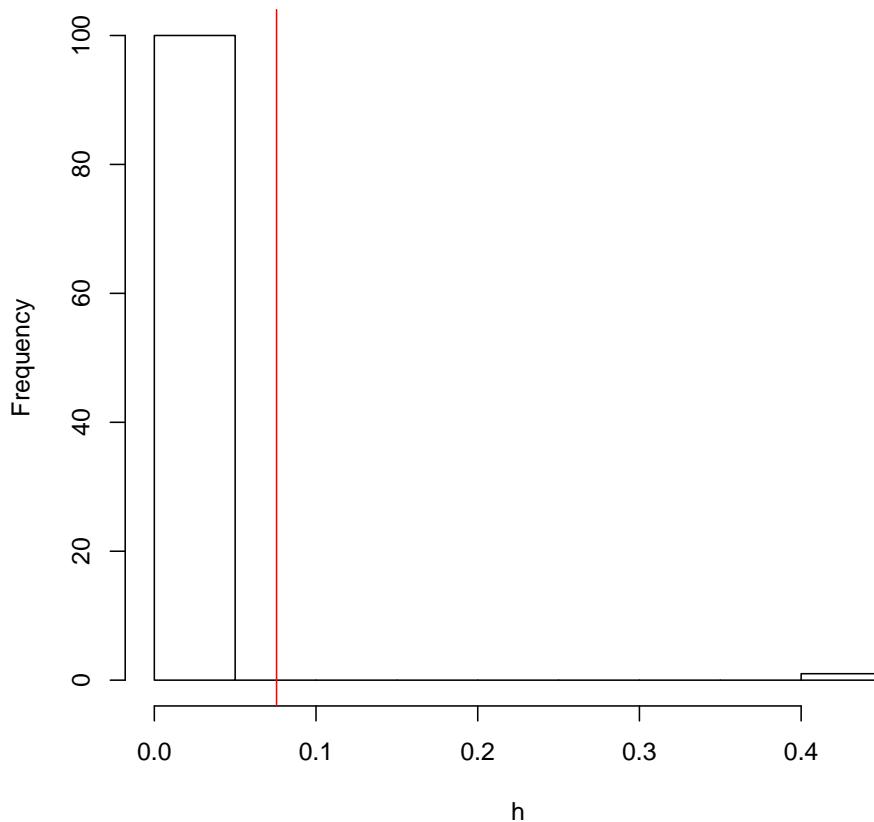
```

##      13      14      15      16      17      18
## 0.009904821 0.010047559 0.015393430 0.010940144 0.016844291 0.011988186
##      19      20      21      22      23      24
## 0.012694003 0.009909524 0.010783477 0.015613766 0.014377311 0.026430223
##      25      26      27      28      29      30
## 0.027216968 0.019758107 0.013022196 0.010258292 0.010237429 0.011070730
##      31      32      33      34      35      36
## 0.010918272 0.025742099 0.023320867 0.017569312 0.009936241 0.010003750
##      37      38      39      40      41      42
## 0.011942004 0.028384907 0.019261160 0.010360598 0.012750206 0.044011838
##      43      44      45      46      47      48
## 0.018949901 0.011532664 0.011110898 0.015664589 0.024077403 0.010135328
##      49      50      51      52      53      54
## 0.010169263 0.022411976 0.013483377 0.021772790 0.010283311 0.009901440
##      55      56      57      58      59      60
## 0.014622512 0.009919743 0.010462941 0.010970526 0.024870289 0.043179560
##      61      62      63      64      65      66
## 0.010071019 0.022960201 0.010106954 0.017072808 0.013870939 0.031660750
##      67      68      69      70      71      72
## 0.011275128 0.017113380 0.010042088 0.022243659 0.010280595 0.015741892
##      73      74      75      76      77      78
## 0.018809585 0.028107436 0.015082336 0.011562731 0.010981484 0.016024099
##      79      80      81      82      83      84
## 0.009984227 0.012026311 0.010325303 0.012165423 0.011822918 0.030890535
##      85      86      87      88      89      90
## 0.011724814 0.020611107 0.010403093 0.011844300 0.010075573 0.010975300
##      91      92      93      94      95      96
## 0.017748764 0.016495166 0.020891645 0.012378353 0.017951181 0.010450659
##      97      98      99     100     101
## 0.013202082 0.025081124 0.013982154 0.014844840 0.439305551

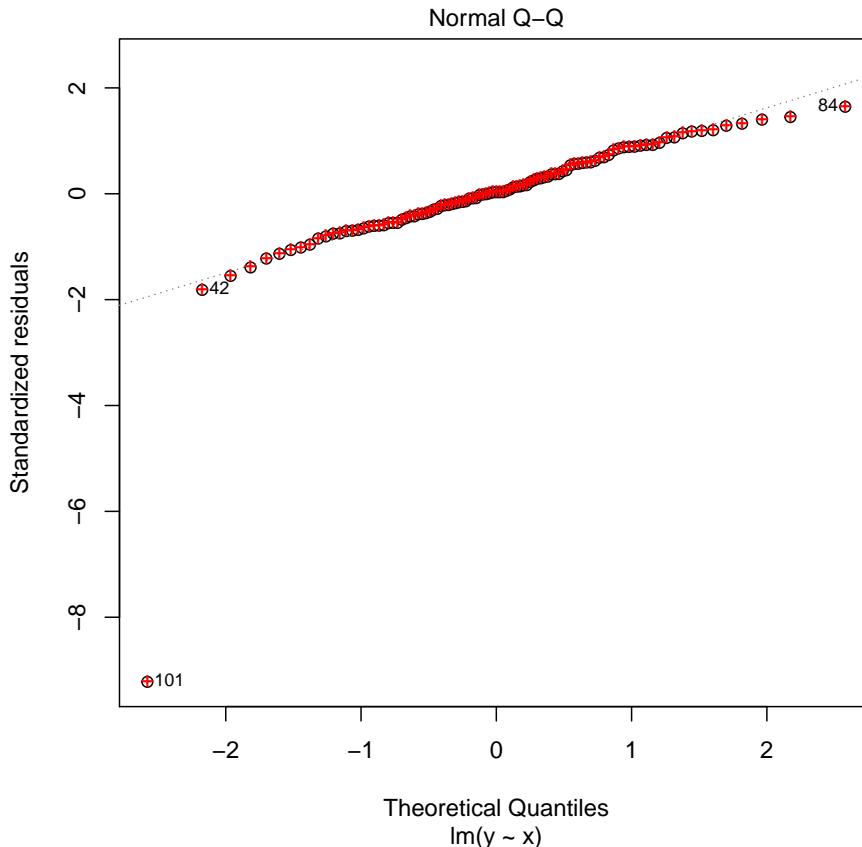
# Another option
h <- hat(x = x)

# 1% most influential points
n <- length(x)
p <- 1
hist(h)
abline(v = (qchisq(0.99, df = p) + 1) / n, col = 2)

```

**Histogram of  $h$** 

```
# Standardized residuals
rs <- rstandard(m2)
plot(m2, 2) # QQ-plot
points(qnorm(ppoints(n = n)), sort(rs), col = 2, pch = '+') # Manually computed
```



### 3.5.7 Case study application



*Moore's law* (Moore, 1965) is an empirical law that states that the power of a computer doubles approximately every two years. Translated into a mathematical formula, Moore's law is

$$\text{transistors} \approx 2^{\text{years}/2}.$$

Applying logarithms to both sides gives

$$\log(\text{transistors}) \approx \frac{\log(2)}{2} \text{years}.$$

We can write the above formula more generally as

$$\log(\text{transistors}) = \beta_0 + \beta_1 \text{years} + \varepsilon,$$

where  $\varepsilon$  is a random error. This is a linear model!

The dataset `cpus.txt` (download) contains the transistor counts for the CPUs appeared in the time range 1971–2015. For this data, do the following:

- Import conveniently the data and name it as `cpus`.
- Show a scatterplot of `Transistor.count` vs `Date.of.introduction` with a linear regression.
- Are the assumptions verified in `Transistor.count ~ Date.of.introduction`? Which ones are which are more “problematic”?
- Create a new variable, named `Log.Transistor.count`, containing the logarithm of `Transistor.count`.
- Show a scatterplot of `Log.Transistor.count` vs `Date.of.introduction` with a linear regression.
- Are the assumptions verified in `Log.Transistor.count ~ Date.of.introduction`? Which ones are which are more “problematic”?
- Regress `Log.Transistor.count ~ Date.of.introduction`.
- Summarize the fit. What are the estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$ ? Is  $\hat{\beta}_1$  close to  $\frac{\log(2)}{2}$ ?
- Compute the CI for  $\beta_1$  at  $\alpha = 0.05$ . Is  $\frac{\log(2)}{2}$  inside it? What happens at levels  $\alpha = 0.10, 0.01$ ?
- We want to forecast the average log-number of transistors for the CPUs to be released in 2017. Compute the adequate prediction and CI.
- A new CPU design is expected for 2017. What is the range of log-number of transistors expected for it, at a 95% level of confidence?
- Compute the ANOVA table for `Log.Transistor.count ~ Date.of.introduction`. Is  $\beta_1$  significant?



The dataset `gpus.txt` (download) contains the transistor counts for the GPUs appeared in the period 1997–2016. Repeat the previous analysis for this dataset.

## 3.6 Dimension reduction techniques

As we have seen in Section 3.2, the selection of the best linear model from a set of  $p$  predictors is a challenging task that increases with the *dimension* of the problem, that is, with  $p$ . In addition to the expansion of the set of possible models as  $p$  grows, the model space becomes more and more complicated to explore due to the potential multicollinearity among the predictors. We will see in this section two methods to deal with these two problems simultaneously.

### 3.6.1 Review on principal component analysis

Principal Component Analysis (PCA) is a multivariate technique designed to summarize the most important features and relations of  $p$  numerical random variables  $X_1, \dots, X_p$ . PCA computes a new set of variables, the **principal components**  $PC_1, \dots, PC_p$ , that contain the same information as  $X_1, \dots, X_p$  but expressed in a more convenient way. The goal is to retain only a limited number  $1 \leq l < p$  of principal components such that they explain most of the information and do *dimension reduction*.

If  $X_1, \dots, X_p$  are *centred* (this is,  $\mathbb{E}[X_j] = 0$ ,  $j = 1, \dots, p$  – this is important since PCA is sensitive to the centering of the data), then the principal components are *orthonormal linear combinations* of  $X_1, \dots, X_p$ :

$$PC_j := a_{1j}X_1 + a_{2j}X_2 + \dots + a_{pj}X_p = \mathbf{a}'_j \mathbf{X}, \quad j = 1, \dots, p, \quad (3.2)$$

where  $\mathbf{a}_j := (a_{1j}, \dots, a_{pj})'$ ,  $\mathbf{X} := (X_1, \dots, X_p)'$ , and the *orthonormality* condition is

$$\mathcal{D}'_i \mathcal{D}_j = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Remarkably, PCA computes the principal components in an *ordered* way:  $\text{PC}_1$  is the principal component that explains the most of the *information* (quantified as the variance) of  $X_1, \dots, X_p$ , and then the explained information decreases monotonically down to  $\text{PC}_p$ , the principal component that explains the least information. Precisely:

$$\text{Var}[\text{PC}_1] \geq \text{Var}[\text{PC}_2] \geq \dots \geq \text{Var}[\text{PC}_p]. \quad (3.3)$$

Mathematically, PCA reduces to compute the *spectral decomposition*<sup>5</sup> of the covariance matrix  $\Sigma := \text{Var}[\mathbf{X}]$ :

$$\Sigma = \mathbf{A}' \Lambda \mathbf{A},$$

where  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_p)$  contains the eigenvalues of  $\Sigma$  and  $\mathbf{A}$  is the orthogonal matrix that contains the unit-norm eigenvectors of  $\Sigma$  as columns.  $\mathbf{A}$  gives, thus, the coefficients of the orthonormal linear combinations:

$$\mathbf{A} = (\mathbf{a}_1 \quad \mathbf{a}_2 \quad \dots \quad \mathbf{a}_p) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \dots & a_{pp} \end{pmatrix}.$$

If the data is not centered, the computation of the principal components is done by first subtracting  $\mu = \mathbb{E}[\mathbf{X}]$  and then premultiplying with  $\mathbf{A}^{-1}$ :

$$\mathbf{PC} = \mathbf{A}'(\mathbf{X} - \mu), \quad (3.4)$$

where  $\mathbf{PC} = (\text{PC}_1, \dots, \text{PC}_p)'$ ,  $\mathbf{X} = (X_1, \dots, X_p)'$ , and  $\mu = (\mu_1, \dots, \mu_p)'$ . From (3.4), it is evident than we can express the random variables  $\mathbf{X}$  in terms of  $\mathbf{PC}$ :

$$\mathbf{X} = \mu + \mathbf{APC}, \quad (3.5)$$

which admits an insightful interpretation: the  $\mathbf{PC}$  are uncorrelated variables that, once *rotated* by  $\mathbf{A}$  and *translated* to the location  $\mu$ , produce exactly  $\mathbf{X}$ .

From (3.4), the linearity of the expectation (1.2), and the spectral decomposition, it is trivial to show that: (i) (3.3) holds; (ii) the **principal components are centred and uncorrelated between them**:

$$\mathbb{E}[\text{PC}_i] = 0 \text{ and } \text{Cov}[\text{PC}_i, \text{PC}_j] = \begin{cases} \lambda_i, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

The **proportion of variance explained** by  $l \leq p$  principal components is  $\frac{\sum_{j=1}^l \lambda_j}{\sum_{j=1}^p \lambda_j}$ .

In the sample case where  $\mathbf{X}_1, \dots, \mathbf{X}_n$  is given and  $\mu$  and  $\Sigma$  are unknown,  $\mu$  is replaced by  $\bar{\mathbf{X}}$  and  $\Sigma$  by  $\mathbf{S} = \frac{1}{n} \sum_{i=1}^n (\mathbf{X}_i - \bar{\mathbf{X}})(\mathbf{X}_i - \bar{\mathbf{X}})'$ , and then the spectral decomposition of  $\mathbf{S}$  is computed. This gives  $\hat{\mathbf{A}}$  and produces the *scores*

$$\hat{\mathbf{PC}}_1 := \hat{\mathbf{A}}'(\mathbf{X}_1 - \bar{\mathbf{X}}), \dots, \hat{\mathbf{PC}}_n := \hat{\mathbf{A}}'(\mathbf{X}_n - \bar{\mathbf{X}}),$$

a transformation of the sample that makes the scores *centred, uncorrelated*, and with sample *variances* in each vector's entry *sorted* in a decreasing way.

<sup>5</sup>Recall that the covariance matrix is a real, symmetric, semi-positive definite matrix.



We implicitly assume that  $n > p$ . Otherwise, the maximum number of principal components is  $\min(n - 1, p)$ .

Let's see an example of these concepts in *La Liga 2015/2016* (download) dataset. It contains the standings and team statistics for La Liga 2015/2016.

```
library(readxl)
laliga <- read_excel("la-liga-2015-2016.xlsx", sheet = 1, col_names = TRUE)
laliga <- as.data.frame(laliga) # Avoid tibble since it drops row.names
```

A quick preprocessing gives:

```
rownames(laliga) <- laliga$Team # Set teams as case names
laliga$Team <- NULL
laliga <- laliga[, -c(2, 8)] # Do not add relevant information
summary(laliga)
##          Points          Wins          Draws          Loses
##  Min.   :32.00   Min.   : 8.00   Min.   : 4.00   Min.   : 4.00
##  1st Qu.:41.25   1st Qu.:10.00   1st Qu.: 8.00   1st Qu.:12.00
##  Median :44.50   Median :12.00   Median : 9.00   Median :15.50
##  Mean   :52.40   Mean   :14.40   Mean   : 9.20   Mean   :14.40
##  3rd Qu.:60.50   3rd Qu.:17.25   3rd Qu.:10.25   3rd Qu.:18.25
##  Max.   :91.00   Max.   :29.00   Max.   :18.00   Max.   :22.00
##          Goals.scored      Goals.conceded      Percentage.scored.goals
##  Min.   : 34.00   Min.   :18.00   Min.   :0.890
##  1st Qu.: 40.00   1st Qu.:42.50   1st Qu.:1.050
##  Median : 45.50   Median :52.50   Median :1.195
##  Mean   : 52.15   Mean   :52.15   Mean   :1.371
##  3rd Qu.: 51.25   3rd Qu.:63.25   3rd Qu.:1.347
##  Max.   :112.00   Max.   :74.00   Max.   :2.950
##          Percentage.conceded.goals      Shots          Shots.on.goal
##  Min.   :0.470          Min.   :346.0   Min.   :129.0
##  1st Qu.:1.115          1st Qu.:413.8   1st Qu.:151.2
##  Median :1.380          Median :438.0   Median :165.0
##  Mean   :1.371          Mean   :452.4   Mean   :173.1
##  3rd Qu.:1.663          3rd Qu.:455.5   3rd Qu.:180.0
##  Max.   :1.950          Max.   :712.0   Max.   :299.0
##          Penalties.scored      Assistances      Fouls.made
##  Min.   : 1.00   Min.   :23.00   Min.   :385.0
##  1st Qu.: 1.00   1st Qu.:28.50   1st Qu.:483.8
##  Median : 3.00   Median :32.50   Median :530.5
##  Mean   : 3.45   Mean   :37.85   Mean   :517.6
##  3rd Qu.: 4.50   3rd Qu.:36.75   3rd Qu.:552.8
##  Max.   :11.00   Max.   :90.00   Max.   :654.0
##          Matches.without.conceding      Yellow.cards      Red.cards          Offsides
##  Min.   : 4.0          Min.   : 66.0   Min.   :1.00   Min.   : 72.00
##  1st Qu.: 7.0          1st Qu.: 97.0   1st Qu.:4.00   1st Qu.: 83.25
##  Median :10.5          Median :108.5   Median :5.00   Median : 88.00
##  Mean   :10.7          Mean   :106.2   Mean   :5.05   Mean   : 92.60
##  3rd Qu.:13.0          3rd Qu.:115.2   3rd Qu.:6.00   3rd Qu.:103.75
##  Max.   :24.0          Max.   :141.0   Max.   :9.00   Max.   :123.00
```

Let's check that R's function for PCA, `princomp`, returns the same principal components we outlined in the theory.

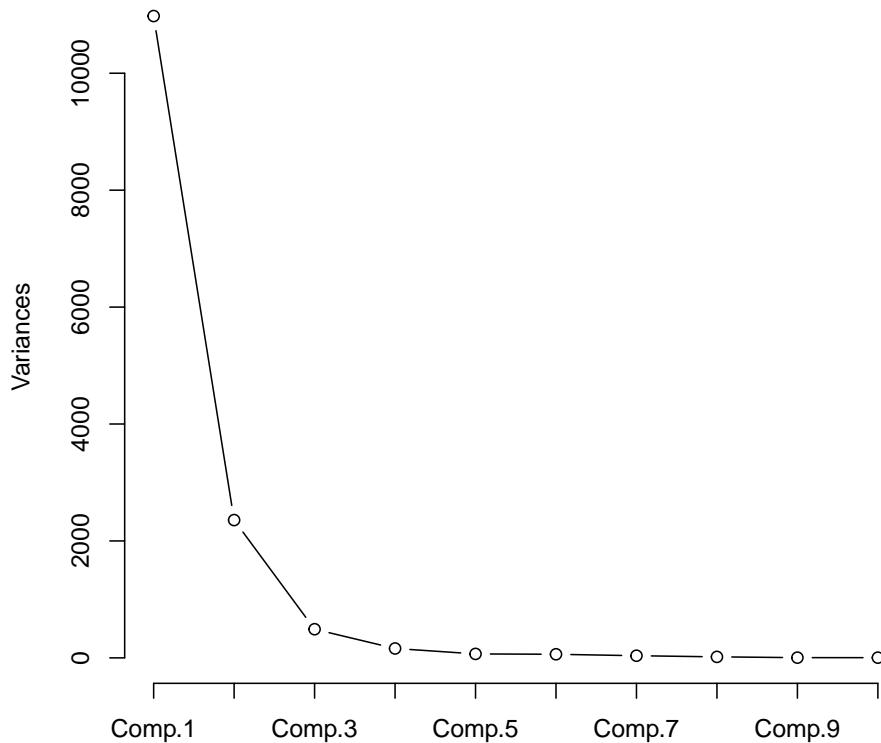
```

# PCA
pcaLaliga <- princomp(laliga)
summary(pcaLaliga)
## Importance of components:
##           Comp.1    Comp.2    Comp.3    Comp.4
## Standard deviation 104.7782561 48.5461449 22.13337511 12.66692413
## Proportion of Variance 0.7743008 0.1662175 0.03455116 0.01131644
## Cumulative Proportion 0.7743008 0.9405183 0.97506949 0.98638593
##           Comp.5    Comp.6    Comp.7    Comp.8
## Standard deviation 8.234215354 7.83426116 6.068864168 4.137079559
## Proportion of Variance 0.004782025 0.00432876 0.002597659 0.001207133
## Cumulative Proportion 0.991167955 0.99549671 0.998094374 0.999301507
##           Comp.9    Comp.10   Comp.11   Comp.12
## Standard deviation 2.0112480391 1.8580509157 1.126111e+00 9.568824e-01
## Proportion of Variance 0.0002852979 0.0002434908 8.943961e-05 6.457799e-05
## Cumulative Proportion 0.9995868048 0.9998302956 9.999197e-01 9.999843e-01
##           Comp.13   Comp.14   Comp.15   Comp.16
## Standard deviation 4.716064e-01 1.707105e-03 8.365534e-04 5.867584e-07
## Proportion of Variance 1.568652e-05 2.055361e-10 4.935768e-11 2.428208e-17
## Cumulative Proportion 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##           Comp.17
## Standard deviation 0
## Proportion of Variance 0
## Cumulative Proportion 1

# The standard deviations are the square roots of the eigenvalues
# The cumulative proportion of variance explained accumulates the
# variance explained starting at the first component

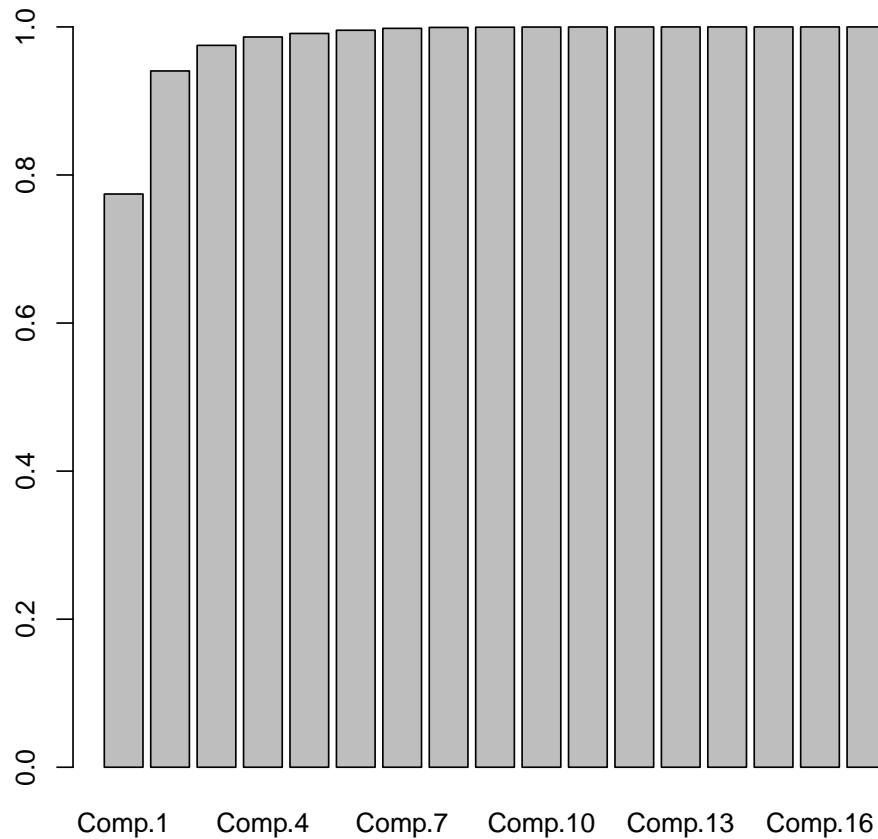
# Plot of variances of each component (screeplot)
plot(pcaLaliga, type = "1")

```

**pcaLaliga**

```
# Useful for detecting an "elbow" in the graph whose location gives the
# "right" number of components to retain. Ideally, this elbow appears
# when the next variances are almost similar and notably smaller when
# compared with the previous
```

```
# Plot of the cumulated percentage of variance
barplot(cumsum(pcaLaliga$sdev^2) / sum(pcaLaliga$sdev^2))
```



```

# Computation of PCA from the spectral decomposition
n <- nrow(laliga)
eig <- eigen(cov(laliga) * (n - 1) / n)
A <- eig$vectors

# Same eigenvalues
pcaLaliga$sdev^2 - eig$values
##          Comp.1          Comp.2          Comp.3          Comp.4          Comp.5
## -1.818989e-12  2.728484e-12  2.273737e-13 -2.273737e-13  5.684342e-14
##          Comp.6          Comp.7          Comp.8          Comp.9          Comp.10
##  3.552714e-14  4.263256e-14 -3.552714e-14  1.234568e-13  5.373479e-14
##          Comp.11         Comp.12         Comp.13         Comp.14         Comp.15
##  1.199041e-14  1.054712e-14  1.049161e-14 -3.709614e-15 -2.191892e-15
##          Comp.16         Comp.17
##  4.476020e-13  2.048814e-12

# The eigenvectors are in $loadings
pcaLaliga$loadings
##
## Loadings:
##          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7
## Points      -0.125      0.497  0.195  0.139 -0.340
## Wins        0.184      -0.175
## Draws        0.101  0.186
## Loses       -0.129  -0.114
## Goals.scored -0.181      0.251 -0.186 -0.169 -0.399  0.335
## Goals.conceded -0.471 -0.493 -0.277 -0.257  0.280

```

```

## Percentage.scored.goals
## Percentage.conceded.goals
## Shots -0.718 -0.442 -0.342  0.255  0.241       0.188
## Shots.on.goal -0.386 -0.213  0.182 -0.287 -0.532  0.163 -0.599
## Penalties.scored
## Assists -0.148       0.198       -0.173 -0.362  0.216
## Fouls.made  0.480 -0.844  0.166       0.110
## Matches.without.conceding  0.151  0.129       -0.182
## Yellow.cards  0.141 -0.144 -0.363  0.113  0.225 -0.637 -0.550
## Red.cards
## Offsides -0.108  0.202 -0.696  0.647       -0.106
## Comp.8 Comp.9 Comp.10 Comp.11 Comp.12 Comp.13
## Points  0.425       0.379  0.129  0.166  0.138
## Wins  0.134  0.198       0.139  0.147
## Draws -0.608  0.175  0.185 -0.251 -0.304
## Loses -0.157  0.410 -0.243 -0.166  0.112  0.156
## Goals.scored -0.603  0.155  0.129  0.289 -0.230 -0.153
## Goals.conceded  0.441  0.118  0.297

## Percentage.scored.goals
## Percentage.conceded.goals
## Shots
## Shots.on.goal
## Penalties.scored -0.350 -0.258  0.378 -0.661  0.456 -0.114
## Assists  0.215 -0.356 -0.685 -0.265       -0.102
## Fouls.made
## Matches.without.conceding  0.176  0.369       -0.376 -0.411 -0.664
## Yellow.cards -0.126 -0.156
## Red.cards  0.123 -0.157  0.405  0.666 -0.587
## Offsides
## Comp.14 Comp.15 Comp.16 Comp.17
## Points  0.278 -0.315
## Wins  0.907
## Draws  0.526  0.277
## Loses  0.803
## Goals.scored
## Goals.conceded
## Percentage.scored.goals  0.760 -0.650
## Percentage.conceded.goals -0.650 -0.760
## Shots
## Shots.on.goal
## Penalties.scored
## Assists
## Fouls.made
## Matches.without.conceding
## Yellow.cards
## Red.cards
## Offsides
##
## Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.059  0.059  0.059  0.059  0.059  0.059  0.059
## Cumulative Var  0.059  0.118  0.176  0.235  0.294  0.353  0.412  0.471
## Comp.9 Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15

```

```

## SS loadings    1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var 0.059  0.059  0.059  0.059  0.059  0.059  0.059
## Cumulative Var 0.529  0.588  0.647  0.706  0.765  0.824  0.882
##                  Comp.16 Comp.17
## SS loadings    1.000  1.000
## Proportion Var 0.059  0.059
## Cumulative Var 0.941  1.000

# The scores is the representation of the data in the principal
# components - it has the same information as laliga
head(pcaLaliga$scores)
##                  Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
## Barcelona      -242.23916 21.581016 25.380103 -17.4375054 -7.1797218
## Real Madrid    -313.60263 -63.202402 -8.756998  8.5557906  0.7119181
## Atlético Madrid -45.99393  0.646609 38.540964 31.3888316  3.9162812
## Villarreal     96.22013  42.932869 50.003639 -11.2420481 10.4732634
## Athletic       -14.51728 16.189672 18.884019 -0.4122161 -5.6491352
## Celta          13.07483  -6.792525 5.227118 -9.0945489  6.1264750
##                  Comp.6    Comp.7    Comp.8    Comp.9    Comp.10
## Barcelona     -9.0814106 -5.5920449 -7.3615386  0.3715688  1.7160752
## Real Madrid   -0.2221379  6.7034894  2.4455971 -1.8388132 -2.9660592
## Atlético Madrid -3.2904137  0.2431925  5.0912667  3.0444029  2.0974553
## Villarreal    -2.4293930 -3.0183049  0.1958417 -1.2106025 -1.7453143
## Athletic       6.9329640  8.0652665  2.4783231  2.6920566  0.8950389
## Celta          -11.8794638 -2.6148154  6.9706627 -3.0825781  0.3129865
##                  Comp.11   Comp.12   Comp.13   Comp.14
## Barcelona     0.0264937 -1.0948280  0.15160351  0.0010244179
## Real Madrid   -0.1344557  0.3409538 -0.03316355 -0.0014744734
## Atlético Madrid 0.6771343 -0.3985625 -0.18088616 -0.0004984680
## Villarreal    0.1350586 -0.5735057 -0.58936061  0.0001067413
## Athletic       0.1542005  1.4714997  0.11090055  0.0031346887
## Celta          0.0859623  1.9159241  0.37219921  0.0017960697
##                  Comp.15   Comp.16   Comp.17
## Barcelona     -0.0002177801 -1.047044e-12  7.623799e-12
## Real Madrid   0.0003891502  1.587123e-12 -1.215672e-11
## Atlético Madrid -0.0001116115  6.252046e-13 -1.090230e-12
## Villarreal    -0.0002431758  3.319980e-13  3.406707e-12
## Athletic      -0.0002557828 -3.696130e-12  1.895661e-11
## Celta          0.0002911046 -2.150767e-12  5.513489e-12

# Uncorrelated
head(cov(pcaLaliga$scores))
##                  Comp.1    Comp.2    Comp.3    Comp.4
## Comp.1 1.155630e+04 4.862280e-13 2.162019e-12 1.691176e-13
## Comp.2 4.862280e-13 2.480767e+03 -2.474629e-14 7.954113e-13
## Comp.3 2.162019e-12 -2.474629e-14 5.156698e+02 4.033323e-14
## Comp.4 1.691176e-13 7.954113e-13 4.033323e-14 1.688958e+02
## Comp.5 2.320498e-13 1.969543e-13 4.108336e-14 -7.016555e-14
## Comp.6 3.407172e-13 -3.880511e-13 -2.816775e-13 6.299365e-14
##                  Comp.5    Comp.6    Comp.7    Comp.8
## Comp.1 2.320498e-13 3.407172e-13 -3.393733e-13 -5.877141e-13
## Comp.2 1.969543e-13 -3.880511e-13 -3.803029e-13 2.470739e-14
## Comp.3 4.108336e-14 -2.816775e-13 6.386850e-14 1.204495e-13

```

```

## Comp.4 -7.016555e-14 6.299365e-14 -1.758549e-13 1.987112e-13
## Comp.5 7.137084e+01 9.547836e-14 -1.571647e-13 6.681780e-14
## Comp.6 9.547836e-14 6.460595e+01 1.234172e-13 -3.102085e-14
## Comp.9 Comp.10 Comp.11 Comp.12
## Comp.1 -7.527520e-13 -2.976234e-13 1.188877e-13 -1.846714e-13
## Comp.2 2.084589e-13 -2.995121e-13 -2.767091e-13 2.010033e-13
## Comp.3 1.046232e-13 -7.781915e-14 -1.233361e-13 9.994363e-14
## Comp.4 1.844897e-14 -2.167893e-14 -5.243170e-14 4.235190e-14
## Comp.5 -1.560429e-14 5.633332e-15 -7.068359e-15 3.765278e-14
## Comp.6 2.964049e-15 6.960359e-14 6.971482e-14 -6.595763e-14
## Comp.13 Comp.14 Comp.15 Comp.16
## Comp.1 -2.198476e-13 -1.381159e-14 4.397464e-15 -1.719781e-13
## Comp.2 1.670964e-13 1.002181e-14 -1.438156e-14 5.351418e-13
## Comp.3 7.047013e-14 2.755567e-15 1.519213e-15 1.495199e-13
## Comp.4 3.733297e-14 2.658286e-15 -1.056732e-16 1.093756e-13
## Comp.5 3.783212e-14 4.175649e-15 -8.125894e-15 1.049351e-13
## Comp.6 -6.084523e-14 -6.881101e-15 7.734621e-16 -6.341222e-14
## Comp.17
## Comp.1 -1.818571e-12
## Comp.2 5.250882e-13
## Comp.3 2.945316e-13
## Comp.4 -1.182290e-13
## Comp.5 -1.903964e-13
## Comp.6 1.125271e-13

# The scores are  $A' * (X_i - \mu)$ . We center the data with scale()
# and then multiply each row by  $A'$ 
scores <- scale(laliga, center = TRUE, scale = FALSE) %*% A

# Same as (but this is much slower)
# scores <- t(apply(scale(laliga, center = TRUE, scale = FALSE), 1,
#                   function(x) t(A) %*% x))

# Same scores (up to possible changes in signs)
sum(abs(abs(pcaLaliga$scores) - abs(scores)))
## [1] 1.990578e-10

# Reconstruct the data from the principal components
sweep(pcaLaliga$scores %*% t(pcaLaliga$loadings), 2, pcaLaliga$center, "+")
##          Points Wins Draws Loses Goals.scored Goals.conceded
## Barcelona      91   29     4     5        112          29
## Real Madrid    90   28     6     4        110          34
## Atlético Madrid 88   28     4     6        63           18
## Villarreal     64   18    10    10        44           35
## Athletic        62   18     8    12        58           45
## Celta            60   17     9    12        51           59
## Sevilla          52   14    10    14        51           50
## Málaga           48   12    12    14        38           35
## Real Sociedad   48   13     9    16        45           48
## Betis            45   11    12    15        34           52
## Las Palmas       44   12     8    18        45           53
## Valencia         44   11    11    16        46           48
## Eibar            43   11    10    17        49           61

```

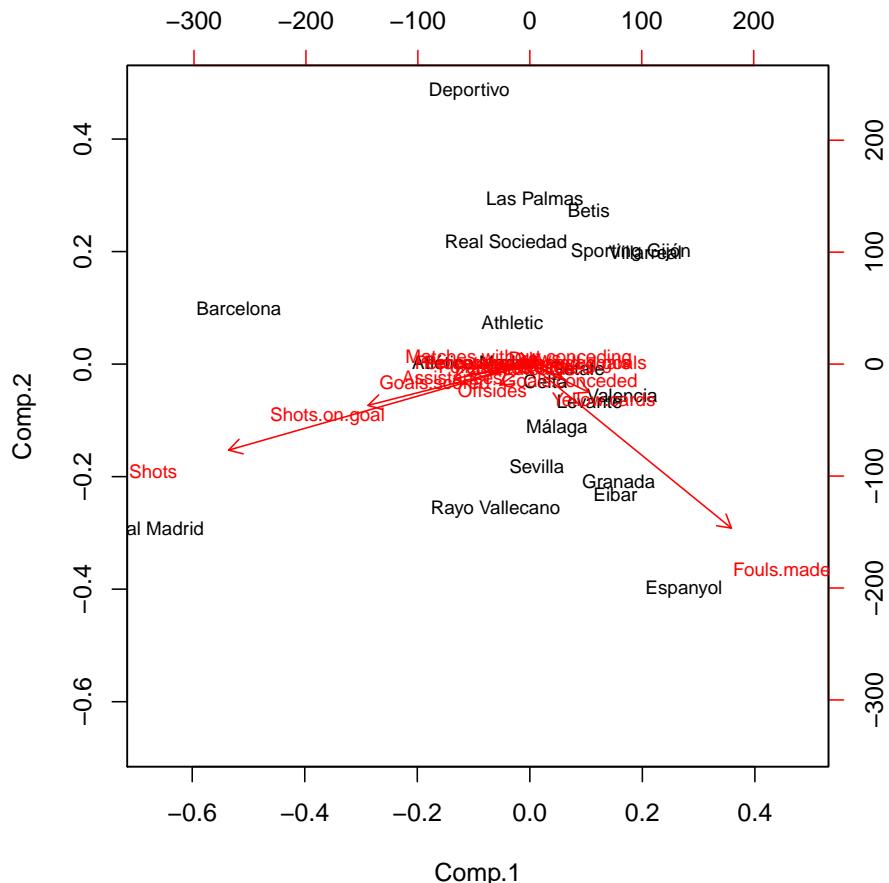
|                    |     |    |                           |                           |             |            |
|--------------------|-----|----|---------------------------|---------------------------|-------------|------------|
| ## Espanyol        | 43  | 12 | 7                         | 19                        | 40          | 74         |
| ## Deportivo       | 42  | 8  | 18                        | 12                        | 45          | 61         |
| ## Granada         | 39  | 10 | 9                         | 19                        | 46          | 69         |
| ## Sporting Gijón  | 39  | 10 | 9                         | 19                        | 40          | 62         |
| ## Rayo Vallecano  | 38  | 9  | 11                        | 18                        | 52          | 73         |
| ## Getafe          | 36  | 9  | 9                         | 20                        | 37          | 67         |
| ## Levante         | 32  | 8  | 8                         | 22                        | 37          | 70         |
| ##                 |     |    | Percentage.scored.goals   | Percentage.conceded.goals | Shots       |            |
| ## Barcelona       |     |    | 2.95                      |                           | 0.76        | 600        |
| ## Real Madrid     |     |    | 2.89                      |                           | 0.89        | 712        |
| ## Atlético Madrid |     |    | 1.66                      |                           | 0.47        | 481        |
| ## Villarreal      |     |    | 1.16                      |                           | 0.92        | 346        |
| ## Athletic        |     |    | 1.53                      |                           | 1.18        | 450        |
| ## Celta           |     |    | 1.34                      |                           | 1.55        | 442        |
| ## Sevilla         |     |    | 1.34                      |                           | 1.32        | 460        |
| ## Málaga          |     |    | 1.00                      |                           | 0.92        | 452        |
| ## Real Sociedad   |     |    | 1.18                      |                           | 1.26        | 454        |
| ## Betis           |     |    | 0.89                      |                           | 1.37        | 398        |
| ## Las Palmas      |     |    | 1.18                      |                           | 1.39        | 418        |
| ## Valencia        |     |    | 1.21                      |                           | 1.26        | 401        |
| ## Eibar           |     |    | 1.29                      |                           | 1.61        | 421        |
| ## Espanyol        |     |    | 1.05                      |                           | 1.95        | 388        |
| ## Deportivo       |     |    | 1.18                      |                           | 1.61        | 451        |
| ## Granada         |     |    | 1.21                      |                           | 1.82        | 430        |
| ## Sporting Gijón  |     |    | 1.05                      |                           | 1.63        | 372        |
| ## Rayo Vallecano  |     |    | 1.37                      |                           | 1.92        | 505        |
| ## Getafe          |     |    | 0.97                      |                           | 1.76        | 433        |
| ## Levante         |     |    | 0.97                      |                           | 1.84        | 434        |
| ##                 |     |    | Shots.on.goal             | Penalties.scored          | Assistances | Fouls.made |
| ## Barcelona       | 277 |    | 11                        | 79                        | 385         |            |
| ## Real Madrid     | 299 |    | 6                         | 90                        | 420         |            |
| ## Atlético Madrid | 186 |    | 1                         | 49                        | 503         |            |
| ## Villarreal      | 135 |    | 3                         | 32                        | 534         |            |
| ## Athletic        | 178 |    | 3                         | 42                        | 502         |            |
| ## Celta           | 170 |    | 4                         | 43                        | 528         |            |
| ## Sevilla         | 189 |    | 6                         | 35                        | 555         |            |
| ## Málaga          | 170 |    | 2                         | 27                        | 552         |            |
| ## Real Sociedad   | 164 |    | 1                         | 33                        | 465         |            |
| ## Betis           | 132 |    | 3                         | 26                        | 490         |            |
| ## Las Palmas      | 169 |    | 4                         | 33                        | 465         |            |
| ## Valencia        | 143 |    | 4                         | 27                        | 567         |            |
| ## Eibar           | 158 |    | 6                         | 33                        | 598         |            |
| ## Espanyol        | 155 |    | 1                         | 31                        | 654         |            |
| ## Deportivo       | 166 |    | 1                         | 32                        | 401         |            |
| ## Granada         | 137 |    | 7                         | 27                        | 586         |            |
| ## Sporting Gijón  | 129 |    | 1                         | 29                        | 522         |            |
| ## Rayo Vallecano  | 192 |    | 2                         | 35                        | 545         |            |
| ## Getafe          | 158 |    | 1                         | 31                        | 533         |            |
| ## Levante         | 154 |    | 2                         | 23                        | 548         |            |
| ##                 |     |    | Matches.without.conceding | Yellow.cards              | Red.cards   | Offsides   |
| ## Barcelona       |     |    | 18                        | 66                        | 1           | 120        |
| ## Real Madrid     |     |    | 14                        | 72                        | 5           | 114        |
| ## Atlético Madrid |     |    | 24                        | 91                        | 3           | 84         |

|                   |    |     |   |     |
|-------------------|----|-----|---|-----|
| ## Villarreal     | 17 | 100 | 4 | 106 |
| ## Athletic       | 13 | 84  | 5 | 92  |
| ## Celta          | 10 | 116 | 6 | 103 |
| ## Sevilla        | 11 | 106 | 7 | 106 |
| ## Málaga         | 12 | 110 | 5 | 85  |
| ## Real Sociedad  | 13 | 108 | 5 | 85  |
| ## Betis          | 12 | 110 | 3 | 80  |
| ## Las Palmas     | 11 | 99  | 4 | 79  |
| ## Valencia       | 7  | 113 | 7 | 92  |
| ## Eibar          | 7  | 109 | 4 | 85  |
| ## Espanyol       | 7  | 131 | 5 | 96  |
| ## Deportivo      | 7  | 84  | 4 | 85  |
| ## Granada        | 4  | 141 | 6 | 91  |
| ## Sporting Gijón | 7  | 107 | 5 | 81  |
| ## Rayo Vallecano | 6  | 115 | 9 | 123 |
| ## Getafe         | 8  | 130 | 6 | 72  |
| ## Levante        | 6  | 133 | 7 | 73  |

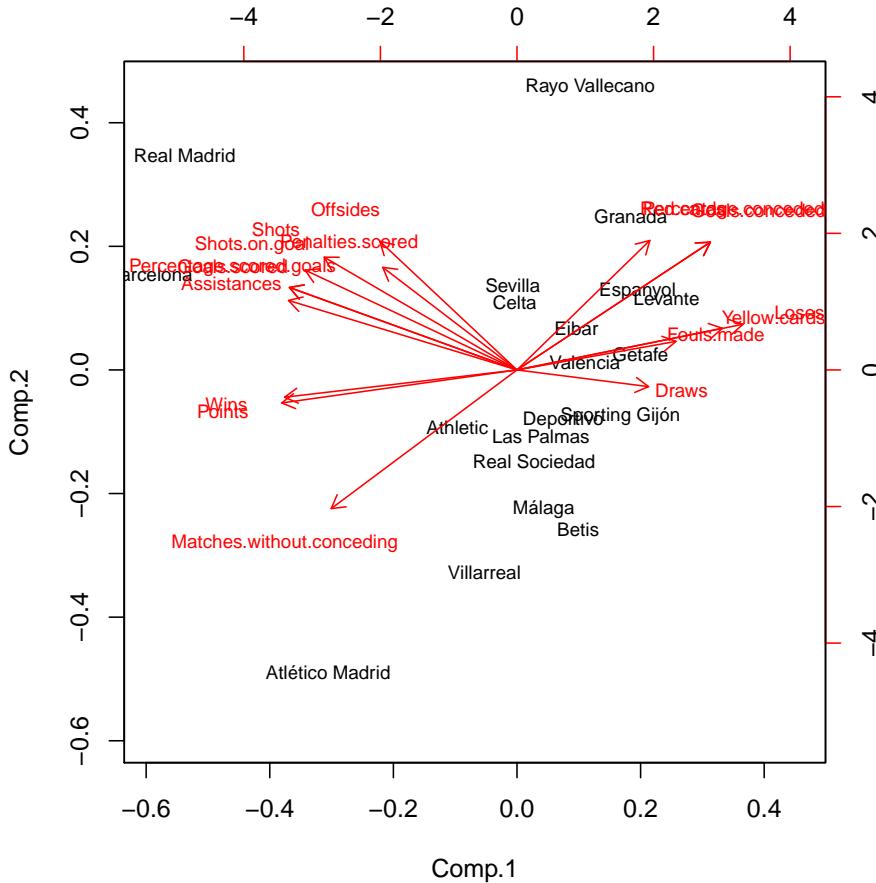
An important issue when doing PCA is the **scale** of the variables, since the variance depends on the units in which the variable is measured. Therefore, a sample of lengths measured in centimeters will have a variance  $10^4$  times larger than the same sample measured in meters – yet it is the same information! The same phenomenon may happen when variables with different ranges are mixed: the variability of one may dominate the other as an artifact of the scale. To prevent this, we **standardize the dataset prior to a PCA**.

```
# Use cor = TRUE to standardize variables (all have unit variance)
# and avoid scale distortions
pcaLaligaStd <- princomp(x = laliga, cor = TRUE)
summary(pcaLaligaStd)
## Importance of components:
##                               Comp.1     Comp.2     Comp.3     Comp.4
## Standard deviation      3.2918365 1.5511043 1.13992451 0.91454883
## Proportion of Variance 0.6374228 0.1415250 0.07643694 0.04919997
## Cumulative Proportion  0.6374228 0.7789478 0.85538472 0.90458469
##                               Comp.5     Comp.6     Comp.7     Comp.8
## Standard deviation      0.85765282 0.59351138 0.45780827 0.370649324
## Proportion of Variance 0.04326873 0.02072093 0.01232873 0.008081231
## Cumulative Proportion  0.94785342 0.96857434 0.98090308 0.988984306
##                               Comp.9     Comp.10    Comp.11    Comp.12
## Standard deviation      0.327182806 0.217470830 0.128381750 0.0976778705
## Proportion of Variance 0.006296976 0.002781974 0.000969522 0.0005612333
## Cumulative Proportion  0.995281282 0.998063256 0.999032778 0.9995940111
##                               Comp.13    Comp.14    Comp.15    Comp.16
## Standard deviation      0.083027923 2.582795e-03 1.226924e-03 2.222483e-08
## Proportion of Variance 0.000405508 3.924019e-07 8.854961e-08 2.905548e-17
## Cumulative Proportion  0.999999519 9.999999e-01 1.000000e+00 1.000000e+00
##                               Comp.17
## Standard deviation      0
## Proportion of Variance 0
## Cumulative Proportion 1

# The effects of the distortion can be clearly seen with the biplot
# Variability absorbed by Shots, Shots.on.goal, Fouls.made
biplot(pcaLaliga, cex = 0.75)
```



```
# The effects of the variables are more balanced
biplot(pcaLaLigaStd, cex = 0.75)
```



The biplot provides a powerful and succinct way of displaying the relevant information for up to two principal components. It shows:

1. The **scores of the data in  $PC_1$  and  $PC_2$**  by points (with optional text labels, depending if there are case names). This is the representation of the data in the first two PCs.
2. The **variables represented in the  $PC_1$  and  $PC_2$  by the arrows**. These arrows are centered at  $(0, 0)$ .

Let's examine the arrow associated to the variable  $X_j$ .  $X_j$  is expressed in terms of  $PC_1$  and  $PC_2$  by the weights  $a_{j1}$  and  $a_{j2}$ :

$$X_j = a_{j1}PC_1 + a_{j2}PC_2 + \dots + a_{jk}PC_p \approx a_{j1}PC_1 + a_{j2}PC_2.$$

$a_{j1}$  and  $a_{j2}$  have the same sign as  $\text{Cor}(X_j, PC_1)$  and  $\text{Cor}(X_j, PC_2)$ , respectively. The arrow associated to  $X_j$  is given by the segment joining  $(0, 0)$  and  $(a_{j1}, a_{j2})$ . Therefore:

- If the arrow *points right* ( $a_{j1} > 0$ ), there is *positive correlation between  $X_j$  and  $PC_1$* . Analogous if the arrow points left.
- If the arrow is *approximately vertical* ( $a_{j1} \approx 0$ ), there is *uncorrelation between  $X_j$  and  $PC_1$* .

Analogously:

- If the arrow *points up* ( $a_{j2} > 0$ ), there is *positive correlation between  $X_j$  and  $PC_2$* . Analogous if the arrow points down.
- If the arrow is *approximately horizontal* ( $a_{j2} \approx 0$ ), there is *uncorrelation between  $X_j$  and  $PC_2$* .

In addition, the **magnitude of the arrow informs us about the correlation**.

The biplot also provides the direct relation between variables, at sight of their expressions in  $PC_1$  and  $PC_2$ . The **angle** of the arrows of variable  $X_j$  and  $X_k$  gives an **approximation to the correlation between**

**them**,  $\text{Cor}(X_j, X_k)$ :

- If angle  $\approx 0^\circ$ , the two variables are highly positively correlated.
- If angle  $\approx 90^\circ$ , they are approximately uncorrelated.
- If angle  $\approx 180^\circ$ , the two variables are highly negatively correlated.

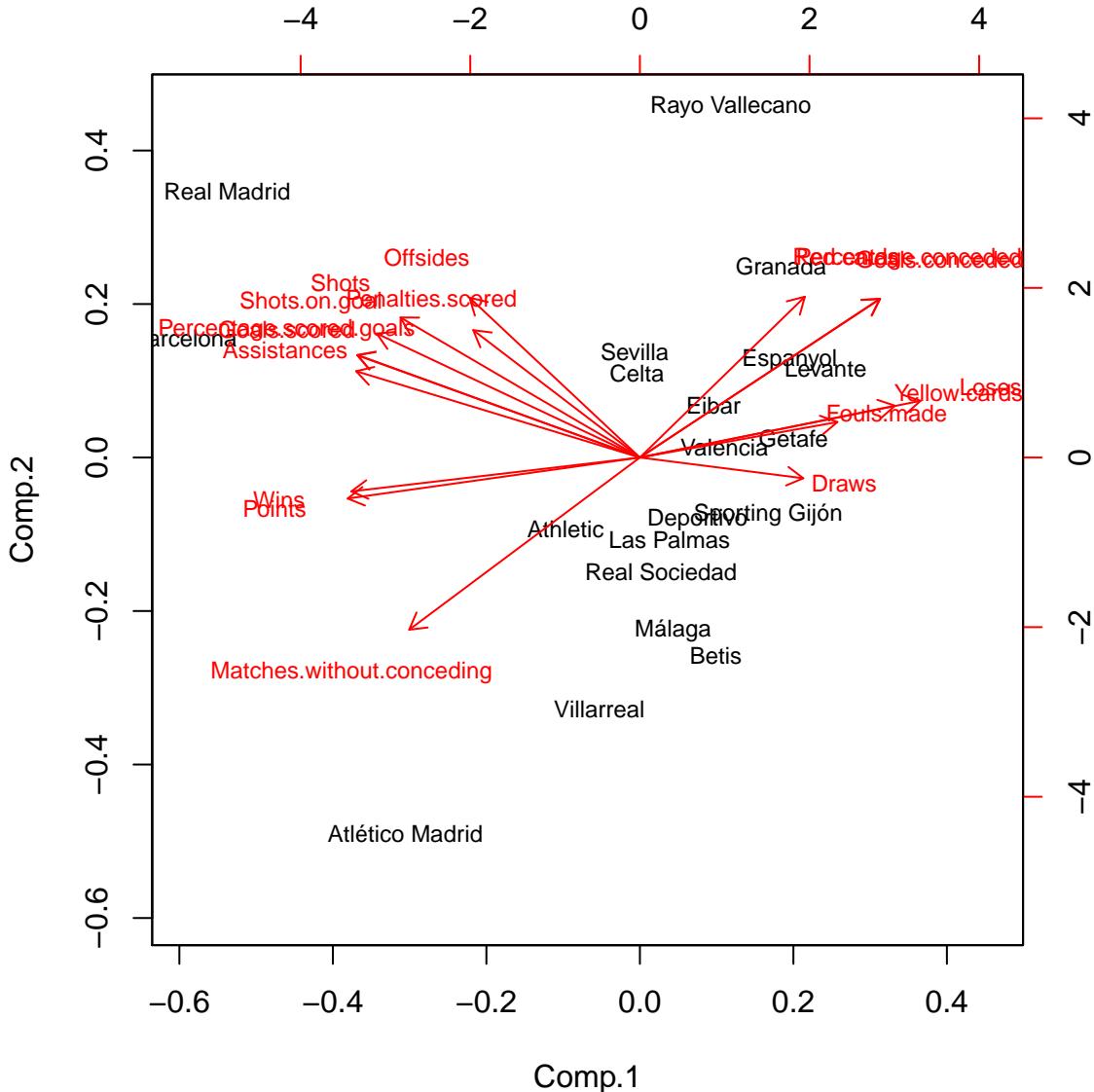


The insights obtained on the correlations between the variables and principal components are as *valid* as the percentage of variance explained by  $\text{PC}_1$  and  $\text{PC}_2$ .

Some interesting insights in La Liga 2015/2016 biplot are:

- $\text{PC}_1$  can be regarded as the *non-performance of a team* during the season. It is negatively correlated with **Wins**, **Points**,...and positively correlated with **Draws**, **Loses**, **Yellow.cards**,...The best performing teams are not surprising: Barcelona, Real Madrid, and Atlético Madrid. On the other hand, among the worst-performing teams are Levante, Getafe, and Granada.
- $\text{PC}_2$  can be seen as the *inefficiency of a team* (conceding points with little participation in the game). Using this interpretation we can see that Rayo Vallecano and Real Madrid were the most inefficient teams and Atlético Madrid and Villareal were the most efficient.
- **Offsides** is approximately uncorrelated with **Red.cards**.
- $\text{PC}_3$  does not have a clear interpretation.

```
biplot(pcaLaligaStd, cex = 0.75)
```



A 3D representation of the biplot can be computed through:

```
library(pca3d)
pca3d(pcaLaligaStd, show.labels = TRUE, biplot = TRUE)
```

### 3.6.2 Principal components regression

The key idea behind *Principal Components Regression* (PCR) is to regress the response  $Y$  in a set of principal components  $PC_1, \dots, PC_l$  obtained from the predictors  $X_1, \dots, X_p$ , where  $l < p$ . The motivation is that often a small number of principal components is enough to explain most of the variability of the predictors and consequently their relationship with  $Y$ <sup>6</sup>. Therefore, we look for fitting the linear model

$$Y = \alpha_0 + \alpha_1 PC_1 + \dots + \alpha_l PC_l + \varepsilon. \quad (3.6)$$

The main advantages of PCR are two:

<sup>6</sup>This does not need to be true, but it is often the case.

1. **Multicollinearity is avoided** by design:  $\text{PC}_1, \dots, \text{PC}_l$  are uncorrelated between them.
2. **The number of coefficients to estimate,  $l$ , is smaller**, hence the accuracy of the estimation increases.

However, keep in mind that PCR affects the linear model in two fundamental ways:

1. *Interpretation of the coefficients is not immediately related with the predictors*, but with the principal components. Hence, the interpretability of a given coefficient is inherited from the interpretability of the associated principal component.
2. *Prediction needs an extra step*, since it is required to obtain the scores of the new observations of the predictors in the principal components.

The first point is worth discussing now. The PCR model (3.6) can be seen as a linear model expressed in terms of the original predictors. To make this point clearer, let re-express (3.6) as

$$Y = \alpha_0 + \mathbf{PC}'_{1:l} \boldsymbol{\alpha}_{1:l} + \varepsilon, \quad (3.7)$$

where the subindex  $1 : l$  denotes the inclusion of the vector entries from 1 to  $l$ . Then, we can express the PCR problem (3.7) in terms of the original predictors, since we know that  $\text{PC}_j = \mathbf{a}'_j(\mathbf{X} - \boldsymbol{\mu})$  (check (3.2) for the centered case):

$$\begin{aligned} Y &= \alpha_0 + (\mathbf{a}'_1(\mathbf{X} - \boldsymbol{\mu}), \dots, \mathbf{a}'_l(\mathbf{X} - \boldsymbol{\mu}))' \boldsymbol{\alpha}_{1:l} + \varepsilon \\ &= \alpha_0 + (\mathbf{A}'_{1:l}(\mathbf{X} - \boldsymbol{\mu}))' \boldsymbol{\alpha}_{1:l} + \varepsilon \\ &= (\alpha_0 - \boldsymbol{\mu}' \mathbf{A}_{1:l} \boldsymbol{\alpha}_{1:l}) + \mathbf{X}' \mathbf{A}_{1:l} \boldsymbol{\alpha}_{1:l} + \varepsilon, \end{aligned} \quad (3.8)$$

where  $\mathbf{A}_{1:l}$  represents the  $\mathbf{A}$  matrix with only its first  $l$  columns. In other words, the  $\boldsymbol{\alpha}_{1:l}$  of the coefficients of the PCR done with  $l$  principal components in (3.7) yield the  $\boldsymbol{\beta}$  coefficients associated to the linear model  $Y = \gamma_0 + \gamma_1 X_1 + \dots + \gamma_p X_p + \varepsilon$ :

$$\gamma_0 = \alpha_0 - \boldsymbol{\mu}' \mathbf{A}_{1:l} \boldsymbol{\alpha}_{1:l}, \quad \gamma_{1:p} = \mathbf{A}_{1:l} \boldsymbol{\alpha}_{1:l}.$$

In the sample case, we have that

$$\hat{\gamma}_0 = \hat{\alpha}_0 - \bar{\mathbf{X}}' \hat{\mathbf{A}}_{1:l} \hat{\boldsymbol{\alpha}}_{1:l}, \quad \hat{\gamma}_{1:p} = \hat{\mathbf{A}}_{1:l} \hat{\boldsymbol{\alpha}}_{1:l}. \quad (3.9)$$

Notice that  $\hat{\gamma}$  is **not the least squares estimator** we have denoted by  $\hat{\boldsymbol{\beta}}$ , but just the coefficient of the PCR associated to the original predictors.  $\hat{\gamma}$  is useful for interpretation of the linear model produced by PCR.

Finally, remember that the **usefulness** of PCR relies on how well we are able to reduce the dimensionality of the predictors (if  $l = p$ , then PCR is equivalent to the least squares estimation) and the veracity of the assumption that the  $l$  principal components are related with  $Y$ .



Keep in mind that PCR considers the **PCA done in the set of predictors**, this is, we exclude the response for obvious reasons (a perfect and useless fit). It is important to remove the response from the call to `princomp` if we want to use the output in `lm`.

We see now two approaches for performing PCR, which we illustrate with the `laliga` dataset. The objective is to predict `Points` using the remaining variables (except from the directly related variables `Wins`, `Draws`, and `Loses`) in order to quantify, explain, and predict the final points of a team from its performance.

The first approach combines the use of the `princomp` and `lm` functions. Its strong points are that is both able to predict and explain, and is linked with techniques we have employed so far. The weak point is that it requires some extra coding.

```
# A linear model is problematic
mod <- lm(Points ~ . - Wins - Draws - Loses - Matches.without.conceding,
          data = laliga)
summary(mod) # Lots of non significant predictors
##
## Call:
## lm(formula = Points ~ . - Wins - Draws - Loses - Matches.without.conceding,
##      data = laliga)
##
## Residuals:
##    Min     1Q   Median     3Q    Max
## -5.2117 -1.4766  0.0544  1.9515  4.1422
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|) 
## (Intercept)                77.11798  26.12915  2.951  0.0214 * 
## Goals.scored             -28.21714  17.44577 -1.617  0.1498  
## Goals.conceded            -24.23628  15.45595 -1.568  0.1608  
## Percentage.scored.goals 1066.98731  655.69726  1.627  0.1477  
## Percentage.conceded.goals 896.94781  584.97833  1.533  0.1691  
## Shots                   -0.10246  0.07754 -1.321  0.2279  
## Shots.on.goal            0.02024  0.13656  0.148  0.8863  
## Penalties.scored         -0.81018  0.77600 -1.044  0.3312  
## Assists                 1.41971  0.44103  3.219  0.0147 * 
## Fouls.made              -0.04438  0.04267 -1.040  0.3328  
## Yellow.cards             0.27850  0.16814  1.656  0.1416  
## Red.cards                0.68663  1.44229  0.476  0.6485  
## Offsides                -0.00486  0.14854 -0.033  0.9748 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.274 on 7 degrees of freedom
## Multiple R-squared:  0.9795, Adjusted R-squared:  0.9443 
## F-statistic: 27.83 on 12 and 7 DF,  p-value: 9.784e-05

# We try to clean the model
modBIC <- stepAIC(mod, k = log(nrow(laliga)), trace = 0)
summary(modBIC) # Better, but...
##
## Call:
## lm(formula = Points ~ Goals.scored + Goals.conceded + Percentage.scored.goals +
##      Percentage.conceded.goals + Shots + Assists + Yellow.cards,
##      data = laliga)
##
## Residuals:
##    Min     1Q   Median     3Q    Max
## -5.4830 -1.4505  0.9008  1.1813  5.8662
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
```

```

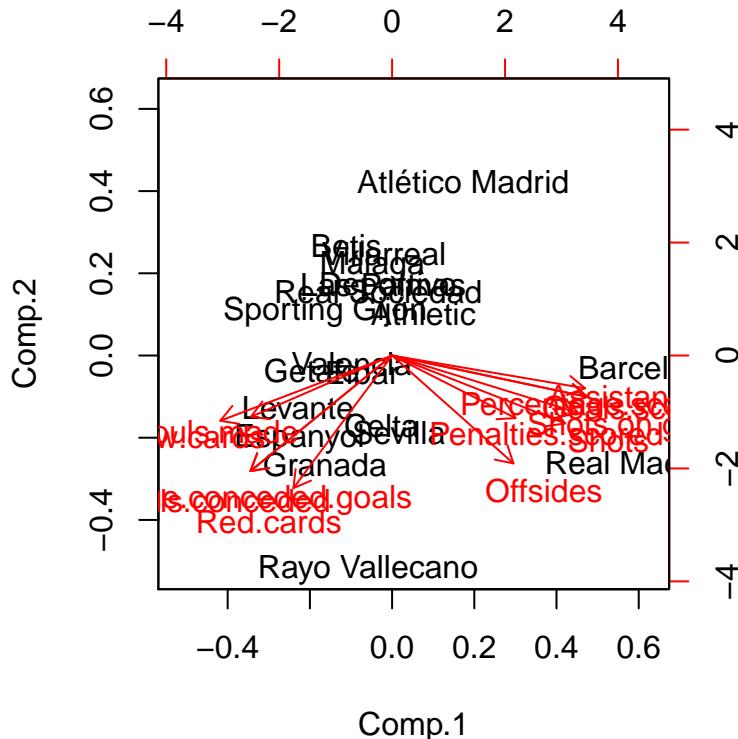
## (Intercept) 62.91373 10.73528 5.860 7.71e-05 ***
## Goals.scored -23.90903 11.22573 -2.130 0.05457 .
## Goals.conceded -12.16610 8.11352 -1.499 0.15959
## Percentage.scored.goals 894.56861 421.45891 2.123 0.05528 .
## Percentage.conceded.goals 440.76333 307.38671 1.434 0.17714
## Shots -0.05752 0.02713 -2.120 0.05549 .
## Assists 1.42267 0.28462 4.999 0.00031 ***
## Yellow.cards 0.11313 0.07868 1.438 0.17603
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.743 on 12 degrees of freedom
## Multiple R-squared: 0.973, Adjusted R-squared: 0.9572
## F-statistic: 61.77 on 7 and 12 DF, p-value: 1.823e-08

# Huge multicollinearity
vif(modBIC)
##          Goals.scored          Goals.conceded
## 77998.044758 22299.952547
##  Percentage.scored.goals Percentage.conceded.goals
## 76320.612577 22322.307151
##          Shots          Assists
## 6.505748 32.505831
##          Yellow.cards
## 3.297224

# PCA without Points, Wins, Draws, Loses, and Matches.without.conceding
# A quick way of removing columns without knowing its position
laligaRed <- subset(laliga, select = -c(Points, Wins, Draws, Loses,
                                         Matches.without.conceding))
pcaLaligaRed <- princomp(x = laligaRed, cor = TRUE)
summary(pcaLaligaRed) # l = 3 gives 86% of variance explained
## Importance of components:
##          Comp.1     Comp.2     Comp.3     Comp.4     Comp.5
## Standard deviation 2.7437329 1.4026745 0.91510249 0.8577839 0.65747209
## Proportion of Variance 0.6273392 0.1639580 0.06978438 0.0613161 0.03602246
## Cumulative Proportion 0.6273392 0.7912972 0.86108155 0.9223977 0.95842012
##          Comp.6     Comp.7     Comp.8     Comp.9
## Standard deviation 0.5310954 0.332556029 0.263170555 0.146091551
## Proportion of Variance 0.0235052 0.009216126 0.005771562 0.001778562
## Cumulative Proportion 0.9819253 0.991141438 0.996913000 0.998691562
##          Comp.10    Comp.11    Comp.12
## Standard deviation 0.125252621 3.130311e-03 1.801036e-03
## Proportion of Variance 0.001307352 8.165704e-07 2.703107e-07
## Cumulative Proportion 0.999998913 9.999997e-01 1.000000e+00

# Interpretation of PC1 and PC2
biplot(pcaLaligaRed)

```



```

# PC1: attack performance of the team

# Create a new dataset with the response + principal components
laligaPCA <- data.frame("Points" = laliga$Points, pcaLaligaRed$scores)

# Regression on all principal components
modPCA <- lm(Points ~ ., data = laligaPCA)
summary(modPCA) # Predictors clearly significative - same R^2 as mod
##
## Call:
## lm(formula = Points ~ ., data = laligaPCA)
##
## Residuals:
##      Min      1Q      Median      3Q      Max
## -5.2117 -1.4766  0.0544  1.9515  4.1422
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 52.4000    0.9557  54.831 1.76e-10 ***
## Comp.1      5.7690    0.3483  16.563 7.14e-07 ***
## Comp.2      2.4376    0.6813   3.578   0.0090 **  
## Comp.3      3.4222    1.0443   3.277   0.0135 *   
## Comp.4      3.6079    1.1141   3.238   0.0143 *   
## Comp.5     -1.9713    1.4535  -1.356   0.2172    
## Comp.6     -5.7067    1.7994  -3.171   0.0157 *   
## Comp.7     -3.4169    2.8737  -1.189   0.2732    
## Comp.8      9.0212    3.6313   2.484   0.0419 *   
## Comp.9     -4.6455    6.5415  -0.710   0.5006    
## Comp.10    -10.2087   7.6299  -1.338   0.2227    
## Comp.11    -222.0340   305.2920  -0.727   0.4907  
## 
```

```

## Comp.12      -954.7650  530.6164  -1.799   0.1150
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.274 on 7 degrees of freedom
## Multiple R-squared:  0.9795, Adjusted R-squared:  0.9443
## F-statistic: 27.83 on 12 and 7 DF,  p-value: 9.784e-05
vif(modPCA) # No problems at all
##   Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8  Comp.9
##       1       1       1       1       1       1       1       1       1
##   Comp.10 Comp.11 Comp.12
##      1       1       1

# Using the first three components
modPCA3 <- lm(Points ~ Comp.1 + Comp.2 + Comp.3, data = laligaPCA)
summary(modPCA3)
##
## Call:
## lm(formula = Points ~ Comp.1 + Comp.2 + Comp.3, data = laligaPCA)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.178 -4.541 -1.401  3.501 16.093
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 52.4000    1.5672  33.435 3.11e-16 ***
## Comp.1      5.7690    0.5712  10.100 2.39e-08 ***
## Comp.2      2.4376    1.1173   2.182   0.0444 *
## Comp.3      3.4222    1.7126   1.998   0.0630 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.009 on 16 degrees of freedom
## Multiple R-squared:  0.8738, Adjusted R-squared:  0.8501
## F-statistic: 36.92 on 3 and 16 DF,  p-value: 2.027e-07

# Coefficients associated to each original predictor (gamma)
alpha <- modPCA3$coefficients
gamma <- pcaLaligaRed$loadings[, 1:3] %*% alpha[-1] # Slopes
gamma <- c(alpha[1] - pcaLaligaRed$center %*% gamma, gamma) # Intercept
gamma
## [1] -44.2288551  1.7305124 -3.4048178  1.7416378 -3.3944235
## [6]  0.2347716   0.8782162  2.6044699  1.4548813 -0.1171732
## [11] -1.7826488 -2.6423211  1.3755697

# We can overpenalize to have a simpler model - also one single
# principal component does quite well
modPCABC <- stepAIC(modPCA, k = 2 * log(nrow(laliga)), trace = 0)
summary(modPCABC)
##
## Call:
## lm(formula = Points ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.6 +

```

```

##      Comp.8, data = laligaPCA)
##
## Residuals:
##      Min      1Q  Median      3Q     Max
## -6.6972 -2.6418 -0.3265  2.3535  8.4944
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 52.4000    1.0706  48.946 3.95e-16 ***
## Comp.1      5.7690    0.3902  14.785 1.65e-09 ***
## Comp.2      2.4376    0.7632   3.194  0.00705 **  
## Comp.3      3.4222    1.1699   2.925  0.01182 *   
## Comp.4      3.6079    1.2481   2.891  0.01263 *   
## Comp.6     -5.7067    2.0158  -2.831  0.01416 *   
## Comp.8      9.0212    4.0680   2.218  0.04502 *  
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.788 on 13 degrees of freedom
## Multiple R-squared:  0.9521, Adjusted R-squared:  0.9301
## F-statistic: 43.11 on 6 and 13 DF,  p-value: 7.696e-08
# Note the order of the principal components does not correspond
# exactly to its importance in the regression

# To perform prediction we need to compute first the scores associated to the
# new values of the predictors, conveniently preprocessed
# Predictions for FCB and RMA (although they are part of the training sample)
newPredictors <- laligaRed[1:2, ]
newPredictors <- scale(newPredictors, center = pcaLaligaRed$center,
                      scale = pcaLaligaRed$scale) # Centred and scaled
newScores <- t(apply(newPredictors, 1,
                     function(x) t(pcaLaligaRed$loadings) %*% x))

# We need a data frame for prediction
newScores <- data.frame("Comp" = newScores)
predict(modPCABC, newdata = newScores, interval = "prediction")
##             fit      lwr      upr
## Barcelona  93.64950 80.35115 106.9478
## Real Madrid 90.05622 77.11876 102.9937

# Reality
laliga[1:2, 1]
## [1] 91 90

```

The second approach employs the function `pqr` from the `pls` library and is more direct, yet less connected with the techniques we have seen so far. It employs a model object different from the `lm` object and, as a consequence, functions like `summary`, `BIC`, `stepAIC`, or `plot` do not work properly. This implies that inference, model selection, and model validation are not so simple. In exchange, `pqr` allows for model fitting in an easier way and model selection through the use of cross-validation. In summary, this is a more *predictive* approach than *predictive and explicative*.

```

# Simple call to pcr
library(pls)

modPcr <- pcr(Points ~ ., data = laligaRed2, scale = TRUE)
# Notice we do not need to create a data.frame with PCA, it is automatically
# done within pcr. We also have flexibility to remove predictors from the PCA
# scale = TRUE means that the variables are scaled prior to compute PCA

# The summary of the model is different
summary(modPcr)
## Data: X dimension: 20 12
## Y dimension: 20 1
## Fit method: svdpc
## Number of components considered: 12
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       62.73    79.13    86.11    92.24    95.84    98.19    99.11
## Points  80.47    84.23    87.38    90.45    90.99    93.94    94.36
##          8 comps  9 comps  10 comps 11 comps 12 comps
## X       99.69    99.87   100.00    100     100.00
## Points  96.17    96.32    96.84    97     97.95
# First row: percentage of variance explained of the predictors
# Second row: percentage of variance explained of Y (the R^2)
# Note we have the same R^2 for 3 and 12 components as in the previous approach

# Slots of information in the model - most of them as 3-dim arrays with the
# third dimension indexing the number of components considered
names(modPcr)
## [1] "coefficients"   "scores"          "loadings"        "Yloadings"
## [5] "projection"     "Xmeans"          "Ymeans"          "fitted.values"
## [9] "residuals"       "Xvar"            "Xtotvar"        "fit.time"
## [13] "ncomp"          "method"          "scale"          "call"
## [17] "terms"          "model"

# The coefficients of the original predictors, not of the components!
modPcr$coefficients[, , 12]
##          Goals.scored          Goals.conceded
##          -602.85050765          -383.07010184
## Percentage.scored.goals Percentage.conceded.goals
##          600.61255371          374.38729000
##          Shots          Shots.on.goal
##          -8.27239221          0.88174787
## Penalties.scored          Assists
##          -2.14313238          24.42240486
##          Fouls.made          Yellow.cards
##          -2.96044265          5.51983512
##          Red.cards          Offsides
##          1.20945331          -0.07231723
# pcr() computes up to ncomp (in this case, 12) linear models, each one
# considering one extra principal component. $coefficients returns in a
# 3-dim array the coefficients of all the linear models

# Prediction is simpler and can be done for different number of components
predict(modPcr, newdata = laligaRed2[1:2, ], ncomp = 12)

```

```

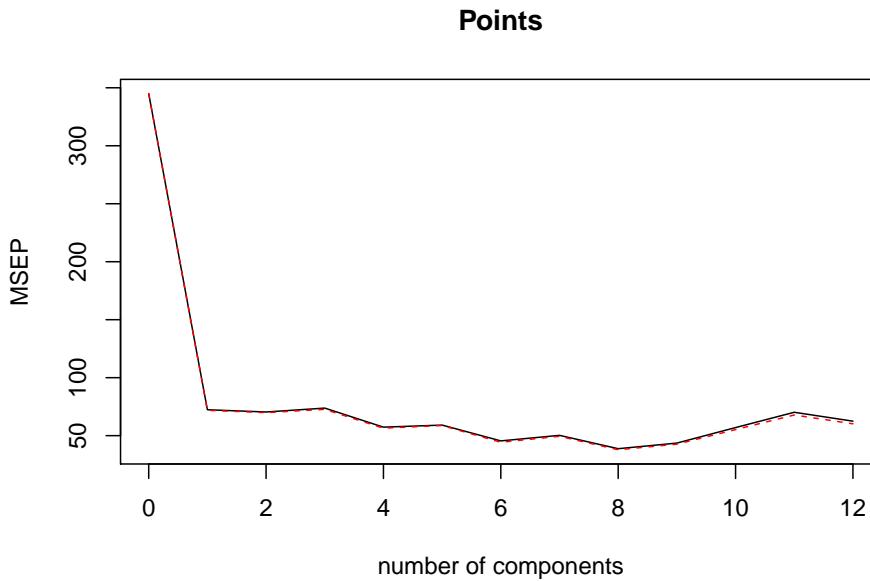
## , , 12 comps
##
## Points
## Barcelona 92.01244
## Real Madrid 91.38026

# Selecting the number of components to retain. All the components up to ncomp
# are selected, no further flexibility is possible
modPcr2 <- pcr(Points ~ ., data = laligaRed2, scale = TRUE, ncomp = 3)
summary(modPcr2)
## Data: X dimension: 20 12
## Y dimension: 20 1
## Fit method: svdpc
## Number of components considered: 3
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps
## X       62.73    79.13    86.11
## Points  80.47    84.23    87.38

# Selecting the number of components to retain by Leave-One-Out
# cross-validation
modPcrCV1 <- pcr(Points ~ ., data = laligaRed2, scale = TRUE,
                   validation = "LOO")
summary(modPcrCV1)
## Data: X dimension: 20 12
## Y dimension: 20 1
## Fit method: svdpc
## Number of components considered: 12
##
## VALIDATION: RMSEP
## Cross-validated using 20 leave-one-out segments.
##          (Intercept) 1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV        18.57     8.505   8.390    8.588   7.571    7.688   6.743
## adjCV    18.57     8.476   8.356    8.525   7.513    7.663   6.655
##          7 comps  8 comps  9 comps  10 comps 11 comps 12 comps
## CV        7.09     6.224   6.603    7.547   8.375    7.905
## adjCV    7.03     6.152   6.531    7.430   8.236    7.760
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       62.73    79.13    86.11    92.24    95.84    98.19    99.11
## Points  80.47    84.23    87.38    90.45    90.99    93.94    94.36
##          8 comps  9 comps  10 comps 11 comps 12 comps
## X       99.69    99.87    100.00   100.00   100.00
## Points  96.17    96.32    96.84    97.00    97.95

# View cross-validation Mean Squared Error in Prediction
validationplot(modPcrCV1, val.type = "MSEP") # l = 8 gives the minimum CV

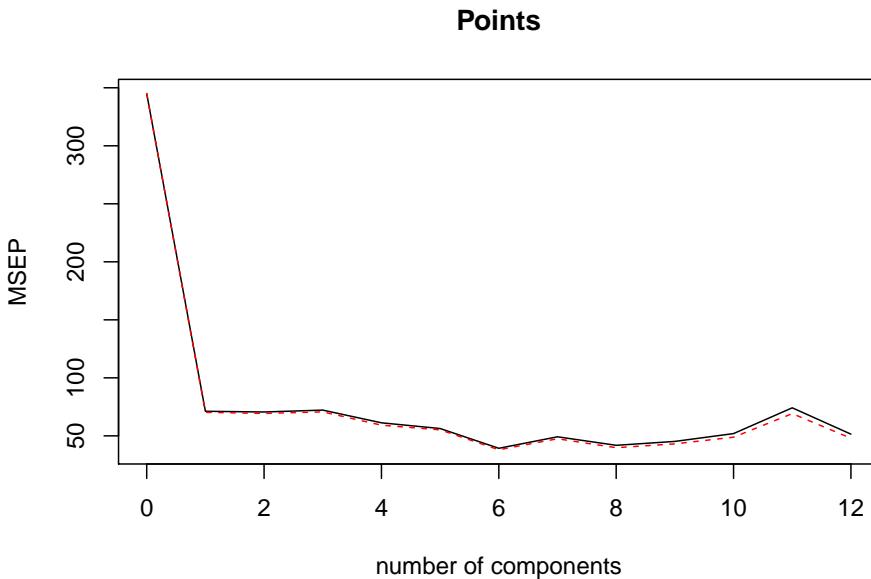
```



```

# Selecting the number of components to retain by 10-fold Cross-Validation
# (k = 10 is the default)
modPcrCV10 <- pcr(Points ~ ., data = laligaRed2, scale = TRUE,
                     validation = "CV")
summary(modPcrCV10)
## Data:    X dimension: 20 12
##  Y dimension: 20 1
## Fit method: svdpc
## Number of components considered: 12
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##          (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
##  CV          18.57    8.437   8.400   8.497   7.825   7.505   6.267
##  adjCV       18.57    8.382   8.325   8.407   7.694   7.428   6.167
##          7 comps 8 comps 9 comps 10 comps 11 comps 12 comps
##  CV          7.013    6.469   6.724   7.208   8.612   7.174
##  adjCV       6.892    6.308   6.566   6.988   8.316   6.923
##
## TRAINING: % variance explained
##          1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps
##  X          62.73    79.13   86.11   92.24   95.84   98.19   99.11
##  Points     80.47    84.23   87.38   90.45   90.99   93.94   94.36
##          8 comps 9 comps 10 comps 11 comps 12 comps
##  X          99.69    99.87   100.00   100.00   100.00
##  Points     96.17    96.32   96.84    97.00   97.95
validationplot(modPcrCV10, val.type = "MSEP") # l = 6 gives the minimum CV

```



`pqr()` does an internal scaling of the predictors by their *quasi-standard deviations*. This means that each variable is divided by  $\frac{1}{\sqrt{n-1}}$ , when in `princomp` a scaling of  $\frac{1}{\sqrt{n}}$  is applied (the *standard deviations* are employed). This results in a minor discrepancy in the `scores` object of both methods that is easily patchable. The `scores` of `princomp()` are the ones of `pqr()` **multiplied** by  $\sqrt{\frac{n}{n-1}}$ . This problem is inherited to the coefficients, which assume scores divided by  $\frac{1}{\sqrt{n-1}}$ . Therefore, the  $\hat{\gamma}$  coefficients described in (3.9) are obtained by **dividing** the coefficients of `pqr()` by  $\sqrt{\frac{n}{n-1}}$ .

The next chunk of code illustrates the previous warning.

```
# Equality of loadings from princomp() and pqr()
sum(abs(abs(pcaLaligaRed$loadings[, 1:3]) - abs(modPqr$loadings[, 1:3])))
## [1] 1.186204e-14

# Equality of scores from princomp() and pqr() (with the same standardization)
sum(abs(abs(pcaLaligaRed$scores[, 1:3]) -
       abs(modPqr$scores[, 1:3] * sqrt(n / (n - 1)))))
## [1] 8.735547e-14

# Equality of the gamma coefficients obtained previously for 3 PCA
# (with the same standardization)
modPqr$coefficients[, , 3] / sqrt(n / (n - 1))
##          Goals.scored          Goals.conceded
##           1.7305124          -3.4048178
## Percentage.scored.goals Percentage.conceded.goals
##           1.7416378          -3.3944235
##          Shots          Shots.on.goal
##           0.2347716          0.8782162
## Penalties.scored          Assists
##           2.6044699          1.4548813
##          Foul.made          Yellow.cards
##           -0.1171732          -1.7826488
##          Red.cards          Offsides
##           -2.6423211          1.3755697
```

```

gamma[-1]
## [1] 1.7305124 -3.4048178 1.7416378 -3.3944235 0.2347716 0.8782162
## [7] 2.6044699 1.4548813 -0.1171732 -1.7826488 -2.6423211 1.3755697

# Coefficients associated to the principal components - same as in modPCA3
lm(Points ~ ., data = data.frame("Points" = laliga$Points,
                                   modPcr$scores[, 1:3] * sqrt(n / (n - 1))))
## 
## Call:
## lm(formula = Points ~ ., data = data.frame(Points = laliga$Points,
##     modPcr$scores[, 1:3] * sqrt(n / (n - 1))))
## 
## Coefficients:
## (Intercept) Comp.1 Comp.2 Comp.3
## 52.400      -5.769   2.438  -3.422
modPCA3
## 
## Call:
## lm(formula = Points ~ Comp.1 + Comp.2 + Comp.3, data = laligaPCA)
## 
## Coefficients:
## (Intercept) Comp.1 Comp.2 Comp.3
## 52.400      5.769   2.438   3.422
# Of course, flipping of signs is always possible with PCA

```

The selection of  $l$  by **cross-validation** attempts to minimize the *Mean Squared Error in Prediction* (MSEP) or, equivalently, the *Root MSEP* (RMSEP), of the model. (Alternatively, the “*in Prediction*” part of the latter terms is dropped and they are just referred as the MSE and RMSE.) This is a **jackknife method** valid for the selection of any tuning parameter  $\lambda$  that affects the form of the estimate  $\hat{m}_\lambda$  of the regression function  $m$  (remember (1.1)). Given the sample  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$ , *leave-one-out cross-validation* considers the tuning parameter

$$\hat{\lambda}_{\text{CV}} := \arg \min_{\lambda} \sum_{i=1}^n (Y_i - \hat{m}_{\lambda, -i}(\mathbf{X}_i))^2,$$

where  $\hat{m}_{\lambda, -i}$  represents the fit of the model  $\hat{m}_\lambda$  without the  $i$ -th observation  $(\mathbf{X}_i, Y_i)$ .

A less computationally expensive variation on leave-one-out cross-validation is *k-fold cross-validation*, which partitions the data into  $k$  folds  $F_1, \dots, F_k$  of approximately equal size, trains the model  $\hat{m}_\lambda$  in the aggregation of  $k - 1$  folds and evaluates its MSEP in the remaining fold:

$$\hat{\lambda}_{k-\text{CV}} := \arg \min_{\lambda} \sum_{j=1}^k \sum_{i \in F_j} (Y_i - \hat{m}_{\lambda, -F_j}(\mathbf{X}_i))^2,$$

where  $\hat{m}_{\lambda, -F_j}$  represents the fit of the model  $\hat{m}_\lambda$  excluding the data from the  $j$ -th fold  $F_j$ . Recall that leave-one-out cross-validation is a particular case of *k-fold cross-validation* with  $k = n$ .



*k*-fold cross validation with  $k < n$  has an inconvenience to be aware of: it depends on the choice of the folds for splitting the data (*i.e.*, how each datum is assigned to each of the  $k$  folds). Some statistical softwares do this assignment **randomly**, which means that the selection of  $\hat{\lambda}_{k-\text{CV}}$  may vary from one run to another. Thus, fixing the seed prior to the parameter selection is important to ensure reproducibility. Another alternative is to aggregate the results of several cross-validations in a convenient way.

Notice that this problem is not present in leave-one-out cross-validation (where  $k = n$ ).



Inference in PCR can be carried out in the same way as it was done for the standard linear model. The key point is to realize that the inferences are about the coefficients  $\alpha_{0:l}$  associated to the  $l$  principal components, and that it can be carried out by the `summary()` function on the output of `lm()`. These are the ones that are estimated by least squares when considering the scores of the  $l$  principal components as our data. The inference is **not about**  $\gamma$ , which on the other hand is directly interpretable in terms of the predictors.



Inference in PCR is based on the assumptions of the linear model being satisfied for the principal components we are considering. The evaluation of the assumptions can be done using the exact same tools described in Section 3.5, but considering the principal components instead of the predictors.

**PCA is a linear transformation of the data.** Therefore:

- If the **linearity** assumption fails for the predictors  $X_1, \dots, X_p$ , then it will likely fail for  $PC_1, \dots, PC_l$ , since the transformation will not introduce nonlinearities able to capture the nonlinear effects.
- Similarly, if the **homoscedasticity, normality, or independence** assumptions fail for  $X_1, \dots, X_p$ , then they will likely fail for  $PC_1, \dots, PC_l$ .

Exceptions to the previous common implications are possible, and may involve the association of one or several *problematic* predictors (e.g., have nonlinear effects on the response) to the principal components that are excluded from the model. Up to which extent the failure of the assumptions in the original predictors can be mitigated by PCR depends on each application.

### 3.6.3 Partial least squares

PCR works by replacing the predictors  $X_1, \dots, X_p$  by a set of principal components  $PC_1, \dots, PC_l$  under the hope that these directions, that explain most of the variability of the predictors, are also the best directions for predicting the response  $Y$ . However, this is not guaranteed, and the idea behind *Partial Least Squares* (PLS) is to regress the response  $Y$  in a set of new variables  $PLS_1, \dots, PLS_l$  that are constructed with the objective of *best predicting the response  $Y$  from the predictors  $X_1, \dots, X_p$* .

As with PCA, the idea to find linear combinations of the predictors  $X_1, \dots, X_p$ , this is to have:

$$PLS_1 := \sum_{j=1}^p a_{j1} X_j, \dots, PLS_p := \sum_{j=1}^p a_{jp} X_j.$$

The question is how to choose the coefficients  $a_{jk}$ ,  $j, k = 1, \dots, p$ . PLS does it by *placing the most weight in the predictors that are most strongly correlated with  $Y$* . After standardizing the variables, for  $PLS_1$  this is achieved by setting  $a_{j1}$  equal to the theoretical slope coefficient of regressing  $Y$  into  $X_j$ , that is (recall (2.3) for the sample version):

$$a_{j1} := \beta_{j1} := \frac{\text{Cov}[X_j, Y]}{\text{Var}[X_j]},$$

where  $\beta_{j1}$  stems from  $Y = \beta_0 + \beta_{j1} X_j + \varepsilon$ . The second partial least squares direction,  $PLS_2$ , is computed in a similar way, but once the linear effects of the  $PLS_1$  are removed. This is achieved by:

1. Regressing each of  $X_1, \dots, X_p$  on  $PLS_1$ . That is, fit the  $p$  simple linear models

$$X_j = \beta_0 + \beta_{j1} PLS_1 + \varepsilon_j, \quad j = 1, \dots, p.$$

2. Regress  $Y$  in each of the  $p$  random errors  $\varepsilon_1, \dots, \varepsilon_p$  (they play the role of  $X_1, \dots, X_p$  in the computation of  $\text{PLS}_1$ ) from the above regressions, yielding

$$a_{j2} := \beta_{j2} := \frac{\text{Cov}[\varepsilon_j, Y]}{\text{Var}[\varepsilon_j]},$$

where  $\beta_{j2}$  stems from  $Y = \beta_0 + \beta_{j2}\varepsilon_j + \varepsilon$ . If the process is iterated, the coefficients for  $\text{PLS}_j$ ,  $j > 2$ , can be computed<sup>7</sup>. Once  $\text{PLS}_1, \dots, \text{PLS}_l$  are obtained, then PLS proceeds as PCR and fits the model

$$Y = \beta_0 + \beta_1 \text{PLS}_1 + \dots + \beta_l \text{PLS}_l + \varepsilon.$$

The implementation of PLS can be done by the function `plsr` from the `pls` package. It has an analogous syntax to `pcr`.

```
# Simple call to plsr - very similar to pcr
modPls <- plsr(Points ~ ., data = laligaRed2, scale = TRUE)

# The summary of the model
summary(modPls)
# First row: percentage of variance explained of the predictors
# Second row: percentage of variance explained of Y (the R^2)
# Note we have the same R^2 for 12 components as in the linear model

# Slots of information in the model
names(modPls)

# PLS scores
head(modPls$scores)

# Also uncorrelated
head(cov(modPls$scores))

# The coefficients of the original predictors, not of the components!
modPls$coefficients[, , 2]

# Obtaining the coefficients of the PLS components
lm(formula = Points ~ ., data = data.frame("Points" = laliga$Points,
                                             modPls$scores[, 1:3]))

# Prediction
predict(modPls, newdata = laligaRed2[1:2, ], ncomp = 12)

# Selecting the number of components to retain
modPls2 <- plsr(Points ~ ., data = laligaRed2, scale = TRUE, ncomp = 2)
summary(modPls2)

# Selecting the number of components to retain by Leave-One-Out cross-validation
modPlsCV1 <- plsr(Points ~ ., data = laligaRed2, scale = TRUE,
                    validation = "LOO")
summary(modPlsCV1)

# View cross-validation Mean Squared Error Prediction
validationplot(modPlsCV1, val.type = "MSEP") # l = 4 gives the minimum CV
```

<sup>7</sup>Of course, in practice, the computations need to be done in terms of the sample.

```

# Selecting the number of components to retain by 10-fold Cross-Validation
# (k = 10 is the default)
modPlsCV10 <- plsr(Points ~ ., data = laligaRed2, scale = TRUE,
                     validation = "CV")
summary(modPlsCV10)
validationplot(modPlsCV10, val.type = "MSEP")
# l = 4 is close to the minimum CV

# Regress manually Points in the scores, in order to have an lm object
# Create a new dataset with the response + PLS components
laligaPLS <- data.frame("Points" = laliga$Points, cbind(modPls$scores))

# Regression on all principal components
modPLS <- lm(Points ~ Comp.1 + Comp.2, data = laligaPLS)
summary(modPLS) # Predictors clearly significative - same R^2 as in modPls2
vif(modPLS) # No problems at all

```



For regression, PLS does a more clever selection of the directions of dimensionality reduction than PCA. However, **in practise PCR and PLS tend to perform similarly**, since PLS potentially increases the variance of the directions with respect to PCR.



Inference for PLS is more involved since, differently to what happens in PCR, the PLS directions are dependent on the response. We do not go into details on how to perform inference on PLS in these notes.



Let's perform PCR and PLS in the `iris` dataset. Recall that this dataset contains a factor variable, which can not be treated directly by `princomp` but can be used in PCR/PLS (it is transformed internally to two dummy variables). Do the following:

- Compute the PCA of `iris`, excluding `Species`. What is the percentage of variability explained with two components?
- Draw the biplot and look for interpretations of the principal components.
- Plot the PCA scores in a `scatterplotMatrix` plot such that the scores are coloured by the levels in `Species` (you can use the `groups` argument).
- Compute the PCR and PLS regressions of `Petal.Width` with  $l = 2$  (exclude `Species`). Inspect by CV the best selection of  $l$  for each regression.
- Plot the PLS scores of the data in a `scatterplotMatrix` plot such that the scores are coloured by the levels in `Species`. Compare with the PCA scores. (A quick and dirty way of converting the `scores` to a matrix is with `rbind`.)

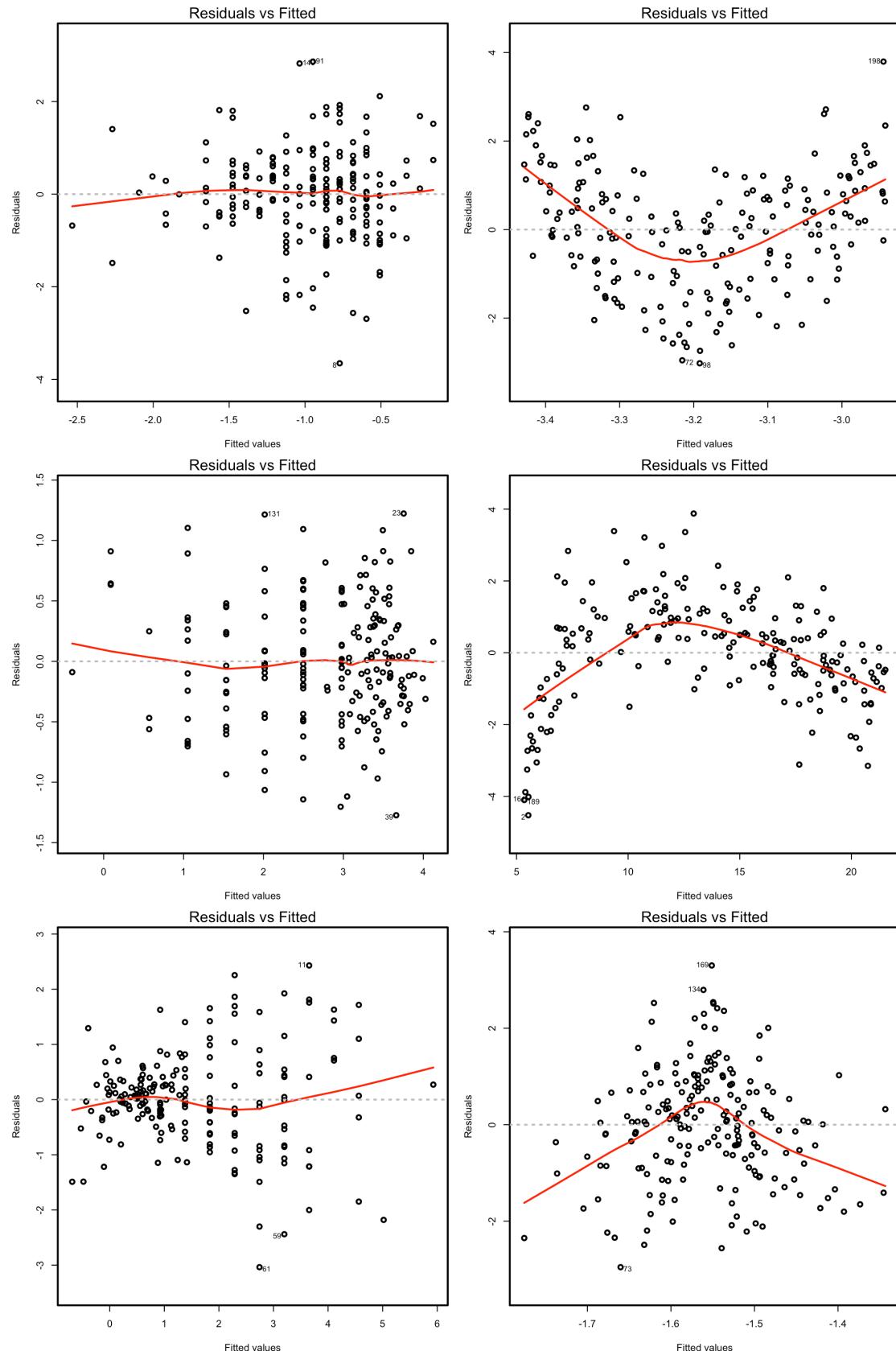


Figure 3.6: Residuals vs. fitted values plots for datasets respecting (left column) and violating (right column) the linearity assumption.

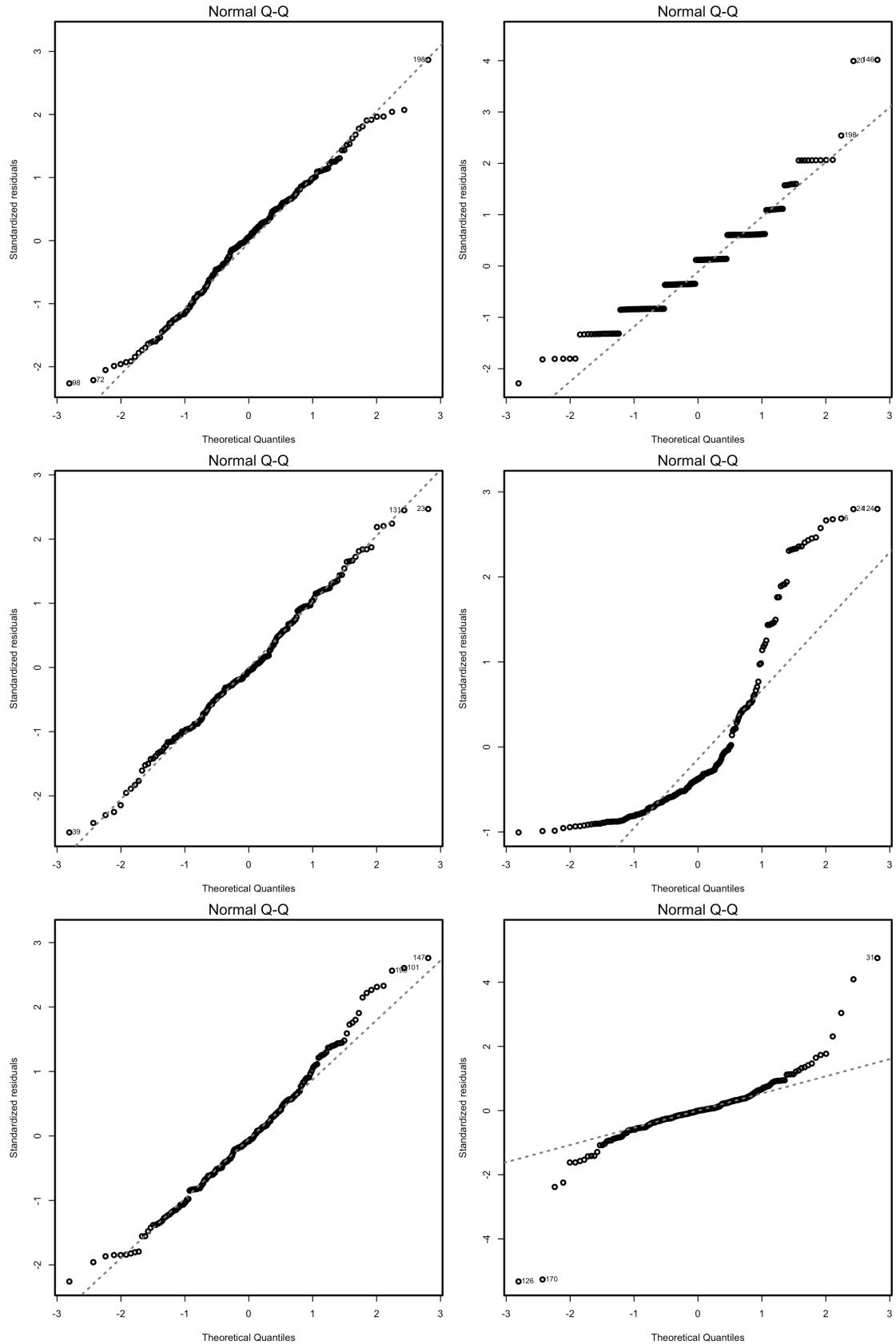


Figure 3.7: QQ-plots for datasets respecting (left column) and violating (right column) the normality assumption.

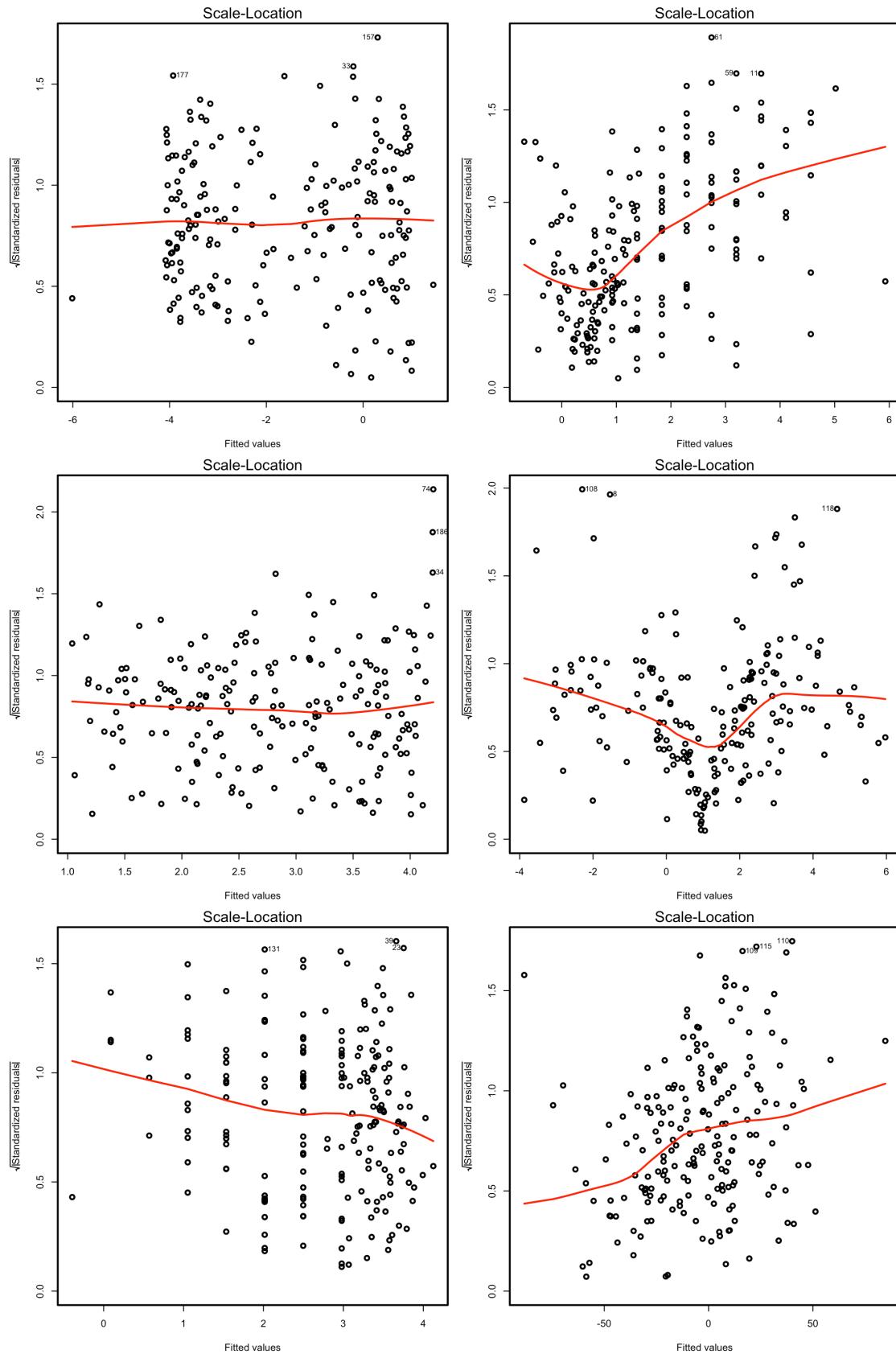


Figure 3.8: Scale-location plots for datasets respecting (left column) and violating (right column) the homoscedasticity assumption.

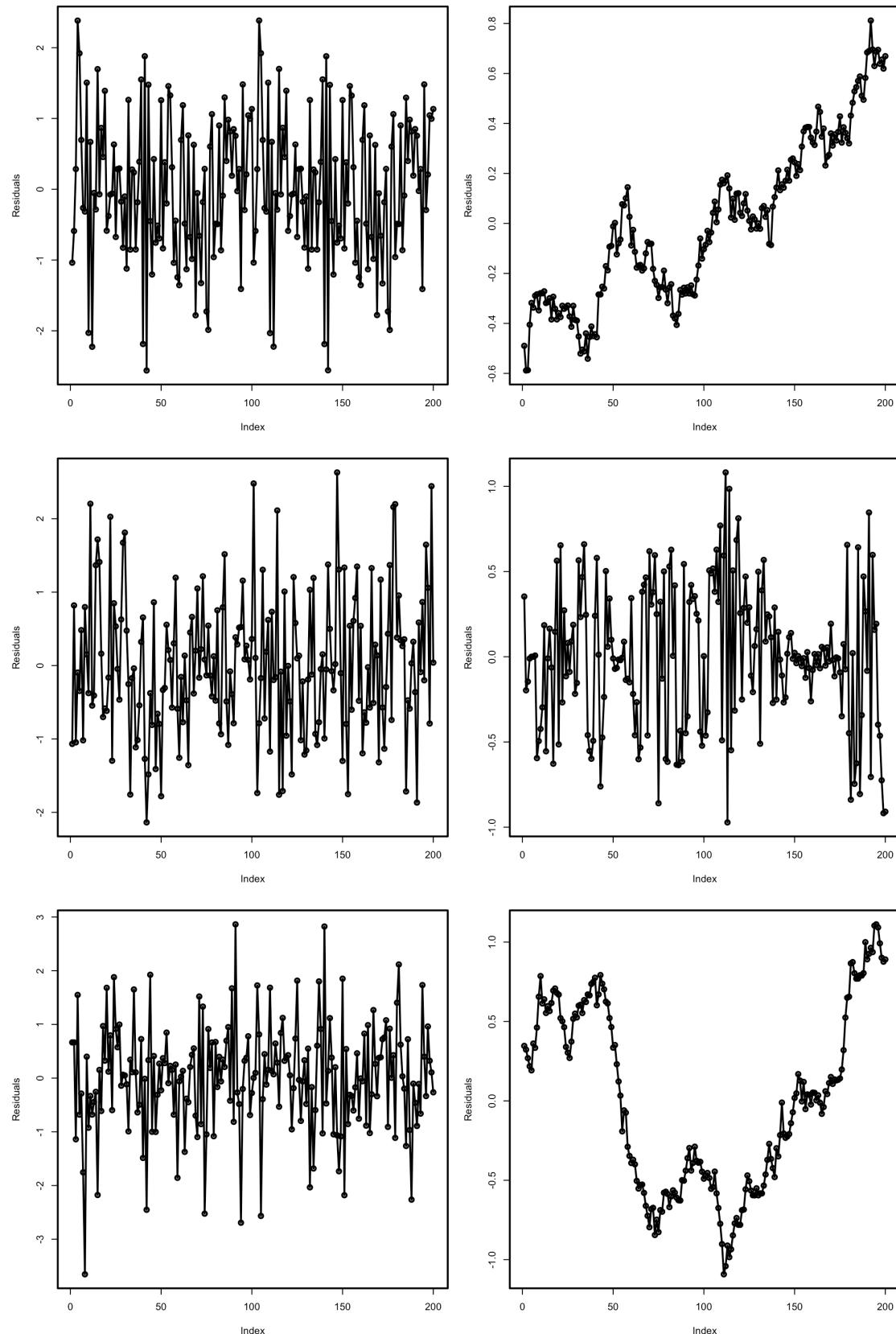


Figure 3.9: Serial plots of the residuals for datasets respecting (left column) and violating (right column) the independence assumption.



# Chapter 4

## Linear models III: shrinkage and big data

### 4.1 Shrinkage

As we saw in Section 2.4.1, the least squares estimates  $\hat{\beta}$  of the linear model

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \varepsilon,$$

were the minimizers of the residual sum of squares

$$\text{RSS}(\beta) = \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \dots - \beta_p X_{ip})^2.$$

Under the validity of the assumptions of Section 2.3, in Section 2.4 we saw that

$$\hat{\beta} \sim \mathcal{N}_{p+1}(\beta, \sigma^2(\mathbf{X}'\mathbf{X})^{-1}).$$

A particular consequence of this result is that  $\hat{\beta}$  is *unbiased* in estimating  $\beta$ , that is,  $\hat{\beta}$  does not make any systematic error in the estimation. However, bias is only one dimension of the quality of an estimate: variance is also important. Indeed, the *bias-variance trade-off* (see Section 1.3) arises from the bias-variance decomposition of the Mean Squared Error (MSE) of an estimate. For example, for the estimate  $\hat{\beta}_j$  of  $\beta_j$ , we have

$$\text{MSE}[\hat{\beta}_j] := \mathbb{E}[(\hat{\beta}_j - \beta_j)^2] = \underbrace{(\mathbb{E}[\hat{\beta}_j] - \beta_j)^2}_{\text{Bias}^2} + \underbrace{\text{Var}[\hat{\beta}_j]}_{\text{Variance}}. \quad (4.1)$$

*Shrinkage* methods pursue the following idea:

Add a small amount of *smart bias* to  $\hat{\beta}$  in order to reduce its variance, in such a way that we obtain *simpler interpretations* from the biased version of  $\hat{\beta}$ .

This is done by enforcing *sparsity*, that is, by biasing the estimates of  $\beta$  towards being non-null only in the most important relations between the response and predictors. The two methods covered in this section, **ridge regression** and **lasso** (*least absolute shrinkage and selection operator*), use this idea in a different way. Both methods consider the standard linear model, what is different now is *the way of estimating  $\beta$* .

The way they enforce sparsity in the estimates is by minimizing the RSS plus a penalty term that favors sparsity on the estimated coefficients:

$$\text{RSS}(\beta) + \lambda \sum_{j=1}^p |\beta_j|^{(2-\alpha)}. \quad (4.2)$$

Ridge regression corresponds to  $\alpha = 0$  (quadratic penalty) and lasso to  $\alpha = 1$  (linear penalty). Obviously, if  $\lambda = 0$ , we are back to the least squares problem and theory. The optimization of (4.2) gives

$$\hat{\beta}_\lambda := \arg \min_{\beta \in \mathbb{R}^{p+1}} \left\{ \text{RSS}(\beta) + \lambda \sum_{j=1}^p |\beta_j|^{(2-\alpha)} \right\}, \quad (4.3)$$

which is the penalized estimation of  $\beta$ . Note that the sparsity is enforced in the slopes, not in the intercept, since this depends on the scale of  $Y$ .  $\lambda$  is a tuning parameter that will need to be chosen suitably. What is important now is to recall that the *predictors need to be standardized*, or otherwise its scale will distort the optimization of (4.2).

An equivalent way of viewing (4.3) that helps in visualizing the differences between the ridge and lasso regressions try to solve

$$\hat{\beta}_\lambda := \arg \min_{\beta \in \mathbb{R}^{p+1}: \sum_{j=1}^p |\beta_j|^{(2-\alpha)} \leq s_\lambda} \text{RSS}(\beta), \quad (4.4)$$

where  $s_\lambda$  is certain scalar that does not depend on  $\beta$ .

We will work with the `Hitters` dataset from the `ISLR` package. It contains statistics and salaries from baseball players from the 1986 and 1987 seasons. The objective will be to predict the `Salary` from the remaining predictors.

```
# Load data - baseball players statistics
library(ISLR)
data(Hitters)

# Discard NA's
Hitters <- na.omit(Hitters)

# The glmnet function works with the design matrix of predictors (without
# the ones). This can be obtained easily through model.matrix()
x <- model.matrix(Salary ~ ., data = Hitters)[, -1]

# Interestingly, note that in Hitters there are two-level factors and these
# are automatically transformed into dummy variables in x
head(Hitters[, 14:20])
##          League Division PutOuts Assists Errors Salary NewLeague
## -Alan Ashby      N       W     632      43     10  475.0        N
## -Alvin Davis     A       W     880      82     14  480.0        A
## -Andre Dawson    N       E     200      11      3  500.0        N
## -Andres Galarraga N       E     805      40      4  91.5        N
## -Alfredo Griffin  A       W     282     421     25  750.0        A
## -Al Newman        N       E      76     127      7  70.0        A
head(x[, 14:19])
##          LeagueN DivisionW PutOuts Assists Errors NewLeagueN
```

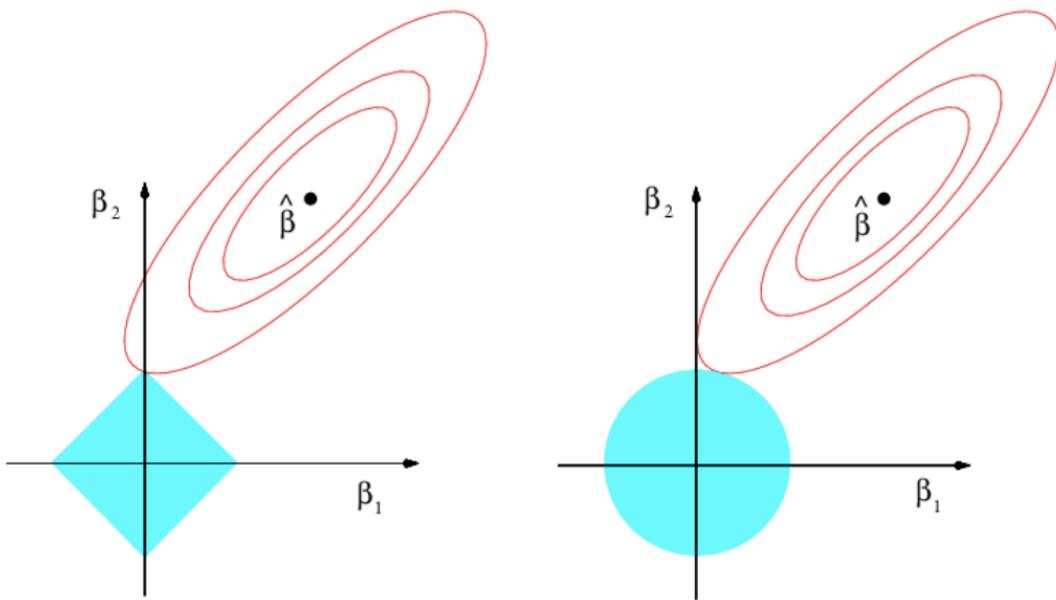


Figure 4.1: Comparison of ridge and lasso solutions from the optimization problem (4.4) with  $p = 2$ . The elliptical contours show the regions with equal  $\text{RSS}(\beta_1, \beta_2)$ , the objective function, for  $(\beta_1, \beta_2) \in \mathbb{R}^2$  ( $\beta_0 = 0$  is assumed). The diamond ( $\alpha = 1$ ) and circular ( $\alpha = 0$ ) regions show the feasibility regions determined by  $\sum_{j=1}^p |\beta_j|^{(2-\alpha)} \leq s_\lambda$  for the optimization problem. The *sharpness* of the diamond makes the lasso attain solutions with many coefficients *exactly* zero, in a similar situation to the one depicted. Extracted from James et al. (2013).

```

## -Alan Ashby      1      1    632     43    10      1
## -Alvin Davis    0      1    880     82    14      0
## -Andre Dawson    1      0    200     11     3      1
## -Andres Galarraga 1      0    805     40     4      1
## -Alfredo Griffin 0      1    282    421    25      0
## -Al Newman       1      0     76    127     7      0

# We also need the vector of responses
y <- Hitters$Salary

```

### 4.1.1 Ridge regression

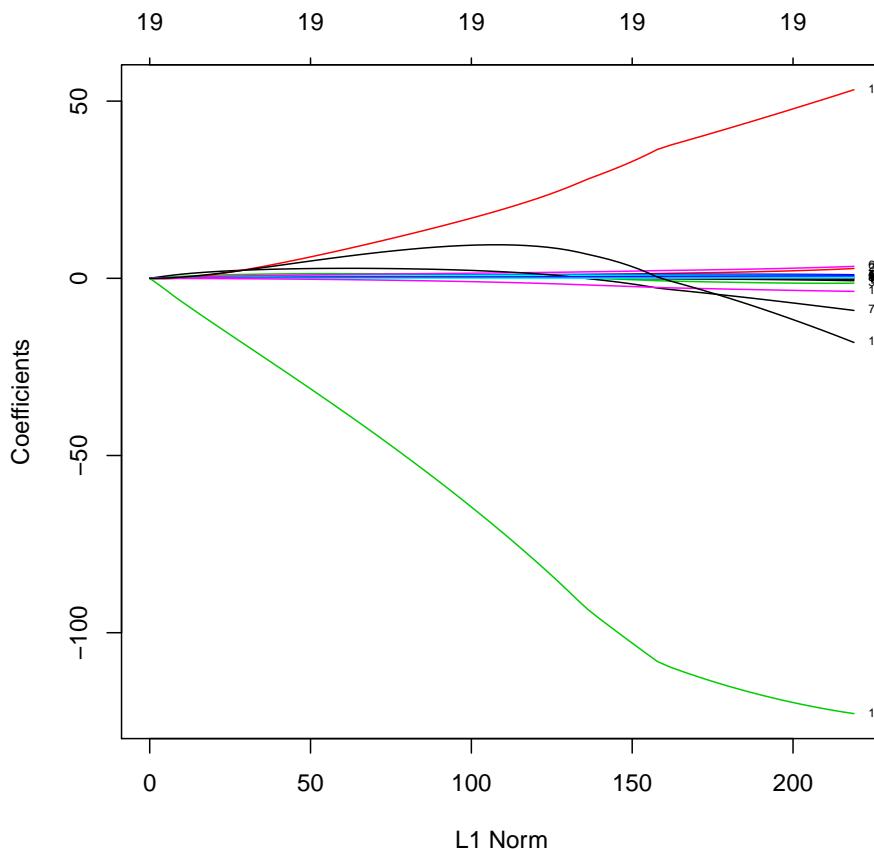
#### Fitting

```

# Call to the main function - use alpha = 0 for ridge regression
library(glmnet)
ridgeMod <- glmnet(x = x, y = y, alpha = 0)
# By default, it computes the ridge solution over a set of lambdas
# automatically chosen. It also standardizes the variables by default to make
# the model fitting since the penalization is scale-sensitive. Importantly,
# the coefficients are returned on the original scale of the predictors

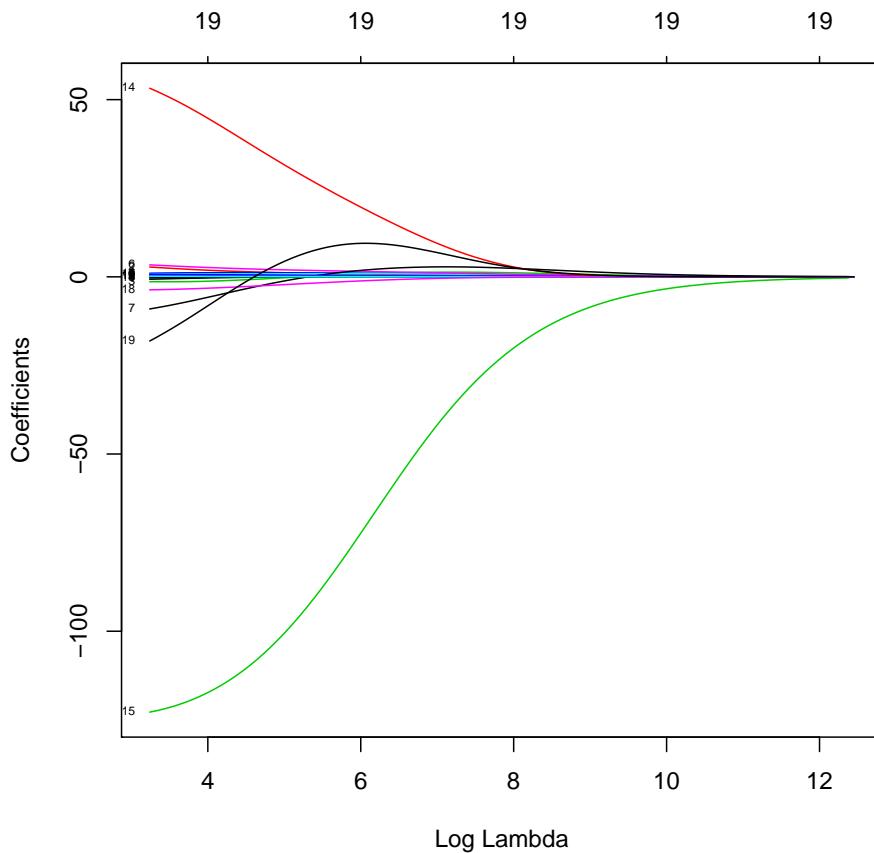
```

```
# Plot of the solution path - gives the value of the coefficients for different
# measures in xvar (penalization imposed to the model or fitness)
plot(ridgeMod, xvar = "norm", label = TRUE)
```

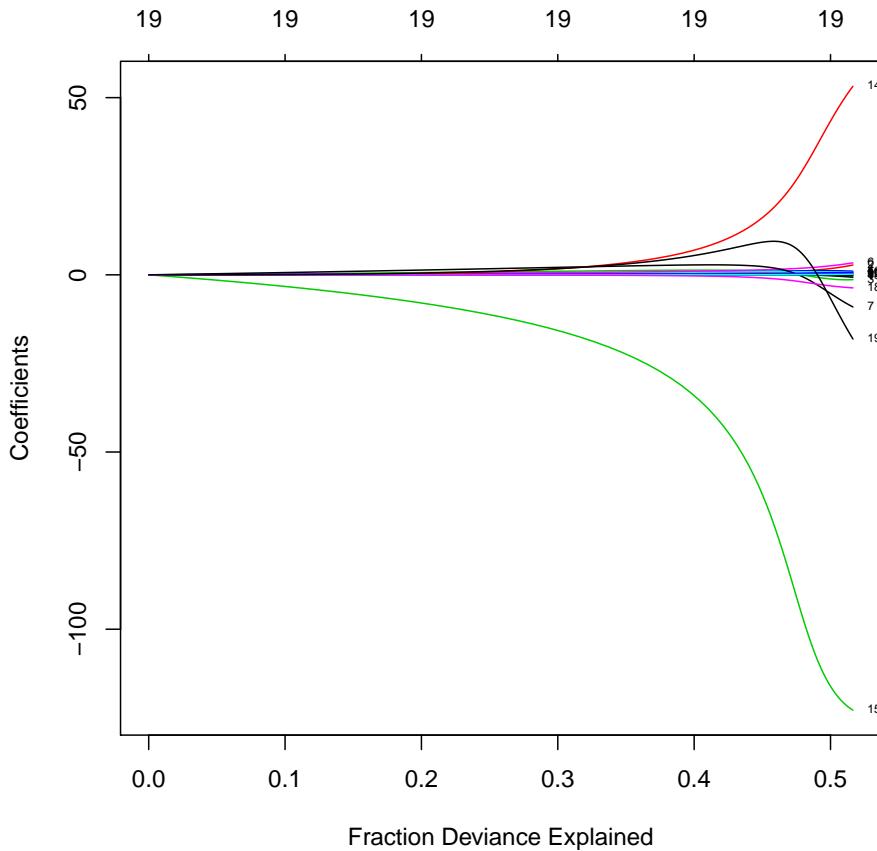


```
# xvar = "norm" is the default: L1 norm of the coefficients sum_j abs(beta_j)

# Versus lambda
plot(ridgeMod, label = TRUE, xvar = "lambda")
```



```
# Versus the percentage of deviance explained - this is a generalization of the
# R^2 for generalized linear models. Since we have a linear model, this is the
# same as the R^2
plot(ridgeMod, label = TRUE, xvar = "dev")
```



```

# The maximum  $R^2$  is slightly above 0.5

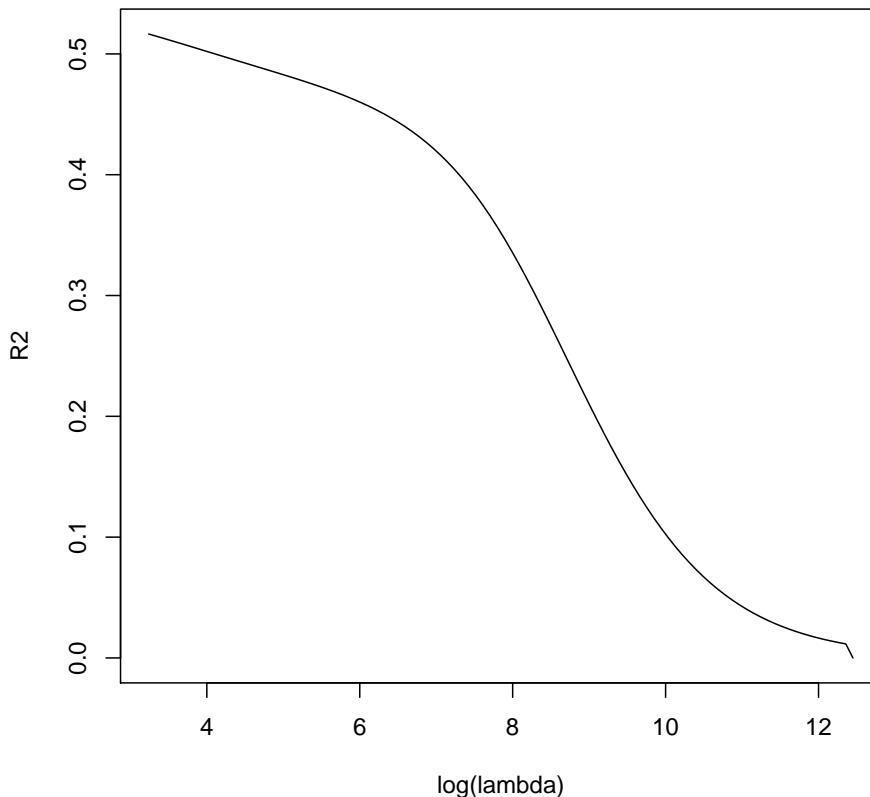
# Indeed, we can see that  $R^2 = 0.5461$ 
summary(lm(Salary ~ ., data = Hitters))$r.squared
## [1] 0.5461159

# Some persistently important predictors are 15, 14, and 19
colnames(x)[c(15, 14, 19)]
## [1] "DivisionW" "LeagueN"   "NewLeagueN"

# What is inside glmmnet's output?
names(ridgeMod)
##  [1] "a0"          "beta"        "df"          "dim"         "lambda"
##  [6] "dev.ratio"   "nulldev"     "npasses"     "jerr"        "offset"
## [11] "call"        "nobs"

# lambda versus  $R^2$  - fitness decreases when sparsity is introduced, in
# in exchange of better variable interpretation and avoidance of overfitting
plot(log(ridgeMod$lambda), ridgeMod$dev.ratio, type = "l",
     xlab = "log(lambda)", ylab = "R2")

```



```

ridgeMod$dev.ratio[length(ridgeMod$dev.ratio)]
## [1] 0.5164455
# Slightly different to lm's because of compromises in accuracy for speed

# The coefficients for different values of lambda are given in $a0 (intercepts)
# and $beta (slopes) or, alternatively, both in coef(ridgeMod)
length(ridgeMod$a0)
## [1] 100
dim(ridgeMod$beta)
## [1] 19 100
length(ridgeMod$lambda) # 100 lambda's were automatically chosen
## [1] 100

# Inspecting the coefficients associated to the 50th lambda
coef(ridgeMod)[, 50]
##   (Intercept)      AtBat      Hits      HmRun      Runs
## 213.066443434  0.090095728  0.371252756  1.180126956  0.596298287
##      RBI      Walks      Years      CAtBat      CHits
##  0.594502390  0.772525466  2.473494238  0.007597952  0.029272172
##     CHmRun      CRuns      CRBI      CWalks      LeagueN
##  0.217335716  0.058705097  0.060722036  0.058698830  3.276567828
##    DivisionW      PutOuts      Assists      Errors      NewLeagueN
## -21.889942619  0.052667119  0.007463678 -0.145121336  2.972759126
ridgeMod$lambda[50]
## [1] 2674.375

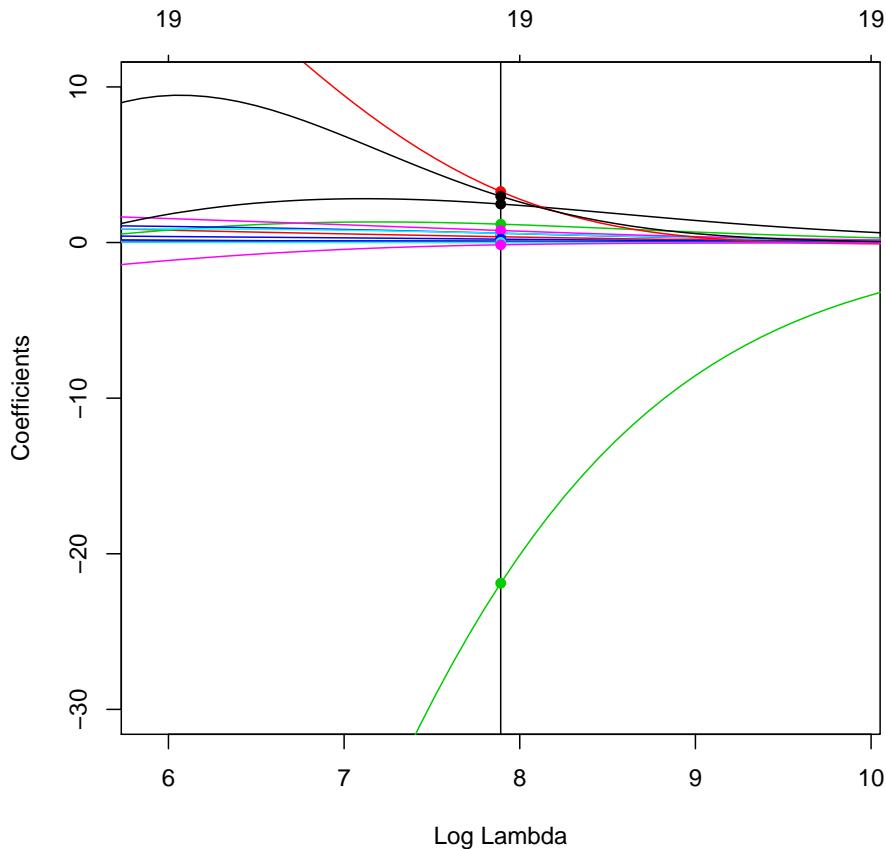
# Zoom in path solution
plot(ridgeMod, label = TRUE, xvar = "lambda",

```

```

xlim = log(ridgeMod$lambda[50]) + c(-2, 2), ylim = c(-30, 10))
abline(v = log(ridgeMod$lambda[50]))
points(rep(log(ridgeMod$lambda[50]), 19), ridgeMod$beta[, 50],
      pch = 16, col = 1:6)

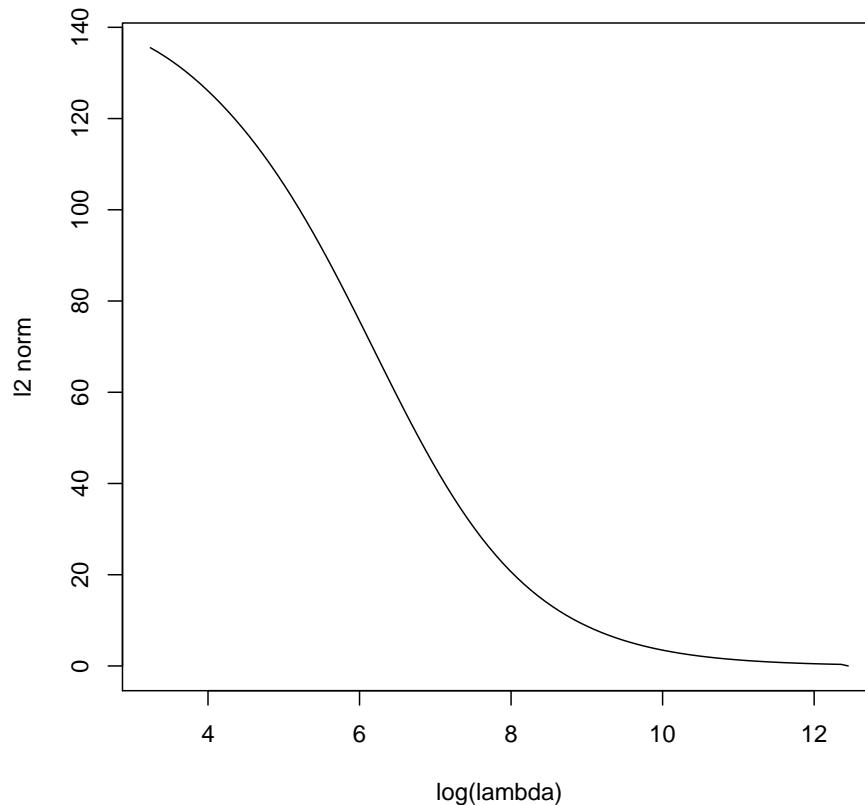
```



```

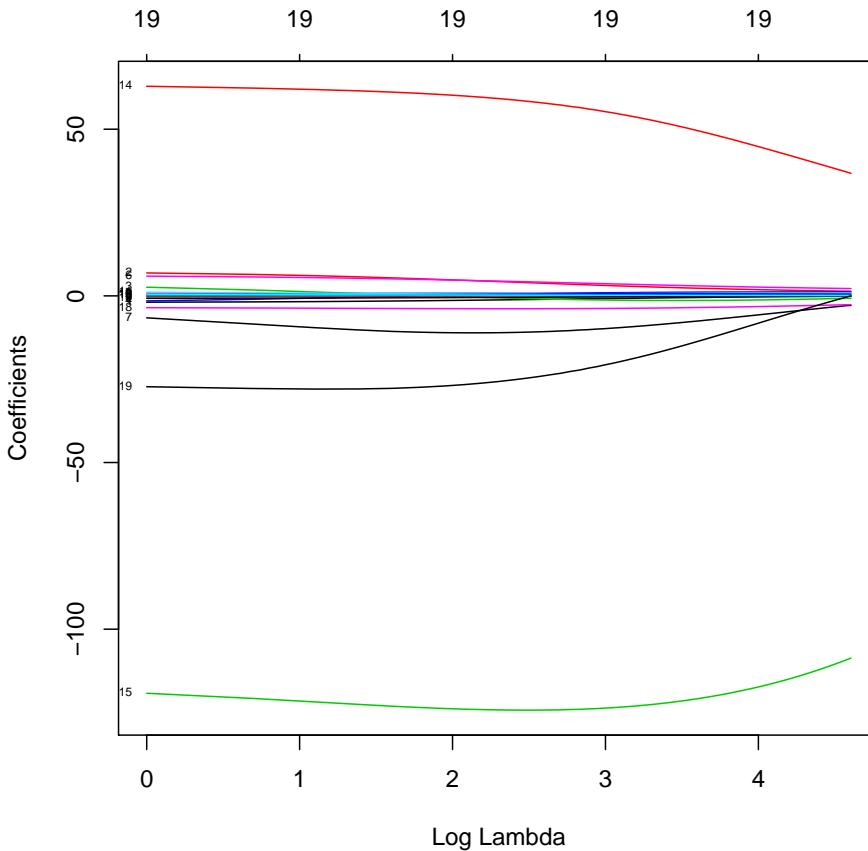
# The squared l2-norm of the coefficients decreases as lambda increases
plot(log(ridgeMod$lambda), sqrt(colSums(ridgeMod$beta^2)), type = "l",
      xlab = "log(lambda)", ylab = "l2 norm")

```



### Tuning parameter selection

```
# If we want, we can choose manually the grid of penalty parameters to explore
# The grid should be descending
ridgeMod2 <- glmnet(x = x, y = y, alpha = 0, lambda = 100:1)
plot(ridgeMod2, label = TRUE, xvar = "lambda") # Not a good choice!
```



```

# Lambda is a tuning parameter that can be chosen by cross-validation, using as
# error the MSE (other possible error can be considered for generalized models
# using the argument type.measure)

# 10-fold cross-validation. Change the seed for a different result.
set.seed(12345)
kcvRidge <- cv.glmnet(x = x, y = y, alpha = 0, nfolds = 10)
# The lambda grid in which CV is done is automatically selected

# The lambda that minimises the CV error is
kcvRidge$lambda.min
## [1] 28.01718

# Equivalent to
indMin <- which.min(kcvRidge$cvm)
kcvRidge$lambda[indMin]
## [1] 28.01718

# The minimum CV error
kcvRidge$cvm[indMin]
## [1] 117431.3
min(kcvRidge$cvm)
## [1] 117431.3

# Potential problem! Minimum occurs at one extreme of the lambda grid in which
# CV is done. This was automatically selected, but can be manually inputted

```

```

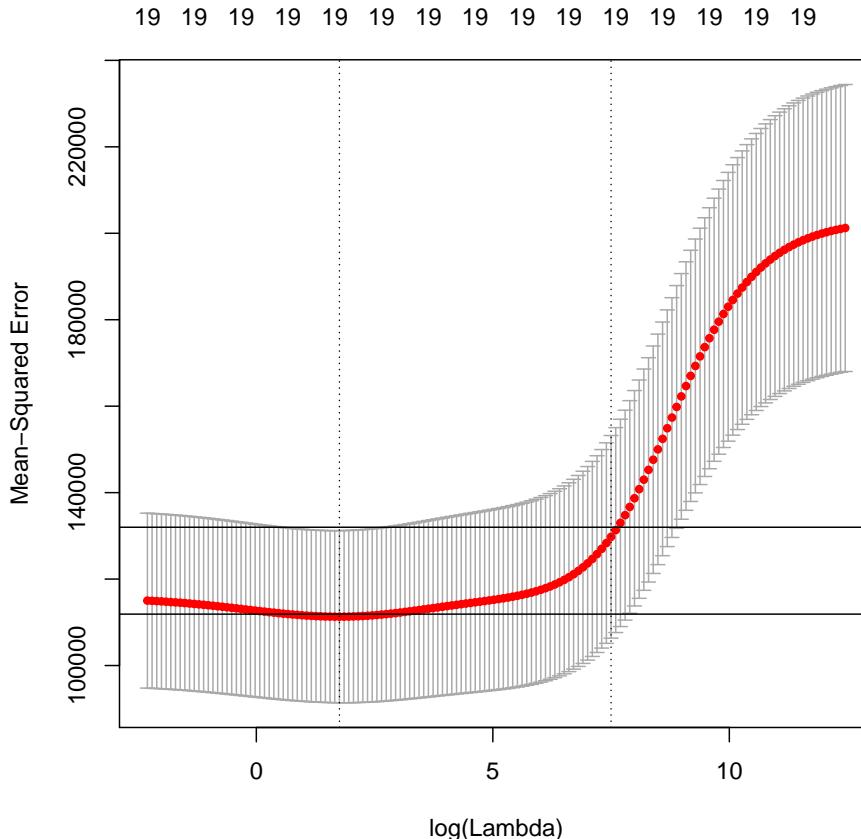
range(kcvRidge$lambda)
## [1] 28.01718 255282.09651
lambdaGrid <- 10^seq(log10(kcvRidge$lambda[1]), log10(0.1),
                      length.out = 150)
kcvRidge2 <- cv.glmnet(x = x, y = y, nfolds = 10, alpha = 0,
                        lambda = lambdaGrid)

# Much better
plot(kcvRidge2)
kcvRidge2$lambda.min
## [1] 5.794362

# But the CV curve is random, since it depends on the sample. Its variability
# can be estimated by considering the CV curves of each fold. An alternative
# approach to select lambda is to choose the largest within one standard
# deviation of the minimum error, in order to favour simplicity of the model
# around the optimal lambda value. This is known as the "one standard error rule"
kcvRidge2$lambda.1se
## [1] 1807.23

# Location of both optimal lambdas in the CV loss function in dashed vertical
# lines, and lowest CV error and lowest CV error + one standard error
plot(kcvRidge2)
abline(h = kcvRidge2$cvm[indMin] + c(0, kcvRidge2$cvsd[indMin]))

```



```

# The consideration of the one standard error rule for selecting lambda makes
# special sense when the CV function is quite flat around the minimum (hence an

```

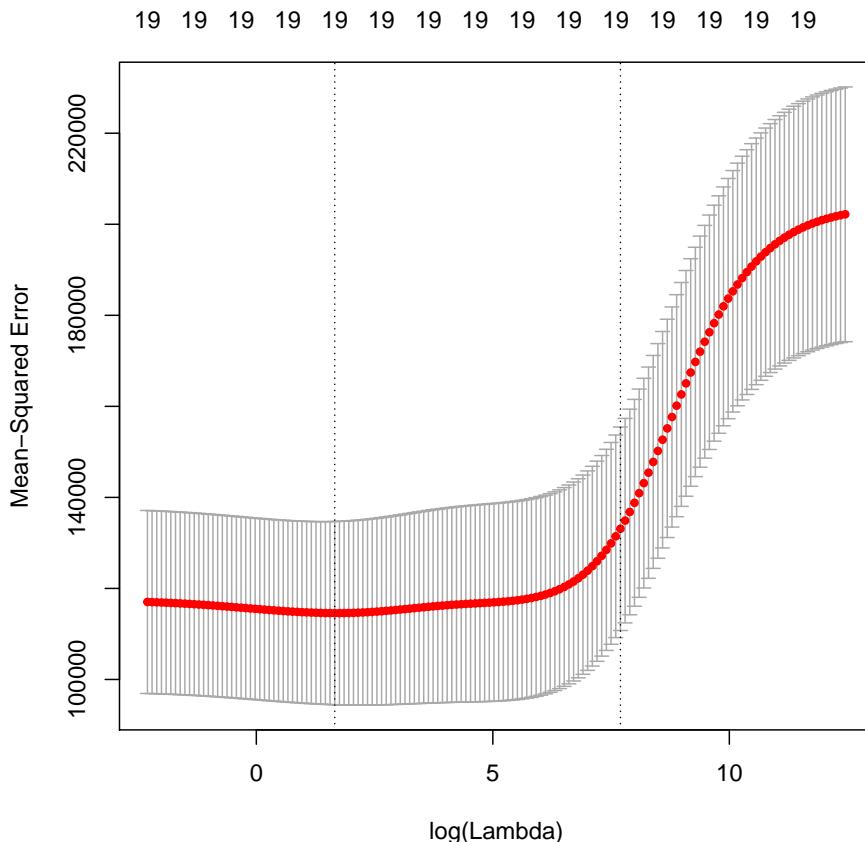
```

# overpenalization that gives more sparsity does not affect so much the CV loss)

# Leave-one-out cross-validation. More computationally intense but completely
# objective in the choice of the fold-assignment
ncvRidge <- cv.glmnet(x = x, y = y, alpha = 0, nfolds = nrow(Hitters),
                      lambda = lambdaGrid)
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

# Location of both optimal lambdas in the CV loss function
plot(ncvRidge)

```



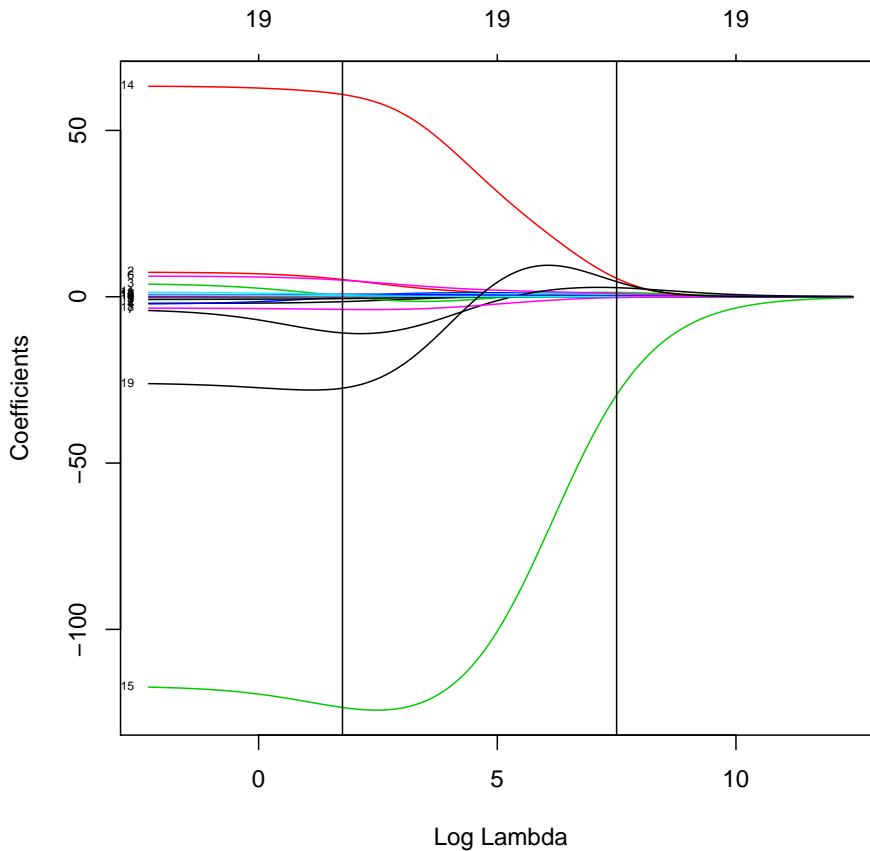
## Prediction

```

# The glmnet fit is inside the output of cv.glmnet
modRidgeCV <- kcvRidge2$glmnet.fit

# Inspect the best models
plot(modRidgeCV, label = TRUE, xvar = "lambda")
abline(v = log(c(kcvRidge2$lambda.min, kcvRidge2$lambda.1se)))

```



```

# The model associated to lambda.1se (or any other lambda not included in the
# original path solution - obtained by an interpolation) can be retrieved with
predict(modRidgeCV, type = "coefficients", s = kcvRidge2$lambda.1se)
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                                     1
## (Intercept) 157.091705238
## AtBat        0.103054749
## Hits         0.450955230
## HmRun        1.292991027
## Runs         0.708454038
## RBI          0.691436450
## Walks        0.934205919
## Years        2.724001539
## CAtBat       0.008803293
## CHits        0.034626132
## CHmRun       0.255471355
## CRuns        0.069406759
## CRBI         0.071893549
## CWalks       0.066451221
## LeagueN      5.528124640
## DivisionW   -29.524169177
## PutOuts      0.068685576
## Assists      0.009301321
## Errors       -0.241992404
## NewLeagueN   4.544466824

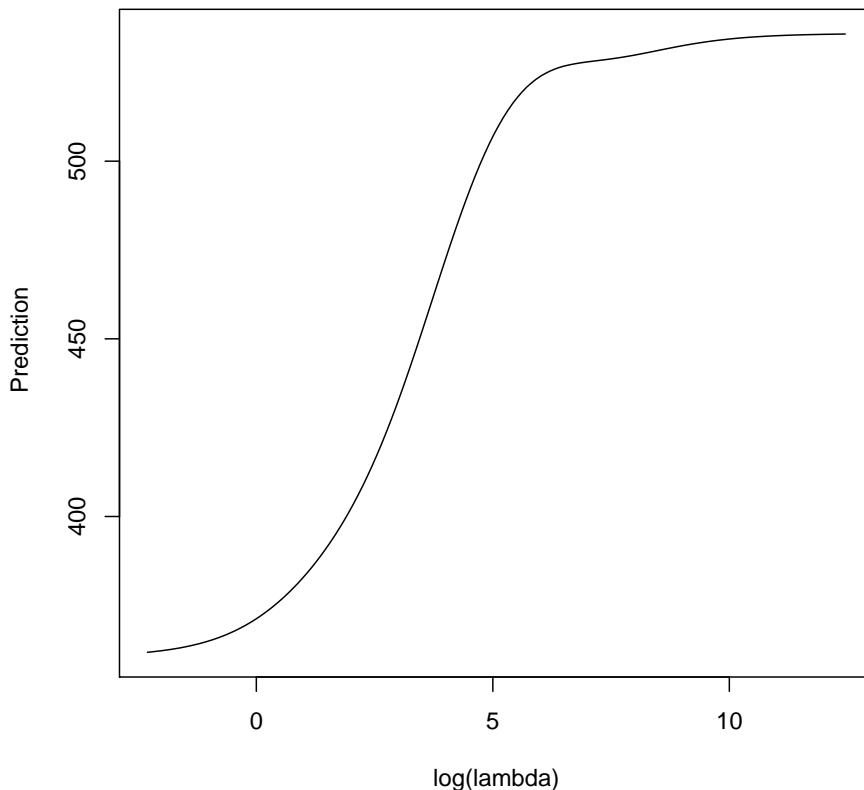
```

```

# Predictions for the first two observations
predict(modRidgeCV, type = "response", s = kcvRidge2$lambda.1se,
        newx = x[1:2, ])
##          1
## -Alan Ashby 528.7831
## -Alvin Davis 590.7228

# Predictions for the first observation, for all the lambdas. We can see how
# the prediction for one observation changes according to lambda
plot(log(modRidgeCV$lambda),
     predict(modRidgeCV, type = "response", newx = x[1, , drop = FALSE]),
     type = "l", xlab = "log(lambda)", ylab = "Prediction")

```



### 4.1.2 Lasso

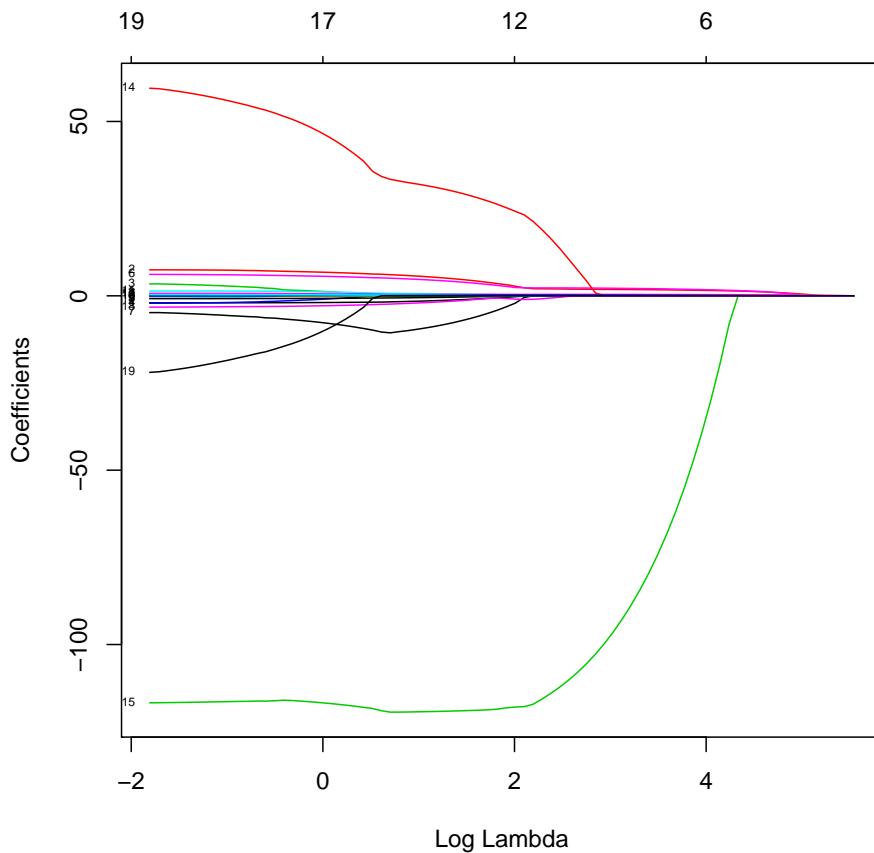
#### Fitting and model selection

```

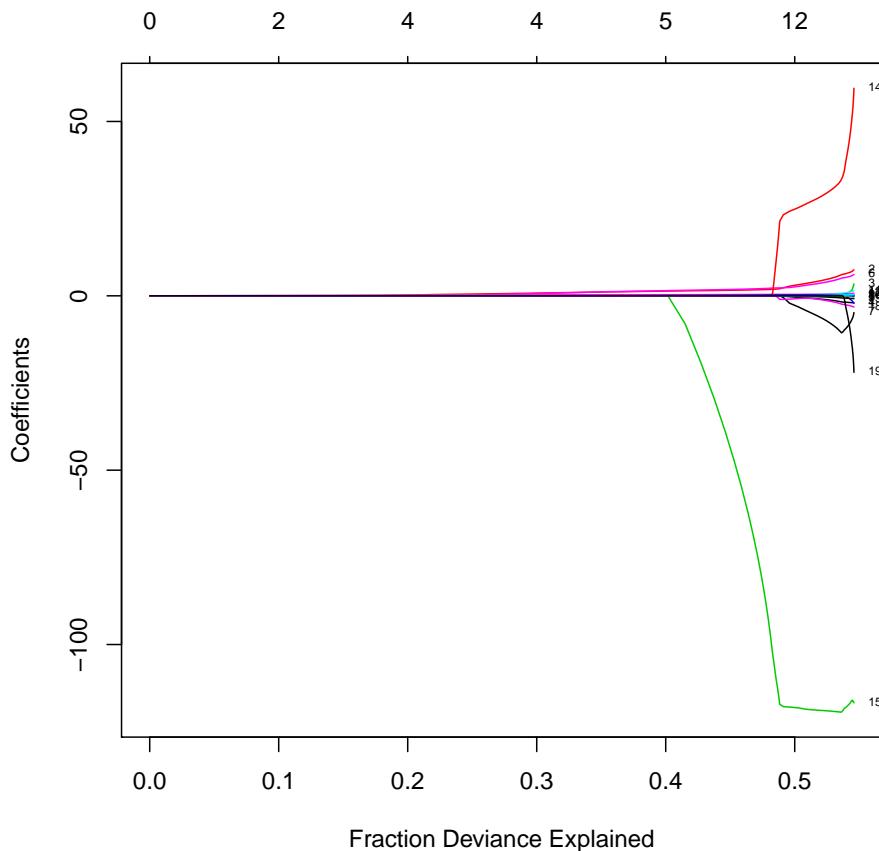
# Call to the main function - use alpha = 1 for lasso regression (the default)
lassoMod <- glmnet(x = x, y = y, alpha = 1)
# Same defaults as before, same object structure

# Plot of the solution path - now the paths are not smooth when decreasing to
# zero (they are zero exactly). This is a consequence of the l1 norm
plot(lassoMod, xvar = "lambda", label = TRUE)

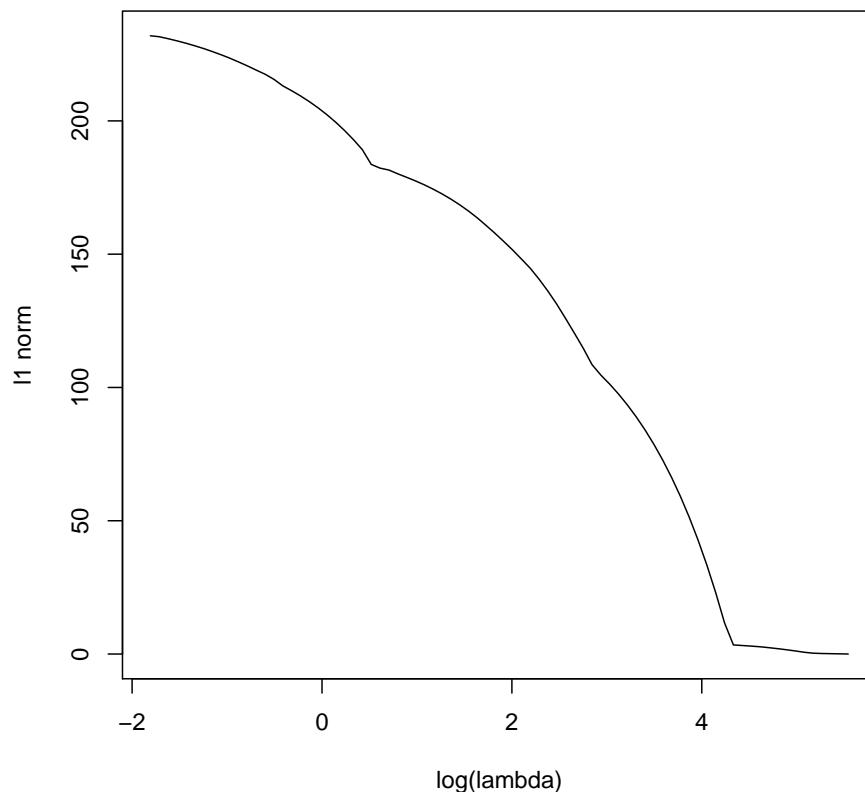
```



```
# Some persistently important predictors are 15, 14, and 19
# Versus the R^2 - same maximum R^2 as before
plot(lassoMod, label = TRUE, xvar = "dev")
```



```
# Now the l1-norm of the coefficients decreases as lambda increases
plot(log(lassoMod$lambda), colSums(abs(lassoMod$beta)), type = "l",
     xlab = "log(lambda)", ylab = "l1 norm")
```

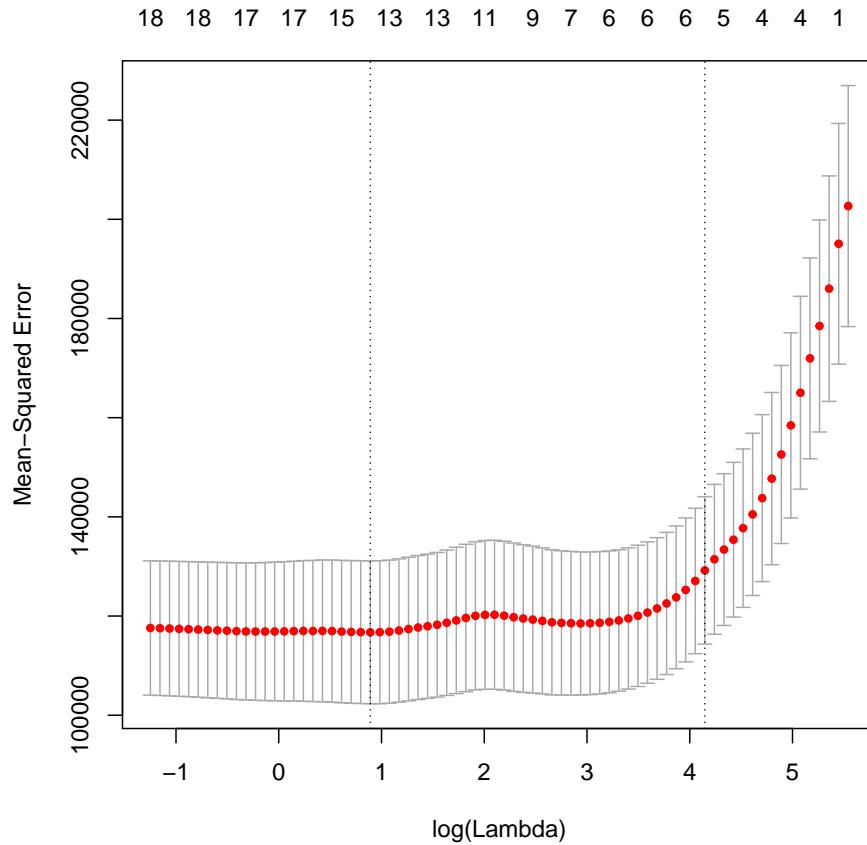


```
# 10-fold cross-validation. Change the seed for a different result.
set.seed(12345)
kcvLasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = 10)

# The lambda that minimises the CV error is
kcvLasso$lambda.min
## [1] 2.436791

# The "one standard error rule" for lambda
kcvLasso$lambda.1se
## [1] 63.23532

# Location of both optimal lambdas in the CV loss function
plot(kcvLasso)
abline(h = kcvLasso$cvm[indMin] + c(0, kcvLasso$cvsd[indMin]))
```



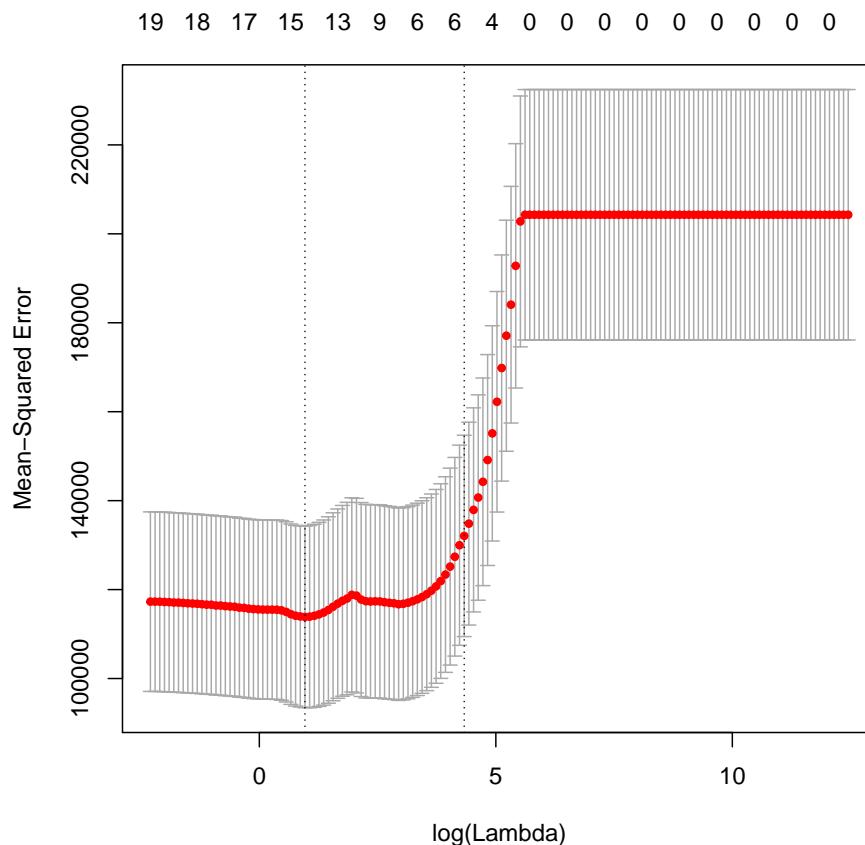
```

# No problems now: minimum does not occur at one extreme
# Interesting: note that the numbers on top of the figure gives the number of
# coefficients *exactly* different from zero - the number of predictors
# effectively considered in the model!
# In this case, the one standard error rule makes also sense

# Leave-one-out cross-validation
ncvLasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = nrow(Hitters),
                      lambda = lambdaGrid)
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold

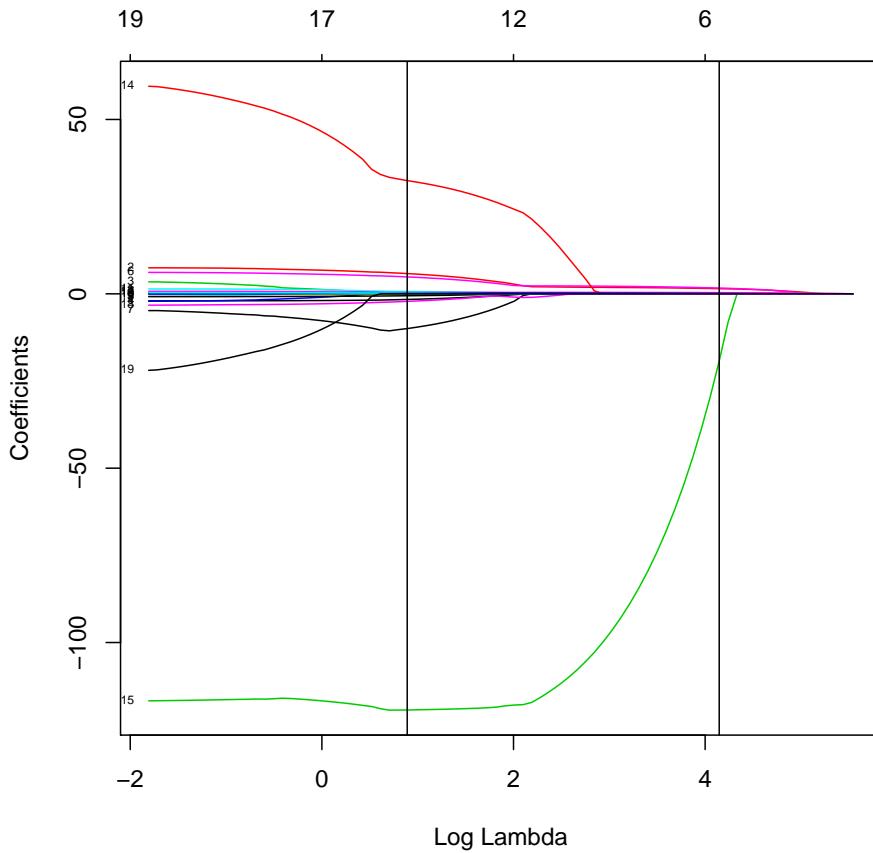
# Location of both optimal lambdas in the CV loss function
plot(ncvLasso)

```



## Prediction

```
# Inspect the best models
modLassoCV <- kcvLasso$glmnet.fit
plot(modLassoCV, label = TRUE, xvar = "lambda")
abline(v = log(c(kcvLasso$lambda.min, kcvLasso$lambda.1se)))
```



```

# The model associated to lambda.min (or any other lambda not included in the
# original path solution - obtained by an interpolation) can be retrieved with
predict(modLassoCV, type = "coefficients",
       s = c(kcvLasso$lambda.min, kcvLasso$lambda.1se))
## 20 x 2 sparse Matrix of class "dgCMatrix"
##           1          2
## (Intercept) 129.4155571 115.3773590
## AtBat      -1.6130155   .
## Hits       5.8058915  1.4753071
## HmRun      .
## Runs       .
## RBI        .
## Walks      4.8469340  1.6566947
## Years     -9.9724045   .
## CAtBat     .
## CHits      .
## CHmRun     0.5374550   .
## CRuns      0.6811938  0.1660465
## CRBI       0.3903563  0.3453397
## CWalks     -0.5560144   .
## LeagueN    32.4646094   .
## DivisionW -119.3480842 -19.2435216
## PutOuts     0.2741895  0.1000068
## Assists     0.1855978   .
## Errors     -2.1650837   .
## NewLeagueN .

```

```
# Predictions for the first two observations
predict(modLassoCV, type = "response",
        s = c(kcvLasso$lambda.min, kcvLasso$lambda.1se),
        newx = x[1:2, ])
##          1         2
## -Alan Ashby 422.5936 539.7206
## -Alvin Davis 700.8934 630.8933
```

## Variable selection

```
# We can use lasso for model selection!
selPreds <- predict(modLassoCV, type = "coefficients",
                     s = c(kcvLasso$lambda.min, kcvLasso$lambda.1se))[-1, ] > 0
x1 <- x[, selPreds[, 1]]
x2 <- x[, selPreds[, 2]]

# Least squares fit with variables selected by lasso
summary(lm(y ~ x1))
##
## Call:
## lm(formula = y ~ x1)
##
## Residuals:
##      Min      1Q      Median      3Q      Max
## -937.37 -170.34  -22.57  102.47 2185.04
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -122.91388  61.95178 -1.984 0.048329 *
## x1Hits       2.27523   0.62276  3.653 0.000314 ***
## x1Walks      2.30446   1.28779  1.789 0.074732 .
## x1CHmRun     0.33320   0.81790  0.407 0.684071
## x1CRuns      0.31278   0.22128  1.414 0.158732
## x1CRBI       0.27209   0.34717  0.784 0.433921
## x1LeagueN    46.20082  42.96703  1.075 0.283278
## x1PutOuts    0.25901   0.08017  3.231 0.001397 **
## x1Assists    -0.03601   0.15972 -0.225 0.821802
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 334.2 on 254 degrees of freedom
## Multiple R-squared:  0.468,  Adjusted R-squared:  0.4513
## F-statistic: 27.93 on 8 and 254 DF,  p-value: < 2.2e-16
summary(lm(y ~ x2))
##
## Call:
## lm(formula = y ~ x2)
##
## Residuals:
##      Min      1Q      Median      3Q      Max
## -914.21 -171.94  -33.26  97.63 2197.08
##
```

```

## Coefficients:
##              Estimate Std. Error t value Pr(>|t|) 
## (Intercept) -96.96096  55.62583 -1.743 0.082513 .
## x2Hits       2.09338   0.57376  3.649 0.000319 ***
## x2Walks      2.51513   1.22010  2.061 0.040269 *  
## x2CRuns      0.26490   0.19463  1.361 0.174679
## x2CRBI       0.39549   0.19755  2.002 0.046339 *  
## x2PutOuts     0.26620   0.07857  3.388 0.000814 ***
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 333 on 257 degrees of freedom
## Multiple R-squared:  0.4654, Adjusted R-squared:  0.455 
## F-statistic: 44.75 on 5 and 257 DF,  p-value: < 2.2e-16

# Comparison with stepwise selection
modBIC <- stepAIC(lm(Salary ~ ., data = Hitters), k = log(nrow(Hitters)),
                    trace = 0)
summary(modBIC)
## 
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks + CRuns + CRBI + CWalks +
##     Division + PutOuts, data = Hitters)
## 
## Residuals:
##      Min      1Q      Median      3Q      Max 
## -794.06 -171.94  -28.48  133.36 2017.83 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|) 
## (Intercept) 117.15204  65.07016  1.800 0.072985 .
## AtBat       -2.03392   0.52282 -3.890 0.000128 ***
## Hits        6.85491   1.65215  4.149 4.56e-05 ***
## Walks       6.44066   1.52212  4.231 3.25e-05 ***
## CRuns       0.70454   0.24869  2.833 0.004981 ** 
## CRBI        0.52732   0.18861  2.796 0.005572 ** 
## CWalks      -0.80661   0.26395 -3.056 0.002483 ** 
## DivisionW  -123.77984  39.28749 -3.151 0.001824 ** 
## PutOuts      0.27539   0.07431  3.706 0.000259 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 314.7 on 254 degrees of freedom
## Multiple R-squared:  0.5281, Adjusted R-squared:  0.5133 
## F-statistic: 35.54 on 8 and 254 DF,  p-value: < 2.2e-16

# The lasso selections are similar, although the model is slightly worse in
# terms of adjusted R^2 and significance of the predictors. However, keep in
# mind that lasso is solving a constrained least squares problem, so it is
# expected to achieve better R^2 and adjusted R^2 via a selection procedure
# that employs solutions of unconstrained least squares. What is remarkable
# is the speed of lasso on selecting variables, and the fact that gives quite
# good starting points for performing further model selection

```



Consider `la-liga-2015-2016.xlsx` dataset. We will predict `Points` after removing the perfectly related linear variables to it. Do the following:

- Lasso regression. Select  $\lambda$  by cross-validation. Obtain the estimated coefficients for the chosen `lambda`.
- Use the predictors with non-null coefficients for creating a model with `lm`.
- Summarize the model and check for multicollinearity.

## 4.2 A note of caution with inference after model-selection



Inferences from models that result from model-selection procedures, such as stepwise regression, ridge, or lasso, have to be analysed with caution. The reason is because *we are using the sample twice*: one for selecting the most significant / informative predictors in order to be included in the model, and other for making inference using the same sample. While making this, we are biasing the significance tests, and thus obtaining **unrealistically small p-values**. In other words, when included in the model, some selected predictors will be shown as significant when in reality they are not.

A simple solution for performing valid inference in a data-driven selected model is to split the dataset in two parts: one part for model-selection, and other for performing inference on the coefficients.

The next simulation exercise exemplifies the previous remark. Consider the following linear model

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \varepsilon, \quad (4.5)$$

where  $\beta_1 = \beta_2 = 1$ ,  $\beta_3 = \beta_4 = 0$ , and  $\varepsilon \sim \mathcal{N}(0, 1)$ . The next chunk of code analyses the significances of the four coefficients for:

1. **The model with all the predictors.** The inferences for the coefficients are correct: the distribution of the *p*-values (`pvalues1`) is uniform whenever  $H_0 : \beta_j = 0$  holds (for  $j = 3, 4$ ) and concentrated around 0 when  $H_0$  does not hold (for  $j = 1, 2$ ).
2. **The model with selected predictors by stepwise regression.** The inferences for the coefficients are biased: when  $X_3$  and  $X_4$  are included in the model is because they are highly significant for the given sample by mere chance. Therefore, the distribution of the *p*-values (`pvalues2`) is not uniform but concentrated at 0.
3. **The model with selected predictors by stepwise regression, but fitted in a separate dataset.** In this case, the *p*-values (`pvalues3`) are not unrealistically small if the non-significant predictors are included in the model.

```
# Simulation setting
n <- 2e2
p <- 4
p0 <- p %/% 2
beta <- c(rep(1, p0), rep(0, p - p0))

# Generate two sets of independent data following the same linear model
# with coefficients beta and null intercept
x1 <- matrix(rnorm(n * p), nrow = n, ncol = p)
data1 <- data.frame("x" = x1)
```

```

xbeta1 <- x1 %*% beta
x2 <- matrix(rnorm(n * p), nrow = n, ncol = p)
data2 <- data.frame("x" = x2)
xbeta2 <- x2 %*% beta

# Objects for the simulation
M <- 1e4
pvalues1 <- pvalues2 <- pvalues3 <- matrix(NA, nrow = M, ncol = p)
set.seed(12345678)
data1$y <- xbeta1 + rnorm(n)
nam <- names(lm(y ~ 0 + ., data = data1)$coefficients)

# Simulation
# pb <- txtProgressBar(style = 3)
for (i in 1:M) {

  # Generate new data
  data1$y <- xbeta1 + rnorm(n)

  # Obtain the significances of the coefficients for the usual linear model
  mod1 <- lm(y ~ 0 + ., data = data1)
  s1 <- summary(mod1)
  pvalues1[i, ] <- s1$coefficients[, 4]

  # Obtain the significances of the coefficients for a data-driven selected
  # linear model (in this case, by stepwise regression using BIC)
  mod2 <- stepAIC(mod1, k = log(n), trace = 0)
  s2 <- summary(mod2)
  ind <- match(x = names(s2$coefficients[, 4]), table = nam)
  pvalues2[i, ind] <- s2$coefficients[, 4]

  # Generate independent data
  data2$y <- xbeta2 + rnorm(n)

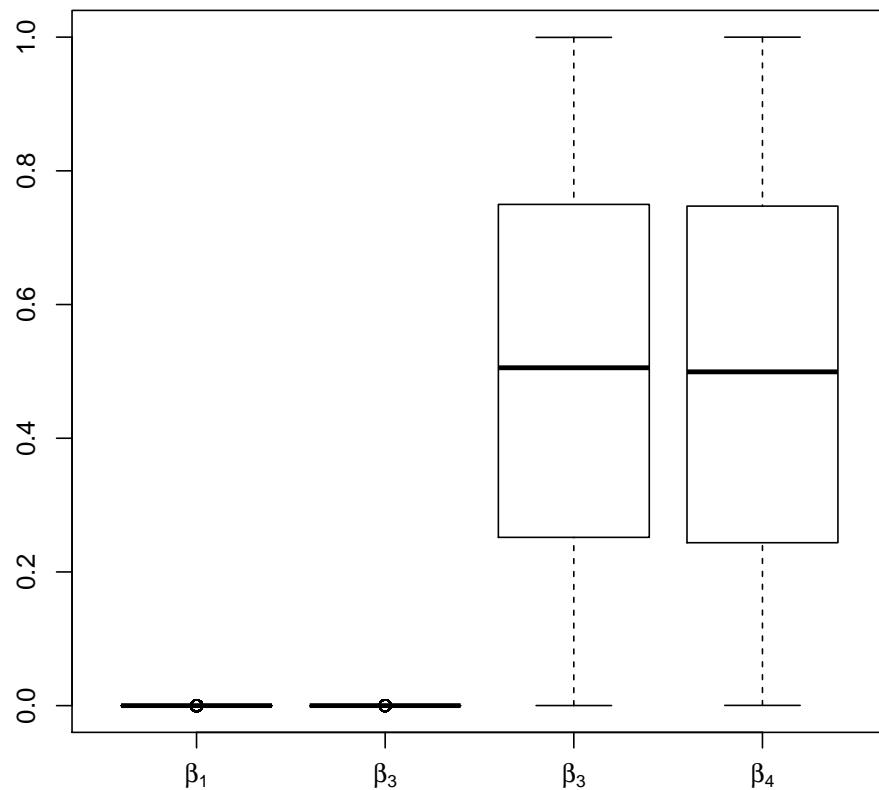
  # Significances of the coefficients by the data-driven selected model
  s3 <- summary(lm(y ~ 0 + ., data = data2[, c(ind, p + 1)]))
  pvalues3[i, ind] <- s3$coefficients[, 4]

  # Progress
  # setTxtProgressBar(pb = pb, value = i / M)
}

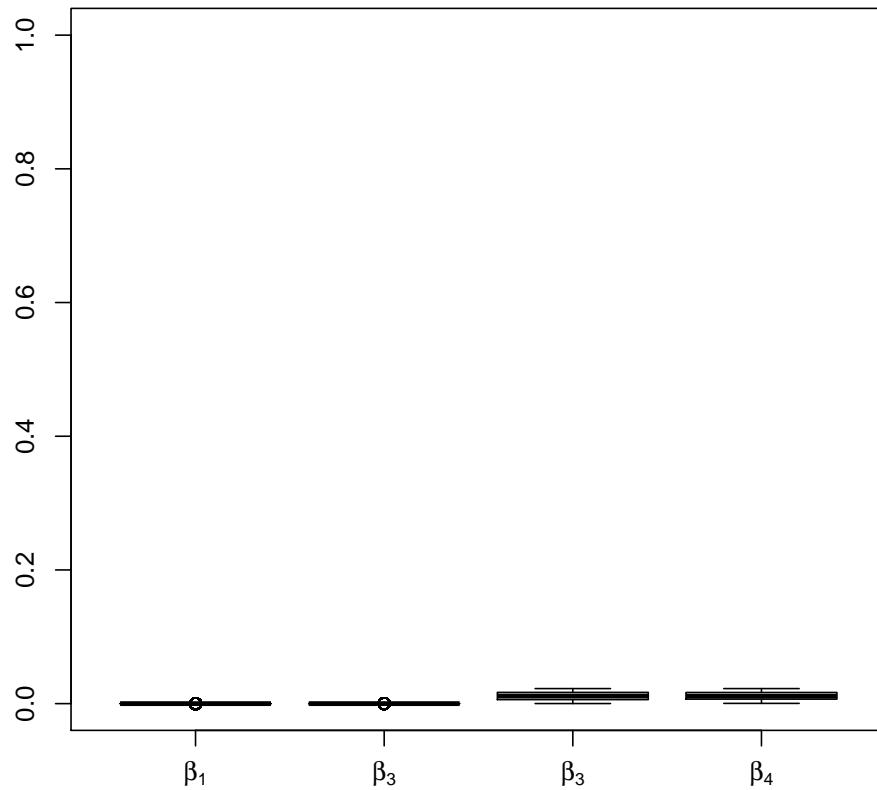
# Percentage of NA's: NA = predictor excluded
apply(pvalues2, 2, function(x) mean(is.na(x)))
## [1] 0.0000 0.0000 0.9792 0.9785

# Boxplots of significances
boxplot(pvalues1, names = expression(beta[1], beta[3], beta[3], beta[4]),
        main = "p-values in the full model", ylim = c(0, 1))

```

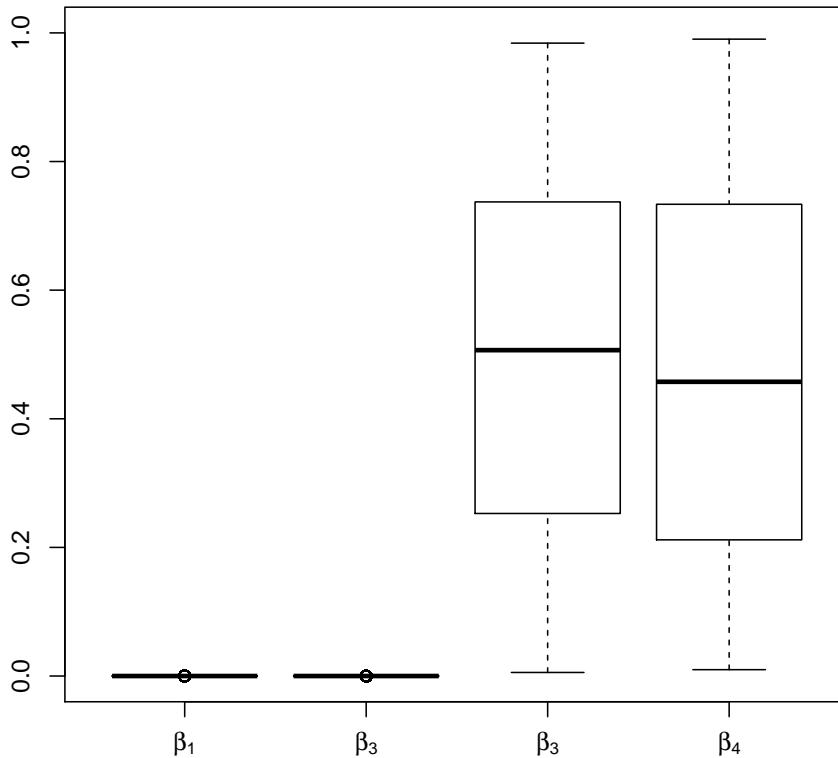
**p-values in the full model**

```
boxplot(pvalues2, names = expression(beta[1], beta[3], beta[3], beta[4]),
        main = "p-values in the stepwise model", ylim = c(0, 1))
```

**p-values in the stepwise model**

```
boxplot(pvalues3, names = expression(beta[1], beta[3], beta[3], beta[4]),
        main = "p-values in the model with the predictors selected by
        stepwise regression, and fitted in an independent sample",
        ylim = c(0, 1))
```

**p-values in the model with the predictors selected by stepwise regression, and fitted in an independent sample**



```
# Test uniformity of the p-values associated to the coefficients that are 0
apply(pvalues1[, (p0 + 1):p], 2, function(x) ks.test(x, y = "punif")$p.value)
## [1] 0.2384572 0.4398724
apply(pvalues2[, (p0 + 1):p], 2, function(x) ks.test(x, y = "punif")$p.value)
## [1] 0 0
apply(pvalues3[, (p0 + 1):p], 2, function(x) ks.test(x, y = "punif")$p.value)
## [1] 0.9056299 0.1379287
```

## 4.3 Big data considerations

The computation of the least squares estimator

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \quad (4.6)$$

involves inverting the  $(p + 1) \times (p + 1)$  matrix  $\mathbf{X}'\mathbf{X}$ , where  $\mathbf{X}$  is an  $n \times (p + 1)$  matrix. The vector to be obtained,  $\hat{\beta}$ , is of size  $p + 1$ . However, computing it directly from (4.6) requires allocating  $\mathcal{O}(np + p^2)$  elements in memory. When  $n$  is very large, this can be prohibitive. In addition, for convenience of the statistical analysis, R's `lm` returns several objects of the same size as  $\mathbf{X}$  and  $\mathbf{Y}$ , thus notably increasing the memory usage. For these reasons, alternative approaches for computing  $\hat{\beta}$  with big data are required.

An alternative for computing (4.6) in a memory-friendly way is to split the computation of  $(\mathbf{X}'\mathbf{X})^{-1}$  and  $\mathbf{X}'\mathbf{Y}$  by *blocks* that are storables in memory. A possibility is to *update* sequentially the estimation of the vector of coefficients. This can be done with the following expression, which relates  $\hat{\beta}$  with  $\hat{\beta}_{-i}$ , the vector of estimated coefficients *without* the  $i$ -th datum:

$$\hat{\beta} = \hat{\beta}_{-i} + (\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_i \left( Y_i - \mathbf{x}'_i \hat{\beta}_{-i} \right). \quad (4.7)$$

In (4.7) above,  $\mathbf{x}'_i$  is the  $i$ -th row of the design matrix  $\mathbf{X}$  and  $\mathbf{X}_{-i}$  is  $\mathbf{X}$  but with the  $i$ -th row removed. The expression follows from the Sherman-Morrison formula for an invertible matrix  $\mathbf{A}$  and a vector  $\mathbf{b}$ ,

$$(\mathbf{A} + \mathbf{b}\mathbf{b}')^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{b}\mathbf{b}'\mathbf{A}^{-1}}{1 + \mathbf{b}'\mathbf{A}^{-1}\mathbf{b}},$$

and from the equalities

$$\begin{aligned} \mathbf{X}'\mathbf{X} &= \mathbf{X}'_{-i}\mathbf{X}_{-i} + \mathbf{x}_i\mathbf{x}'_i, \\ \mathbf{X}'\mathbf{Y} &= \mathbf{X}'_{-i}\mathbf{Y}_{-i} + \mathbf{x}_i Y'_i. \end{aligned}$$

In (4.7), using again the Sherman-Morrison formula we can update  $(\mathbf{X}'\mathbf{X})^{-1}$  easily from  $(\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1}$ :

$$(\mathbf{X}'\mathbf{X})^{-1} = (\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1} - \frac{(\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1}\mathbf{x}_{-i}\mathbf{x}'_{-i}(\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1}}{1 + \mathbf{x}'_{-i}(\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1}\mathbf{x}_{-i}}. \quad (4.8)$$

This has the advantage of not requiring to compute  $\mathbf{X}'\mathbf{X}$  and then to invert it. Instead of that, we work directly with  $(\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1}$ , which was already computed and has size  $(p+1) \times (p+1)$ .

This idea can be iterated and we can compute  $\hat{\beta}$  by the following iterative procedure:

1. Start from a reduced dataset  $\mathbf{X}_{\text{old}} \equiv \mathbf{X}_{-i}$  and  $\mathbf{Y}_{\text{old}} \equiv \mathbf{Y}_{-i}$  for which the least squares estimate can be computed. Denote it by  $\hat{\beta}_{\text{old}} \equiv \hat{\beta}_{-i}$ .
2. Add one of the remaining data points to get  $\hat{\beta}_{\text{new}} \equiv \hat{\beta}$  from (4.7) and (4.8).
3. Store  $\hat{\beta}_{\text{new}}$  and  $\mathbf{X}_{\text{new}}$  in  $\hat{\beta}_{\text{old}}$  and  $\mathbf{X}_{\text{old}}$ , respectively.
4. Repeat steps 2–3 until there are no remaining data points left.
5. Return  $\hat{\beta}_{\text{new}}$  as  $\hat{\beta}$ .

The main advantage of this iterative procedure is clear: **we do not need to store any vector or matrix with  $n$  in the dimension** – only matrices of size  $p$ . As a consequence, we do not need to store the data in memory.

A similar iterative approach (yet more sophisticated) is followed by the `biglm` package. We omit the details here (see Miller (1992)) and just comment the main idea: for computing (4.6), `biglm` performs a QR decomposition<sup>1</sup> of  $\mathbf{X}$  that is computed iteratively. Then, instead of computing (4.6), it solves the triangular system

$$\mathbf{R}\hat{\beta} = \mathbf{Q}^T\mathbf{Y}.$$

Let's see how `biglm` works in practice.

```
# Not really "big data", but for the sake of illustration
set.seed(12345)
n <- 1e6
```

<sup>1</sup>The QR decomposition of the matrix  $\mathbf{X}$  of size  $n \times m$  is  $\mathbf{X} = \mathbf{QR}$  such that  $\mathbf{Q}$  is an  $n \times n$  orthogonal matrix and  $\mathbf{R}$  is an  $n \times m$  upper triangular matrix. This factorization is commonly used in numerical analysis for solving linear systems.

```

p <- 10
beta <- seq(-1, 1, length.out = p)^5
x1 <- matrix(rnorm(n * p), nrow = n, ncol = p)
x1[, p] <- 2 * x1[, 1] + rnorm(n, sd = 0.1) # Add some dependence to predictors
x1[, p - 1] <- 2 - x1[, 2] + rnorm(n, sd = 0.5)
y1 <- 1 + x1 %*% beta + rnorm(n)
x2 <- matrix(rnorm(100 * p), nrow = 100, ncol = p)
y2 <- 1 + x2 %*% beta + rnorm(100)
bigData1 <- data.frame("resp" = y1, "pred" = x1)
bigData2 <- data.frame("resp" = y2, "pred" = x2)

# biglm has a very similar syntax to lm - but the formula interface does not
# work always as expected
library(bigm)
# biglm(formula = resp ~ ., data = bigData1) # Does not work
# biglm(formula = y ~ x) # Does not work
# biglm(formula = resp ~ pred.1 + pred.2, data = bigData1) # Does work, but
# not very convenient for a large number of predictors
# Hack for automatic inclusion of all the predictors
f <- formula(paste("resp ~", paste(names(bigData1)[-1], collapse = " + ")))
biglmMod <- biglm(formula = f, data = bigData1)

# lm's call
lmMod <- lm(formula = resp ~ ., data = bigData1)

# The reduction in size of the resulting object is more than notable
print(object.size(bigmMod), units = "Kb")
## 12.1 Kb
print(object.size(lmMod), units = "Mb")
## 358.6 Mb

# Summaries
s1 <- summary(bigmMod)
s2 <- summary(lmMod)
s1
## Large data regression model: biglm(formula = f, data = bigData1)
## Sample size = 1000000
##          Coef    (95%      CI)      SE      p
## (Intercept) 1.0021  0.9939  1.0104 0.0041 0.0000
## pred.1      -0.9733 -1.0133 -0.9333 0.0200 0.0000
## pred.2      -0.2866 -0.2911 -0.2822 0.0022 0.0000
## pred.3      -0.0535 -0.0555 -0.0515 0.0010 0.0000
## pred.4      -0.0041 -0.0061 -0.0021 0.0010 0.0000
## pred.5      -0.0002 -0.0022  0.0018 0.0010 0.8373
## pred.6       0.0003 -0.0017  0.0023 0.0010 0.7771
## pred.7       0.0026  0.0006  0.0046 0.0010 0.0091
## pred.8       0.0521  0.0501  0.0541 0.0010 0.0000
## pred.9       0.2840  0.2800  0.2880 0.0020 0.0000
## pred.10      0.9867  0.9667  1.0067 0.0100 0.0000
s2
##
## Call:
## lm(formula = resp ~ ., data = bigData1)

```

```

## 
## Residuals:
##   Min     1Q  Median     3Q    Max 
## -4.8798 -0.6735 -0.0013  0.6735  4.9060 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 1.0021454 0.0041200 243.236 < 2e-16 ***
## pred.1      -0.9732675 0.0199989 -48.666 < 2e-16 ***
## pred.2      -0.2866314 0.0022354 -128.227 < 2e-16 ***
## pred.3      -0.0534834 0.0009997 -53.500 < 2e-16 ***
## pred.4      -0.0040772 0.0009984 -4.084 4.43e-05 ***
## pred.5      -0.0002051 0.0009990 -0.205  0.83731  
## pred.6       0.0002828 0.0009989  0.283  0.77706  
## pred.7       0.0026085 0.0009996  2.610  0.00907 ** 
## pred.8       0.0520744 0.0009994  52.105 < 2e-16 ***
## pred.9       0.2840358 0.0019992 142.076 < 2e-16 ***
## pred.10      0.9866851 0.0099876  98.791 < 2e-16 ***
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.9993 on 999989 degrees of freedom 
## Multiple R-squared:  0.5777, Adjusted R-squared:  0.5777 
## F-statistic: 1.368e+05 on 10 and 999989 DF,  p-value: < 2.2e-16

# Further information
s1$mat # Coefficients and their inferences
##             Coef      (95% CI)        SE
## (Intercept) 1.0021454430 0.9939053491 1.010385537 0.0041200470
## pred.1      -0.9732674585 -1.0132653005 -0.933269616 0.0199989210
## pred.2      -0.2866314070 -0.2911021089 -0.282160705 0.0022353509
## pred.3      -0.0534833941 -0.0554827653 -0.051484023 0.0009996856
## pred.4      -0.0040771777 -0.0060739907 -0.002080365 0.0009984065
## pred.5      -0.0002051218 -0.0022030377  0.001792794 0.0009989579
## pred.6       0.0002828388 -0.0017149118  0.002280589 0.0009988753
## pred.7       0.0026085425  0.0006093153  0.004607770 0.0009996136
## pred.8       0.0520743791  0.0500755376  0.054073221 0.0009994208
## pred.9       0.2840358104  0.2800374345  0.288034186 0.0019991879
## pred.10      0.9866850849  0.9667099026  1.006660267 0.0099875911
##             p
## (Intercept) 0.000000e+00
## pred.1      0.000000e+00
## pred.2      0.000000e+00
## pred.3      0.000000e+00
## pred.4      4.432709e-05
## pred.5      8.373098e-01
## pred.6      7.770563e-01
## pred.7      9.066118e-03
## pred.8      0.000000e+00
## pred.9      0.000000e+00
## pred.10     0.000000e+00
s1$rsq # R^2
## [1] 0.5777074

```

```

s1$nullrss # SST (as in Section 2.6)
## [1] 2364861

# Extract coefficients
coef(biglmMod)
## (Intercept)      pred.1      pred.2      pred.3      pred.4
## 1.0021454430 -0.9732674585 -0.2866314070 -0.0534833941 -0.0040771777
##      pred.5      pred.6      pred.7      pred.8      pred.9
## -0.0002051218  0.0002828388  0.0026085425  0.0520743791  0.2840358104
##      pred.10
##  0.9866850849

# Prediction works as usual
predict(biglmMod, newdata = bigData2[1:5, ])
##      [,1]
## 1 2.3554732
## 2 2.5631387
## 3 2.4546594
## 4 2.3483083
## 5 0.6587481

# Must contain a column for the response
# predict(biglmMod, newdata = bigData2[1:5, -1]) # Error

# Update the model with training data
update(biglmMod, moredata = bigData2)
## Large data regression model: biglm(formula = f, data = bigData1)
## Sample size = 1000100

# AIC and BIC
AIC(biglmMod, k = 2)
## [1] 998685.1
AIC(biglmMod, k = log(n))
## [1] 998815.1

# Features not immediately available for biglm objects: stepwise selection by
# stepAIC, residuals, variance of the error, model diagnostics, and vifs

# Workaround for obtaining hat(sigma)^2 = SSE / (n - p - 1), SSE = SST * (1 - R^2)
(s1$nullrss * (1 - s1$rsq)) / s1$obj$df.resid
## [1] 0.9986741
s2$sigma^2
## [1] 0.9986741

```

Model selection of `biglm` models can be done, not by `stepAIC`, but with the more advanced `leaps` package. This is achieved by the `regsubsets` function, which returns the *best subset* of up to (by default) `nvmax = 8` predictors among the  $p$  possible predictors to be included in the model. The function requires the *full* `biglm` model to begin the *exhaustive*<sup>2</sup> search (Furnival and Wilson, 1974). The kind of search can be changed using the `method` argument and choosing the `exhaustive` (by default), `forward`, or `backward` selection.

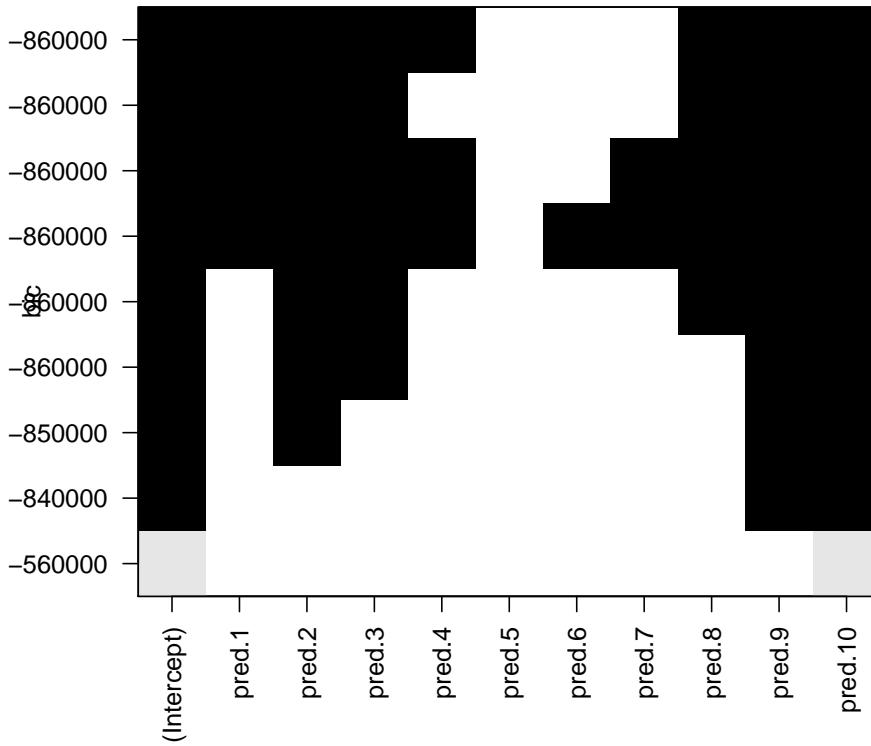
```

# Model selection adapted to big data models
library(leaps)

```

<sup>2</sup>Not really exhaustive: the method behind it, due to Furnival and Wilson (1974), employs an ingenious branch-and-bound algorithm to remove most of the non-interesting models.

```
reg <- regsubsets(biglmMod, nvmax = p, method = "exhaustive")
plot(reg) # Plot best model (top row) to worst model (bottom row)
```



```
# Summarize (otherwise regsubsets's output is hard to decypher)
subs <- summary(reg)
subs
## Subset selection object
## 10 Variables (and intercept)
##      Forced in Forced out
## pred.1      FALSE      FALSE
## pred.2      FALSE      FALSE
## pred.3      FALSE      FALSE
## pred.4      FALSE      FALSE
## pred.5      FALSE      FALSE
## pred.6      FALSE      FALSE
## pred.7      FALSE      FALSE
## pred.8      FALSE      FALSE
## pred.9      FALSE      FALSE
## pred.10     FALSE      FALSE
## 1 subsets of each size up to 9
## Selection Algorithm: exhaustive
##      pred.1 pred.2 pred.3 pred.4 pred.5 pred.6 pred.7 pred.8 pred.9
## 1 ( 1 ) " " " " " " " " " "
## 2 ( 1 ) " " " " " " " " " "
## 3 ( 1 ) " " "*" " " " " " " "
## 4 ( 1 ) " " "*" "*" " " " " " "
## 5 ( 1 ) " " "*" "*" " " " " " "
## 6 ( 1 ) "*" "*" "*" "*" " " " " "
## 7 ( 1 ) "*" "*" "*" "*" " " " " "
```

```

## 8  ( 1 ) "*"  "*"  "*"  "*"  " "  " "  "*"  "*"  "*"
## 9  ( 1 ) "*"  "*"  "*"  "*"  " "  " "  "*"  "*"  "*"
##          pred.10
## 1  ( 1 ) "*"
## 2  ( 1 ) "*"
## 3  ( 1 ) "*"
## 4  ( 1 ) "*"
## 5  ( 1 ) "*"
## 6  ( 1 ) "*"
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
## 9  ( 1 ) "*"

# Lots of useful information
str(subs, 1)
## List of 8
##  $ which : logi [1:9, 1:11] TRUE TRUE TRUE TRUE TRUE TRUE ...
##  ..- attr(*, "dimnames")=List of 2
##  $ rsq  : num [1:9] 0.428 0.567 0.574 0.576 0.577 ...
##  $ rss  : num [1:9] 1352680 1023080 1006623 1003763 1001051 ...
##  $ adjr2 : num [1:9] 0.428 0.567 0.574 0.576 0.577 ...
##  $ cp   : num [1:9] 354480 244444 7968 5106 2392 ...
##  $ bic  : num [1:9] -558604 -837860 -854062 -856894 -859585 ...
##  $ outmat: chr [1:9, 1:10] " " " " " " " ...
##  ..- attr(*, "dimnames")=List of 2
##  $ obj   :List of 27
##  ..- attr(*, "class")= chr "regsubsets"
##  - attr(*, "class")= chr "summary.regsubsets"

# Get the model with lowest BIC
subs$which
## (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5 pred.6 pred.7 pred.8
## 1      TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
## 2      TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
## 3      TRUE  FALSE  TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
## 4      TRUE  FALSE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  FALSE
## 5      TRUE  FALSE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  TRUE
## 6      TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  TRUE
## 7      TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  TRUE
## 8      TRUE  TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  TRUE  TRUE
## 9      TRUE  TRUE  TRUE  TRUE  TRUE  FALSE  TRUE  TRUE  TRUE
## pred.9 pred.10
## 1 FALSE  TRUE
## 2 TRUE  TRUE
## 3 TRUE  TRUE
## 4 TRUE  TRUE
## 5 TRUE  TRUE
## 6 TRUE  TRUE
## 7 TRUE  TRUE
## 8 TRUE  TRUE
## 9 TRUE  TRUE
subs$bic
## [1] -558603.7 -837859.9 -854062.3 -856893.8 -859585.3 -861936.5 -861939.3

```

```

## [8] -861932.3 -861918.6
subs$which[which.min(subs$bic), ]
## (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5
## TRUE TRUE TRUE TRUE TRUE FALSE
## pred.6 pred.7 pred.8 pred.9 pred.10
## FALSE FALSE TRUE TRUE TRUE

# It also works with ordinary linear models and it is much faster and
# informative than stepAIC
reg <- regsubsets(resp ~ ., data = bigData1, nvmax = p,
                   method = "backward")
subs$bic
## [1] -558603.7 -837859.9 -854062.3 -856893.8 -859585.3 -861936.5 -861939.3
## [8] -861932.3 -861918.6
subs$which[which.min(subs$bic), ]
## (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5
## TRUE TRUE TRUE TRUE TRUE FALSE
## pred.6 pred.7 pred.8 pred.9 pred.10
## FALSE FALSE TRUE TRUE TRUE

# Compare it with stepAIC
stepAIC(lm(resp ~ ., data = bigData1), trace = 0,
        direction = "backward", k = log(n))
##
## Call:
## lm(formula = resp ~ pred.1 + pred.2 + pred.3 + pred.4 + pred.8 +
##     pred.9 + pred.10, data = bigData1)
##
## Coefficients:
## (Intercept) pred.1 pred.2 pred.3 pred.4
## 1.002141 -0.973201 -0.286626 -0.053487 -0.004074
## pred.8 pred.9 pred.10
## 0.052076 0.284038 0.986651

```

Finally, let's see an example on how to fit a linear model to a large dataset that does not fit in the RAM of most regular laptops. Imagine that you want to regress a response  $Y$  into a set of  $p = 10$  predictors and the sample size is  $n = 10^8$ . Merely storing the response and the predictors will take up to 8.2 Gb in RAM:

```

# Size of the response
print(object.size(rnorm(1e6)) * 1e2, units = "Gb")
## 0.7 Gb

# Size of the predictors
print(object.size(rnorm(1e6)) * 1e2 * 10, units = "Gb")
## 7.5 Gb

```

In addition to this, if `lm` was called, it will return the `residuals`, `effects`, and `fitted.values` slots (all vectors of length  $n$ , hence  $0.7 \times 3 = 2.1$  Gb more). It will also return the `qr` decomposition of the design matrix and the model matrix (both are  $n \times (p+1)$  matrices, so another  $8.2 \times 2 = 14.4$  Gb more). The final `lm` object will thus be at the very least, size 16.5 Gb. Clearly, this is not a very memory-friendly way of proceeding.

A possible approach is to split the dataset and perform updates of the model in chunks of reasonable size. The next code provides a template for such approach using `biglm` and `update`.

```

# Linear regression with n = 10^8 and p = 10
n <- 10^8
p <- 10
beta <- seq(-1, 1, length.out = p)^5

# Number of chunks for splitting the dataset
nChunks <- 1e3
nSmall <- n / nChunks

# Simulates reading the first chunk of data
set.seed(12345)
x <- matrix(rnorm(nSmall * p), nrow = nSmall, ncol = p)
x[, p] <- 2 * x[, 1] + rnorm(nSmall, sd = 0.1)
x[, p - 1] <- 2 - x[, 2] + rnorm(nSmall, sd = 0.5)
y <- 1 + x %*% beta + rnorm(nSmall)

# First fit
bigMod <- biglm(y ~ x, data = data.frame(y, x))

# Update fit
# pb <- txtProgressBar(style = 3)
for (i in 2:nChunks) {

  # Simulates reading the i-th chunk of data
  set.seed(12345 + i)
  x <- matrix(rnorm(nSmall * p), nrow = nSmall, ncol = p)
  x[, p] <- 2 * x[, 1] + rnorm(nSmall, sd = 0.1)
  x[, p - 1] <- 2 - x[, 2] + rnorm(nSmall, sd = 0.5)
  y <- 1 + x %*% beta + rnorm(nSmall)

  # Update the fit
  bigMod <- update(bigMod, moredata = data.frame(y, x))

  # Progress
  # setTxtProgressBar(pb = pb, value = i / nChunks)
}

# Final model
summary(bigMod)
## Large data regression model: biglm(y ~ x, data = data.frame(y, x))
## Sample size = 100000000
##          Coef      (95%      CI)      SE      p
## (Intercept) 1.0003  0.9995  1.0011 4e-04 0.0000
## x1          -1.0015 -1.0055 -0.9975 2e-03 0.0000
## x2          -0.2847 -0.2852 -0.2843 2e-04 0.0000
## x3          -0.0531 -0.0533 -0.0529 1e-04 0.0000
## x4          -0.0041 -0.0043 -0.0039 1e-04 0.0000
## x5           0.0002  0.0000  0.0004 1e-04 0.0760
## x6          -0.0001 -0.0003  0.0001 1e-04 0.2201
## x7           0.0041  0.0039  0.0043 1e-04 0.0000
## x8           0.0529  0.0527  0.0531 1e-04 0.0000
## x9           0.2844  0.2840  0.2848 2e-04 0.0000

```

```
## x10      1.0007  0.9987  1.0027 1e-03 0.0000
print(object.size(bigMod), units = "Kb")
## 7.1 Kb
```



The `summary` of a `biglm` object yields slightly different significances for the coefficients than for `lm`. The reason is that `biglm` employs  $\mathcal{N}(0, 1)$  approximations for the distributions of the  $t$ -tests instead of the *exact*  $t_{n-1}$  distribution. Obviously, if  $n$  is large, the differences are inappreciable.

# Chapter 5

## Generalized linear models

As we saw in Chapter 2, linear regression assumes that the response variable  $Y$  is such that

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim \mathcal{N}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p, \sigma^2)$$

and hence

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p.$$

This, in particular, implies that  $Y$  is *continuous*. In this chapter we will see how *generalized linear models* can deal with other kinds of distributions for  $Y|(X_1 = x_1, \dots, X_p = x_p)$ , particularly with *discrete* responses, by modelling the *transformed* conditional expectation. The simplest generalized linear model is *logistic regression*, which arises when  $Y$  is a *binary* response, that is, a variable encoding two categories with 0 and 1. This model would be useful, for example, to predict  $Y$  given  $X$  from a sample  $(X_1, Y_1), \dots, (X_n, Y_n)$  like the one in Figure 5.1.

### 5.1 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 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 in order to investigate the causes of 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 unusually cold temperature ( $-0.6$  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 problem 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 5.2a, was:

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

The Rogers Commission noted a major flaw in Figure 5.2a: 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 5.2b). The Rogers Commission concluded:

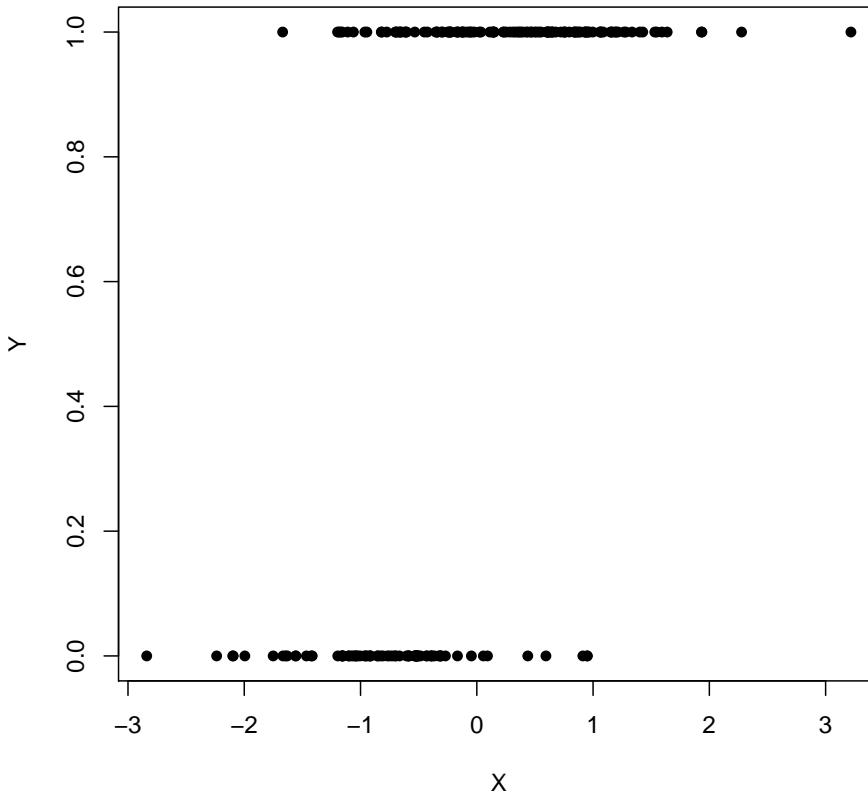


Figure 5.1: Scatterplot of a sample  $(X_1, Y_1), \dots, (X_n, Y_n)$  sampled from a logistic regression.

“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 (download). The dataset contains (shown in Table 5.1) 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:

- `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.

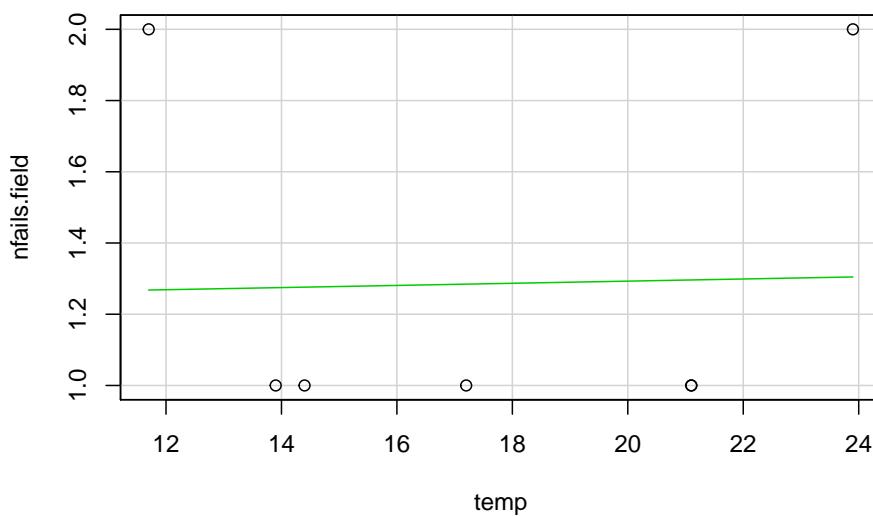
<sup>1</sup> After the shuttle exits the atmosphere, the solid rocket boosters separate and descend to land using a parachute where they are carefully analyzed.

Table 5.1: 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 5.2a and 5.2b and checking that linear regression is not the right tool for answering Q1–Q3.

```
library(car)
scatterplot(nfails.field ~ temp, reg.line = lm, smooth = FALSE, spread = FALSE,
            boxplots = FALSE, data = challenger, subset = nfail.field > 0)
```



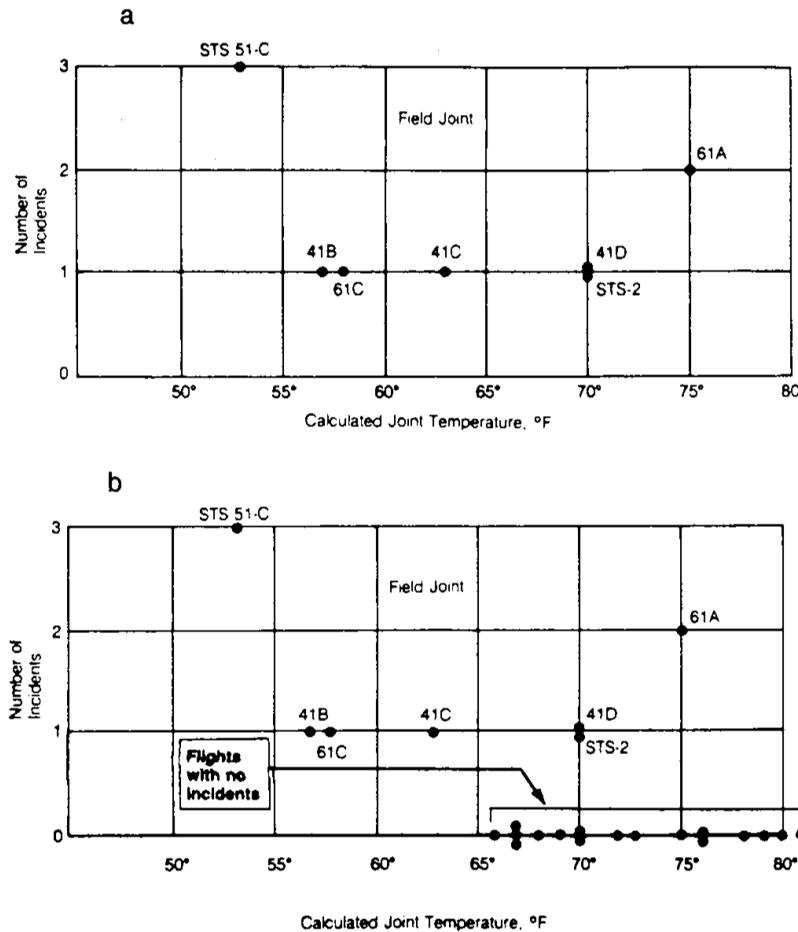
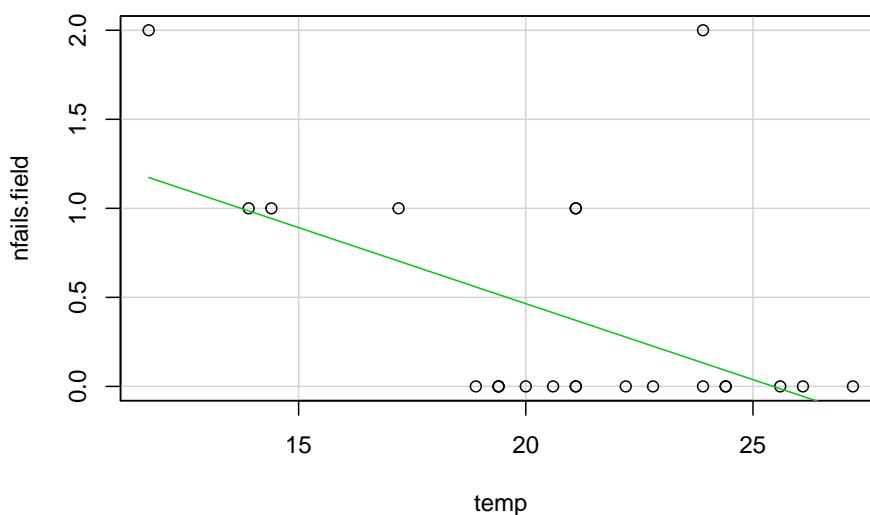


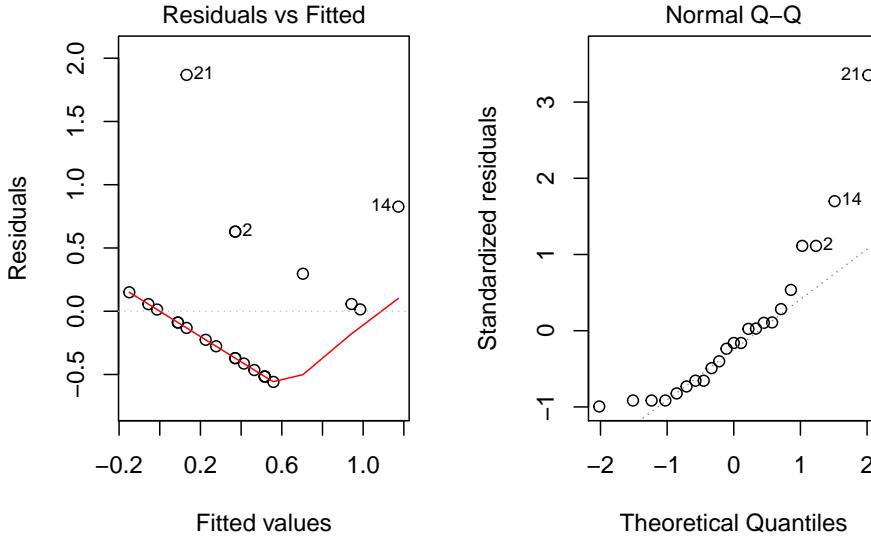
Figure 5.2: Number of incidents in the O-rings (field joints) versus temperatures. Panel *a* includes *only flights with incidents*. Panel *b* contains all flights (with and *without* incidents).

```
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). Let's check this with the corresponding diagnostic plots:

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



Albeit 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 5.2a). This was one of the arguments behind NASA's decision of launching the rocket at a temperature of  $-0.6$  Celsius degrees.
2. However, **the trend for all launches indicates a clear negative dependence between temperature and number of incidents!** (Figure 5.2b). Think about it in this way: the minimum temperature for a launch without incidents ever recorded was above  $18$  Celsius degrees, and the Challenger was launched at  $-0.6$  without clearly knowing the effects of such low temperatures.

## 5.2 Model formulation and estimation

For simplicity, we first study the logistic regression and then study the general case of a generalized linear model.

### 5.2.1 Logistic regression

As we saw in Section 2.2, the multiple linear model described the relation between the random variables  $X_1, \dots, X_p$  and  $Y$  by assuming a linear relation in the conditional expectation:

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p. \quad (5.1)$$

In addition, it made three more assumptions on the data (see Section 2.3) which resulted in the following

one-line summary of the linear model:

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim \mathcal{N}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p, \sigma^2).$$

Recall that a necessary condition is that  $Y$  was continuous, in order to satisfy the normality of the errors. Therefore, the linear model is designed for a *continuous response*.

The situation when  $Y$  is *discrete* (naturally ordered values) or *categorical* (non-ordered categories) requires a special treatment. The simplest situation is when  $Y$  is *binary*: it can only take two values, codified for convenience as 1 (success) and 0 (failure). For binary variables there is no fundamental distinction between the treatment of discrete and categorical variables. Formally, a binary variable is referred as a *Bernoulli variable*:  $Y \sim \text{Ber}(p)$ ,  $0 \leq p \leq 1$  (do not confuse this  $p$  with the  $p$  providing the number of predictors in the model), if

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

or, equivalently, if

$$\mathbb{P}[Y = y] = p^y (1 - p)^{1-y}, \quad y = 0, 1. \quad (5.2)$$

Recall that a *binomial variable with size  $n$  and probability  $p$* ,  $\text{B}(n, p)$ , is obtained by summing  $n$  independent  $\text{Ber}(p)$  (so  $\text{Ber}(p)$  is the same as  $\text{B}(1, p)$ ). Also, recall that a Bernoulli variable is completely determined by the probability  $p$ , and also so do its mean and variance:

$$\mathbb{E}[Y] = \mathbb{P}[Y = 1] = p \quad \text{and} \quad \text{Var}[Y] = p(1 - p).$$

Assume then that  $Y$  is a Bernoulli variable and that  $X_1, \dots, X_p$  are predictors associated to  $Y$ . The purpose in logistic regression is to model

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \mathbb{P}[Y = 1|X_1 = x_1, \dots, X_p = x_p],$$

that is, how the conditional expectation of  $Y$  or, equivalently, the conditional probability of  $Y = 1$ , is changing according to particular values of the predictors. At sight of (5.1), a tempting possibility is to consider the model

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p =: \eta.$$

However, such a model will run into serious problems inevitably: negative probabilities and probabilities larger than one may happen. A solution is to consider a **link function**  $g$  to encapsulate the value of  $\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]$  and map it back to  $\mathbb{R}$ . Or, alternatively, a function  $g^{-1}$  that takes  $\eta \in \mathbb{R}$  and maps it to  $[0, 1]$ , the support of  $\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]$ . There are several alternatives for  $g^{-1} : \mathbb{R} \rightarrow [0, 1]$  (see Figure 5.3). Different choices of  $g^{-1}$  give rise to different models:

- **Uniform**: considers the truncation  $\eta \mathbb{1}_{\{0 < \eta < 1\}} + \mathbb{1}_{\{\eta \geq 1\}}$ .
- **Probit**: considers the *normal distribution function*, this is,  $g^{-1} = \Phi$ .
- **Logit**: considers the **logistic distribution function**:

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

The logistic transformation is the most employed due to its tractability, interpretability, and smoothness (and as we will see, later, because it is the *canonical link function*). Its inverse,  $g : [0, 1] \rightarrow \mathbb{R}$ , is known as the **logit function**:

$$\text{logit}(p) := \text{logistic}^{-1}(p) = \log \frac{p}{1 - p}.$$

In conclusion, with the logit link function we can map the domain of  $Y$  to  $\mathbb{R}$  in order to apply a linear model. The *logistic model* can be then equivalently stated as

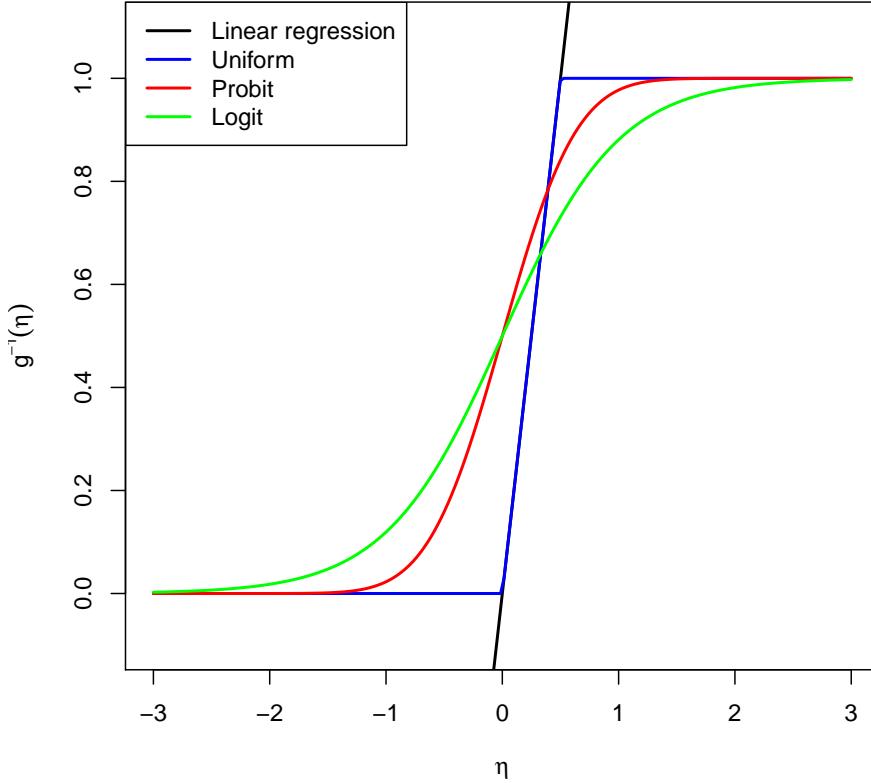


Figure 5.3: Different transformations  $g^{-1}$  mapping the response of a simple linear regression  $\eta = \beta_0 + \beta_1 x$  to  $[0, 1]$ .

$$\text{logit}(\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]) = \eta \quad (5.3)$$

or as

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = \text{logistic}(\eta) = \frac{1}{1 + e^\eta}, \quad (5.4)$$

where remember that  $\eta = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$ . There is a clear interpretation of the role of the linear predictor  $\eta$  in (5.4):

- If  $\eta = 0$ , then  $\mathbb{P}[Y = 1|X_1 = x_1, \dots, X_p = x_p] = \frac{1}{2}$  ( $Y = 1$  and  $Y = 0$  are equally likely).
- If  $\eta < 0$ , then  $\mathbb{P}[Y = 1|X_1 = x_1, \dots, X_p = x_p] < \frac{1}{2}$  ( $Y = 1$  less likely).
- If  $\eta > 0$ , then  $\mathbb{P}[Y = 1|X_1 = x_1, \dots, X_p = x_p] > \frac{1}{2}$  ( $Y = 1$  more likely).

To be more precise on the interpretation of the coefficients we need to introduce the *odds*.

The odds is an equivalent way of expressing the distribution of probabilities in a binary variable  $Y$ . Instead of using  $p$  to characterize the distribution of  $Y$ , we can use

$$\text{odds}(Y) := \frac{p}{1 - p} = \frac{\mathbb{P}[Y = 1]}{\mathbb{P}[Y = 0]}. \quad (5.5)$$

The odds is the *ratio between the probability of success and the probability of failure*. It is extensively used in betting<sup>2</sup> due to its better interpretability. For example, if a horse  $Y$  has a probability  $p = 2/3$  of winning a race ( $Y = 1$ ), then the odds of the horse is  $\frac{p}{1-p} = \frac{2/3}{1/3} = 2$ . This means that the horse has a *probability of winning that is twice larger than the probability of losing*. This is sometimes written as a  $2 : 1$  or  $2 \times 1$  (spelled “two-to-one”). Conversely, if the odds of  $Y$  is given, we can easily know what is the probability of success  $p$ , using the inverse of (5.5):

$$p = \mathbb{P}[Y = 1] = \frac{\text{odds}(Y)}{1 + \text{odds}(Y)}.$$

For example, if the odds of the horse were 5, that would correspond to a probability of winning  $p = 5/6$ .



Recall that the odds is a number in  $[0, +\infty]$ . The 0 and  $+\infty$  values are attained for  $p = 0$  and  $p = 1$ , respectively. The log-odds (or logit) is a number in  $[-\infty, +\infty]$ .

We can rewrite (5.4) (since  $\mathbb{P}[Y = 1|X_1 = x_1, \dots, X_p = x_p] = \mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]$ ) in terms of the odds (5.5). If we do so, we have:

$$\text{odds}(Y|X_1 = x_1, \dots, X_p = x_p) = e^\eta = e^{\beta_0} e^{\beta_1 x_1} \dots e^{\beta_p x_p}. \quad (5.6)$$

Alternatively, taking logarithms, we have the *log-odds* (or logit)

$$\log(\text{odds}(Y|X_1 = x_1, \dots, X_p = x_p)) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p. \quad (5.7)$$

The conditional log-odds (5.7) plays the role of the conditional mean for multiple linear regression. Therefore, we have an analogous interpretation for the coefficients:

- $\beta_0$ : is the log-odds when  $X_1 = \dots = X_p = 0$ .
- $\beta_j$ ,  $1 \leq j \leq p$ : is the **additive** increment of the log-odds for an increment of one unit in  $X_j = x_j$ , provided that the remaining variables  $X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_p$  do not change.

The log-odds is not as easy to interpret as the odds. For that reason, an equivalent way of interpreting the coefficients, this time based on (5.6), is:

- $e^{\beta_0}$ : is the odds when  $X_1 = \dots = X_p = 0$ .
- $e^{\beta_j}$ ,  $1 \leq j \leq p$ : is the **multiplicative** increment of the odds for an increment of one unit in  $X_j = x_j$ , provided that the remaining variables  $X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_p$  do not change. If the increment in  $X_j$  is of  $r$  units, then the multiplicative increment in the odds is  $(e^{\beta_j})^r$ .

As a consequence of this last interpretation, we have:



If  $\beta_j > 0$  (respectively,  $\beta_j < 0$ ) then  $e^{\beta_j} > 1$  ( $e^{\beta_j} < 1$ ) in (5.6). Therefore, an increment of one unit in  $X_j$ , provided that the remaining variables do not change, results in a positive (negative) increment in the odds and in  $\mathbb{P}[Y = 1|X_1 = x_1, \dots, X_p = x_p]$ .

### 5.2.1.1 Case study application

In the Challenger case study we used `fail.field` as an *indicator* of whether “there was at least an incident with the O-rings” (1 = yes, 0 = no). Let’s see if the temperature was associated with O-ring incidents (Q1). For that, we compute the logistic regression of `fail.field` on `temp` and we plot the fitted logistic curve.

<sup>2</sup>Recall that (traditionally) the result of a bet is binary: you either win or lose the bet.

```

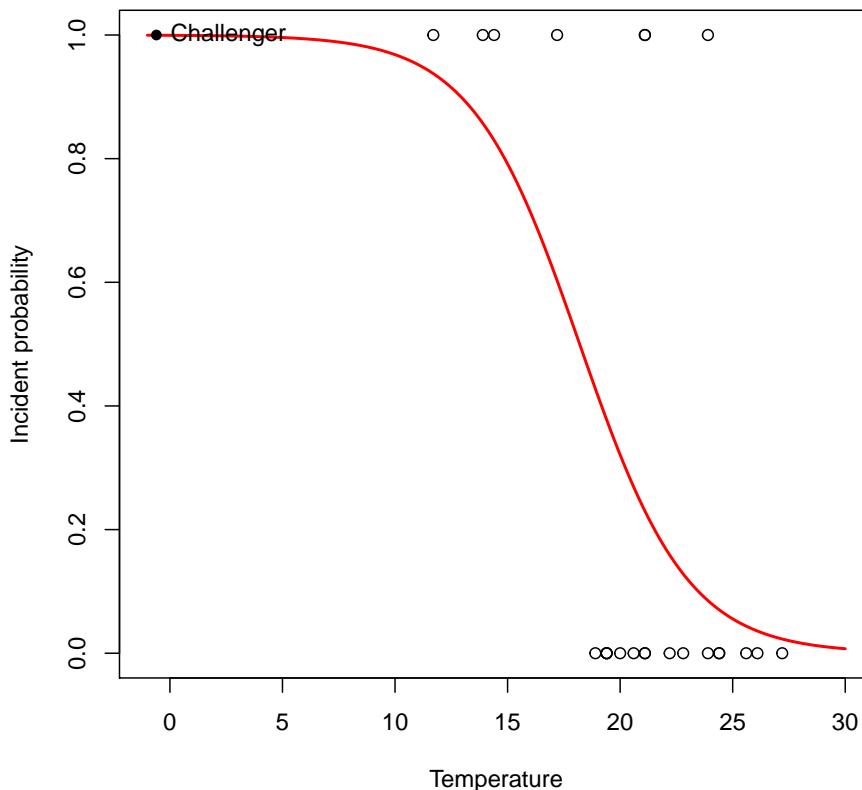
# Logistic regression
nasa <- glm(fail.field ~ temp, family = "binomial", data = challenger)

# 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, 1 = 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 it seems that the temperature was affecting the probability of an O-ring incident (Q1). Let's quantify this statement and answer Q2 by looking to the coefficients of the model:

```

# Exponentiated coefficients ("odds ratios")
exp(coef(nasa))
## (Intercept)      temp
## 1965.9743592   0.6592539

```

The exponentials of the estimated coefficients are:

- $e^{\hat{\beta}_0} = 1965.974$ . This means that, *when the temperature is zero*, the fitted odds is 1965.974, so the probability of having an incident ( $Y = 1$ ) is 1965.974 times larger than the probability of not having

an incident ( $Y = 0$ ). In other words, the probability of having an incident at temperature zero is  $\frac{1965.974}{1965.974+1} = 0.999$ .

- $e^{\hat{\beta}_1} = 0.659$ . This means that each Celsius degree increment in the temperature multiplies the fitted odds by a factor of 0.659  $\approx \frac{2}{3}$ , hence reducing it.

However, for the moment we can not say whether these findings are significant, since we do not have information on the variability of the estimates of  $\beta$ . We will need inference for that.

### 5.2.1.2 Estimation by maximum likelihood

The estimation of  $\beta$  from a sample  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$  is done by *Maximum Likelihood Estimation* (MLE). As it can be seen in Appendix E, MLE is equivalent to least squares in the linear model under the assumptions mentioned in Section 2.3, particularly, normality and independence. In the logistic model, we assume that

$$Y_i | (X_1 = x_{i1}, \dots, X_p = x_{ip}) \sim \text{Ber}(\text{logistic}(\eta_i)), \quad i = 1, \dots, n,$$

where  $\eta_i := \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}$ . Denoting  $p_i(\beta) := \text{logistic}(\eta_i)$ , the likelihood of  $\beta$  is

$$\begin{aligned} \ell(\beta) &= \log \prod_{i=1}^n p_i(\beta)^{Y_i} (1 - p_i(\beta))^{1-Y_i} \\ &= \sum_{i=1}^n [Y_i \log p_i(\beta) + (1 - Y_i) \log(1 - p_i(\beta))]. \end{aligned} \quad (5.8)$$

The MLE estimate of  $\beta$  is

$$\hat{\beta} := \arg \max_{\beta \in \mathbb{R}^{p+1}} \ell(\beta).$$

Unfortunately, due to the non-linearity of (5.8), there is no explicit expression for  $\hat{\beta}$  and it has to be obtained numerically by means of an iterative procedure. We will see it with more detail in the next section. Just be aware that this iterative procedure may fail to converge in low sample size situations with perfect classification, where the likelihood might be numerically unstable.

Figure 5.2.1.2 shows 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’. Application also available here.

The data of the illustration has been generated with the following code:

```
# Data
set.seed(34567)
x <- rnorm(50, sd = 1.5)
y1 <- -0.5 + 3 * x
y2 <- 0.5 - 2 * x
y3 <- -2 + 5 * x
y1 <- rbinom(50, size = 1, prob = 1 / (1 + exp(-y1)))
y2 <- rbinom(50, size = 1, prob = 1 / (1 + exp(-y2)))
y3 <- rbinom(50, size = 1, prob = 1 / (1 + exp(-y3)))

# Data
dataMle <- data.frame(x = x, y1 = y1, y2 = y2, y3 = y3)
```

For fitting a logistic model we employ `glm`, which has the syntax `glm(formula = response ~ predictor, family = "binomial", data = data)`, where `response` is a binary variable. Note that `family = "binomial"` is referring to the fact that the response is Binomial variable (since it is a Bernoulli). Let's check that indeed the coefficients given by `glm` are the ones that maximize the likelihood given in the animation of Figure 5.2.1.2. We do so for `y ~ x1`.

```

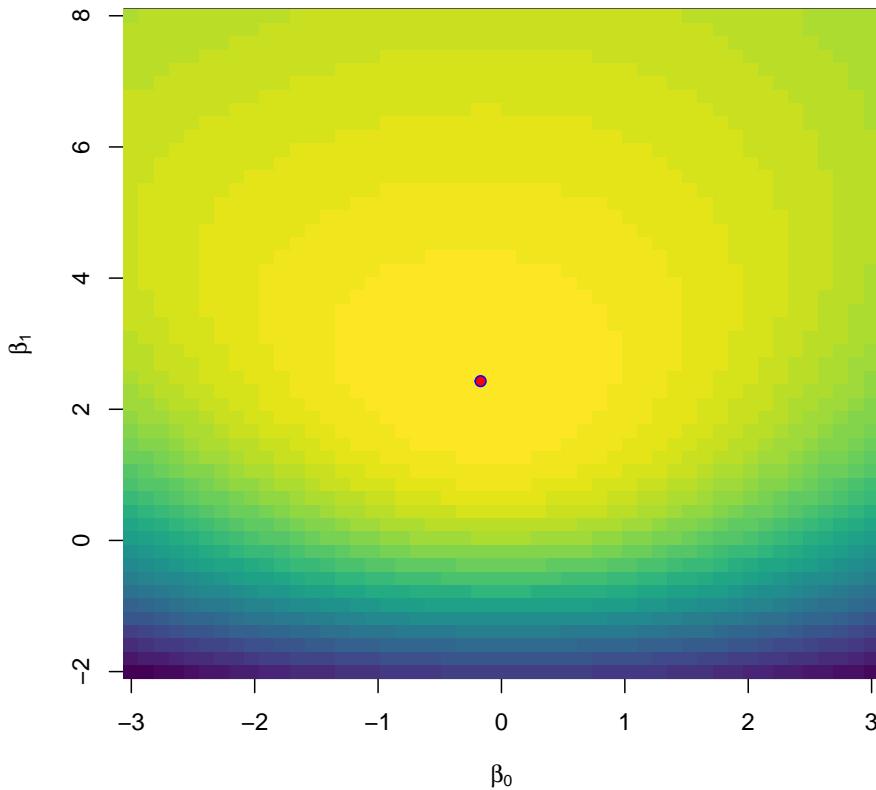
# Call glm
mod <- glm(y1 ~ x, family = "binomial", data = dataMle)
mod$coefficients
## (Intercept)          x
## -0.1691947  2.4281626

# -loglik(beta)
minusLogLik <- function(beta) {
  p <- 1 / (1 + exp(-(beta[1] + beta[2] * x)))
  -sum(y1 * log(p) + (1 - y1) * log(1 - p))
}

# Optimization using as starting values beta = c(0, 0)
opt <- optim(par = c(0, 0), fn = minusLogLik)
opt
## $par
## [1] -0.1691366  2.4285119
##
## $value
## [1] 14.79376
##
## $counts
## function gradient
##       73      NA
##
## $convergence
## [1] 0
##
## $message
## NULL

# Visualization of the minusLogLik surface
library(viridis)
beta0 <- seq(-3, 3, l = 50)
beta1 <- seq(-2, 8, l = 50)
L <- matrix(nrow = length(beta0), ncol = length(beta1))
for (i in seq_along(beta0)) {
  for (j in seq_along(beta1)) {
    L[i, j] <- minusLogLik(c(beta0[i], beta1[j]))
  }
}
image(beta0, beta1, -L, col = viridis(50), xlab = expression(beta[0]),
      ylab = expression(beta[1]))
points(mod$coefficients[1], mod$coefficients[2], col = 2, pch = 16)
points(opt$par[1], opt$par[2], col = 4)

```



For the regressions  $y \sim x_2$  and  $y \sim x_3$ , do the following:

- Check that the true  $\beta$  is close to maximizing the likelihood computed in Figure 5.2.1.2.
- Plot the fitted logistic curve and compare it with the one in Figure 5.2.1.2.

The extension of the logistic model to the case of a *categorical response with more than two levels* is sketched in Appendix G.

## 5.2.2 General case

The same idea we used in logistic regression, namely transforming the conditional expectation of  $Y$  into something that can be modeled by a linear model (this is, a quantity that lives in  $\mathbb{R}$ ), can be generalized. This raises the family of *generalized linear models*, which extend the linear model to different kinds of response variables and provides a convenient parametric framework. The first ingredient is a link function  $g$ , that is monotonic and differentiable, which is going to produce a *transformed expectation* to be modeled by a linear combination of the predictors:

$$g(\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]) = \eta$$

or, equivalently,

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = g^{-1}(\eta),$$

where  $\eta := \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$  is the *linear predictor*.

The second ingredient of generalized linear models is a distribution for  $Y|(X_1, \dots, X_p)$ , just as the linear model assumes normality or the logistic model assumes a Bernoulli random variable. Thus, we have **two generalizations** with respect to the usual linear model:

1. The conditional mean may be modeled by a transformation  $g^{-1}$  of the linear predictor  $\eta$ .
2. The distribution of  $Y|(X_1, \dots, X_p)$  may be different from the Normal.

Generalized linear models are intimately related with the **exponential family**<sup>3</sup>, which is the family of distributions with pdf expressable as

$$f(y; \theta, \phi) = \exp \left\{ \frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi) \right\}, \quad (5.9)$$

where  $a(\cdot)$ ,  $b(\cdot)$ , and  $c(\cdot, \cdot)$  are specific functions. If  $Y$  has the pdf (5.9), then we write  $Y \sim E(\theta, \phi, a, b, c)$ . If the *scale parameter*  $\phi$  is known, this is an exponential family with **canonical parameter**  $\theta$  (if  $\phi$  is unknown, then it may or not may be a two-parameter exponential family). Distributions from the exponential family have some nice properties. Importantly, if  $Y \sim E(\theta, \phi, a, b, c)$ , then

$$\mu := \mathbb{E}[Y] = b'(\theta), \quad \sigma^2 := \text{Var}[Y] = b''(\theta)a(\phi). \quad (5.10)$$

The **canonical link function** is the one that **transforms**  $\mu = b'(\theta)$  **into the canonical parameter**  $\theta$ . For  $E(\theta, \phi, a, b, c)$ , this is happens if

$$\theta = g(\mu) = \eta \quad (5.11)$$

or, equivalently by (5.10), if

$$g(\mu) = (b')^{-1}(\mu). \quad (5.12)$$

In the case of canonical link function, the one-line summary of the generalized linear model is (independence is implicit)

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim E(g(\eta), \phi, a, b, c). \quad (5.13)$$



The **linear model** arises as a particular case of (5.13) with  $a(\phi) = \phi$ ,  $b(\theta) = \frac{\theta^2}{2}$ ,  $c(y, \phi) = -\frac{1}{2}\{\frac{y^2}{\phi} + \log(2\pi\phi)\}$  and scale parameter  $\phi = \sigma^2$ . In this case,  $\theta = \mu$  and the canonical link function  $g$  is the identity.



Show that the Normal, Binomial, Gamma (which includes Exponential and Chi-squared), and Poisson distributions are members of the exponential family. For that, express their pdfs in terms of (5.9) and identify who is  $\theta$  and  $\phi$ .

The following table lists some useful generalized linear models. Recall that the linear and logistic models of Sections 2.2.2 and 5.2.1 are obtained from the first and second rows, respectively.

<sup>3</sup>Not to be confused with the *exponential distribution*  $\text{Exp}(\lambda)$ , which is a *member* of the exponential family.

| Support of $Y$   | Distribution                 | Link $g(\mu)$       | $g^{-1}(\eta)$          | $\phi$     | $Y (X_1 = x_1, \dots, X_p = x_p)$ |
|------------------|------------------------------|---------------------|-------------------------|------------|-----------------------------------|
| $\mathbb{R}$     | $\mathcal{N}(\mu, \sigma^2)$ | $\mu$               | $\eta$                  | $\sigma^2$ | $\mathcal{N}(\eta, \sigma^2)$     |
| $0, 1$           | $B(1, p)$                    | $\text{logit}(\mu)$ | $\text{logistic}(\eta)$ | $1$        | $B(1, \text{logistic}(\eta))$     |
| $0, 1, 2, \dots$ | $\text{Pois}(\lambda)$       | $\log(\mu)$         | $e^\eta$                | $1$        | $\text{Pois}(e^\eta)$             |

The third model is known as **Poisson regression** and is usually employed for modelling **count data** that arises from the recording of the frequencies of a certain phenomenon. It considers that

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim \text{Pois}(e^\eta),$$

this is,

$$\begin{aligned} \mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] &= \lambda(Y|X_1 = x_1, \dots, X_p = x_p) \\ &= e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}. \end{aligned} \quad (5.14)$$

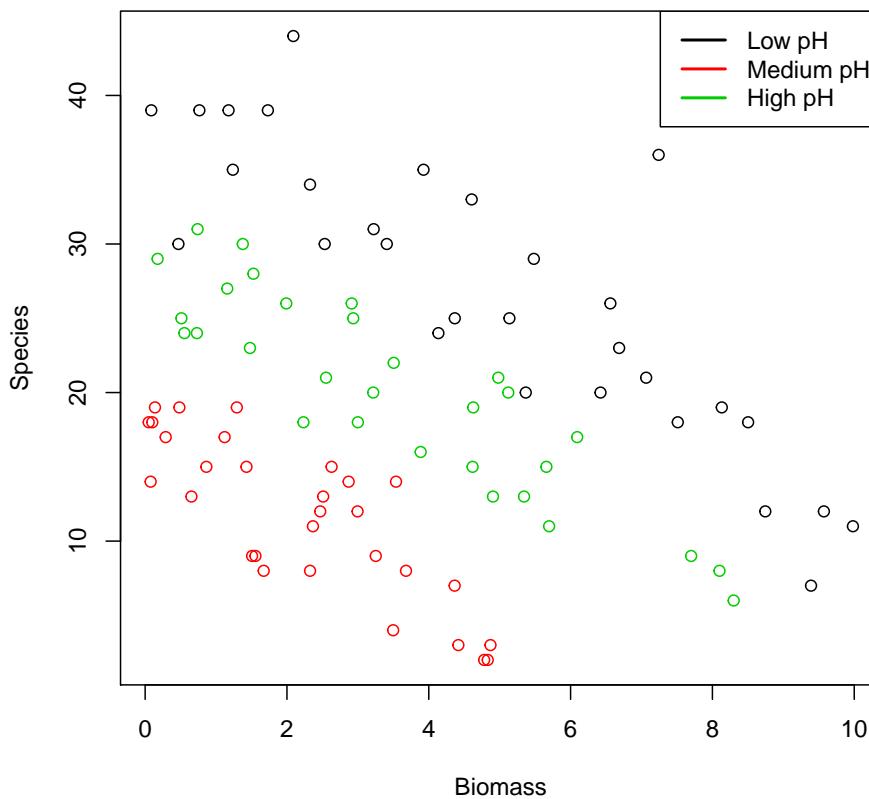
Notice that, since in the Poisson distribution the mean and variance equal, this implies that the variance of  $Y|(X_1 = x_1, \dots, X_p = x_p)$  changes according to the value of the predictors. The interpretation of the coefficients is clear from (5.14):

- $e^{\beta_0}$ : is the expected value *and variance* of  $Y$  when  $X_1 = \dots = X_p = 0$ .
- $e^{\beta_j}$ ,  $1 \leq j \leq p$ : is the *multiplicative* increment of the expectation *and variance* for an increment of one unit in  $X_j = x_j$ , provided that the remaining variables  $X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_p$  do not change.

### 5.2.2.1 Case study application

Let's see how to apply a Poisson regression. For that aim we consider the **species** (download) dataset. The goal is to analyse whether the **Biomass** and **pH** (a factor) of the terrain are influential on the number of **Species**. Incidentally, it will serve to illustrate that the use of factors within **glm** is completely analogous to what we did with **lm**.

```
# Data
plot(Species ~ Biomass, data = species, col = pH)
legend("topright", legend = c("Low pH", "Medium pH", "High pH"),
       col = 1:3, lwd = 2)
```



```

# Fit Poisson regression
species1 <- glm(Species ~ ., data = species, family = poisson)
summary(species1)
##
## Call:
## glm(formula = Species ~ ., family = poisson, data = species)
##
## Deviance Residuals:
##      Min      1Q      Median      3Q      Max
## -2.5959  -0.6989  -0.0737   0.6647   3.5604
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 3.84894   0.05281 72.885 < 2e-16 ***
## pHlow      -1.13639   0.06720 -16.910 < 2e-16 ***
## pHmed      -0.44516   0.05486 -8.114 4.88e-16 ***
## Biomass     -0.12756   0.01014 -12.579 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
## Null deviance: 452.346  on 89  degrees of freedom
## Residual deviance: 99.242  on 86  degrees of freedom
## AIC: 526.43
##
## Number of Fisher Scoring iterations: 4
# Took 4 iterations of the IRLS

```

```

# Interpretation of the coefficients:
exp(species1$coefficients)
## (Intercept)      pHlow      pHmed      Biomass
## 46.9433686   0.3209744   0.6407222   0.8802418
# - 46.9433 is the average number of species when Biomass = 0 and the pH is high
# - For each increment in one unit in Biomass, the number of species decreases
#   by a factor of 0.88 (12% reduction)
# - If pH decreases to med (low), then the number of species decreases by a factor
#   of 0.6407 (0.3209)

# With interactions
species2 <- glm(Species ~ Biomass * pH, data = species, family = poisson)
summary(species2)
##
## Call:
## glm(formula = Species ~ Biomass * pH, family = poisson, data = species)
##
## Deviance Residuals:
##      Min      1Q      Median      3Q      Max
## -2.4978  -0.7485  -0.0402   0.5575   3.2297
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 3.76812   0.06153 61.240 < 2e-16 ***
## Biomass    -0.10713   0.01249 -8.577 < 2e-16 ***
## pHlow     -0.81557   0.10284 -7.931 2.18e-15 ***
## pHmed     -0.33146   0.09217 -3.596 0.000323 ***
## Biomass:pHlow -0.15503   0.04003 -3.873 0.000108 ***
## Biomass:pHmed -0.03189   0.02308 -1.382 0.166954
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
## Null deviance: 452.346 on 89 degrees of freedom
## Residual deviance: 83.201 on 84 degrees of freedom
## AIC: 514.39
##
## Number of Fisher Scoring iterations: 4
exp(species2$coefficients)
## (Intercept)      Biomass      pHlow      pHmed Biomass:pHlow
## 43.2987424   0.8984091   0.4423865   0.7178730   0.8563910
## Biomass:pHmed
## 0.9686112
# - If pH decreases to med (low), then the effect of the biomass in the number
#   of species decreases by a factor of 0.8564 (0.9686). The higher the pH, the
#   stronger the effect of the Biomass in Species

# Draw fits
plot(Species ~ Biomass, data = species, col = pH)
legend("topright", legend = c("High pH", "Medium pH", "Low pH"),
       col = 1:3, lwd = 2)

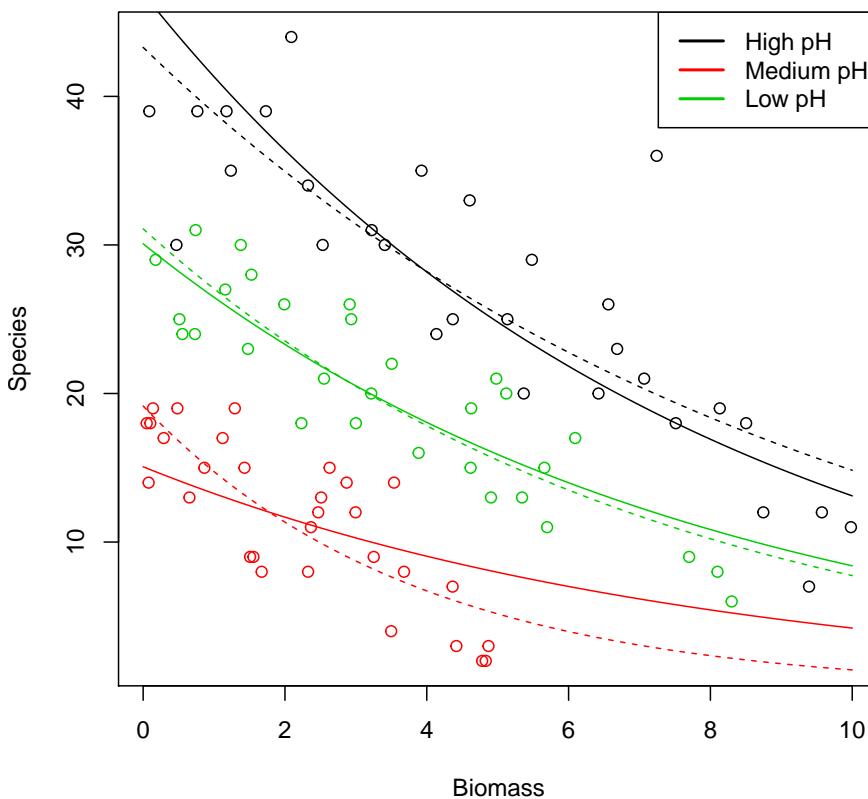
```

```

# Without interactions
bio <- seq(0, 10, 1 = 100)
z <- species1$coefficients[1] + species1$coefficients[4] * bio
lines(bio, exp(z), col = 1)
lines(bio, exp(species1$coefficients[2] + z), col = 2)
lines(bio, exp(species1$coefficients[3] + z), col = 3)

# With interactions seems to provide a significant improvement
bio <- seq(0, 10, 1 = 100)
z <- species2$coefficients[1] + species2$coefficients[2] * bio
lines(bio, exp(z), col = 1, lty = 2)
lines(bio, exp(species2$coefficients[3] + species2$coefficients[5] * bio + z),
      col = 2, lty = 2)
lines(bio, exp(species2$coefficients[4] + species2$coefficients[6] * bio + z),
      col = 3, lty = 2)

```



### 5.2.2.2 Estimation by maximum likelihood

The estimation of  $\beta$  by MLE can be done in a unified framework for all for a generalized linear models thanks to (5.9). Given  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$  and employing a canonical link function (5.12), we have that

$$Y_i | (X_1 = x_{i1}, \dots, X_p = x_{ip}) \sim E(\theta_i, \phi, a, b, c), \quad i = 1, \dots, n,$$

where

$$\begin{aligned} \theta_i &:= g(\eta_i) = g(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}), \\ \mu_i &:= \mathbb{E}[Y_i | X_1 = x_{i1}, \dots, X_p = x_{ip}] = g^{-1}(\eta_i). \end{aligned}$$

Then, the log-likelihood is

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^n \left( \frac{Y_i \theta_i - b(\theta_i)}{a(\phi)} + c(Y_i, \phi) \right). \quad (5.15)$$

Differentiating with respect to  $\boldsymbol{\beta}$  gives

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^n \frac{(Y_i - b'(\theta_i))}{a(\phi)} \frac{\partial \theta_i}{\partial \boldsymbol{\beta}}$$

which, exploiting the properties of the exponential family, can be reduced to

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^n \frac{(Y_i - \mu_i)}{g'(\mu_i) V_i} \mathbf{x}_i,$$

where  $\mathbf{x}_i$  is the  $i$ -th row of the design matrix  $\mathbf{X}$  and  $V_i := \text{Var}[Y_i] = a(\phi)b''(\theta_i)$ . Solving explicitly the system of equations  $\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbf{0}$  is not possible in general and a numerical procedure is required. Newton-Raphson is usually employed, which is based in obtaining  $\boldsymbol{\beta}_{\text{new}}$  from the linear system<sup>4</sup>

$$\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}} (\boldsymbol{\beta}_{\text{new}} - \boldsymbol{\beta}_{\text{old}}) = - \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}}. \quad (5.16)$$

A simplifying trick is to consider the *expectation* of  $\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}}$  in (5.16) rather than its actual value. By doing so, we can arrive at a neat iterative algorithm called *Iterative Reweighted Least Squares* (IRLS). To that aim, we use the following well-known property of the *Fisher information matrix* of the MLE theory:

$$\mathbb{E} \left[ \frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} \right] = -\mathbb{E} \left[ \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \left( \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \right)' \right].$$

Then, it can be seen that<sup>5</sup>

$$\mathbb{E} \left[ \frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}} \right] = - \sum_{i=1}^n w_i \mathbf{x}_i \mathbf{x}_i' = -\mathbf{W}' \mathbf{W}, \quad (5.17)$$

where  $w_i := \frac{1}{V_i(g'(\mu_i))^2}$  and  $\mathbf{W} := \text{diag}(w_1, \dots, w_n)$ . Using this notation,

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}} = \mathbf{X}' \mathbf{W} (\mathbf{Y} - \boldsymbol{\mu}_{\text{old}}) \mathbf{g}'(\boldsymbol{\mu}_{\text{old}}), \quad (5.18)$$

<sup>4</sup>The system stems from a first-order Taylor expansion around the root.

<sup>5</sup>Recall that  $\mathbb{E}[(Y_i - \mu_i)(Y_j - \mu_j)] = \text{Cov}[Y_i, Y_j] = \begin{cases} V_i, & i = j, \\ 0, & i \neq j, \end{cases}$  because of independence.

Substituting (5.17) and (5.18) in (5.16), we have:

$$\begin{aligned}\beta_{\text{new}} &= \beta_{\text{old}} - \mathbb{E} \left[ \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'} \Big|_{\beta=\beta_{\text{old}}} \right]^{-1} \frac{\partial \ell(\beta)}{\partial \beta} \Big|_{\beta=\beta_{\text{old}}} \\ &= \beta_{\text{old}} + (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W} (\mathbf{Y} - \boldsymbol{\mu}_{\text{old}}) \mathbf{g}'(\boldsymbol{\mu}_{\text{old}}) \\ &= (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W} \mathbf{z},\end{aligned}\tag{5.19}$$

where  $\mathbf{z} := \mathbf{X} \beta_{\text{old}} + (\mathbf{Y} - \boldsymbol{\mu}_{\text{old}}) \mathbf{g}'(\boldsymbol{\mu}_{\text{old}})$  is the *working vector*.

As a consequence, **fitting a generalized linear model** by IRLS amounts to performing a **series of weighted linear models** with changing weights and responses given by the working vector. IRLS can be summarized as:

1. Set  $\beta_{\text{old}}$  with some initial estimation.
2. Compute  $\boldsymbol{\mu}_{\text{old}}$ ,  $\mathbf{W}$ , and  $\mathbf{z}_{\text{old}}$ .
3. Compute  $\beta_{\text{new}}$  using (5.19).
4. Iterate steps 2–3 until convergence, then set  $\hat{\beta} = \beta_{\text{new}}$ .



In general,  $\mathbb{E} \left[ \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'} \right] \neq \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'}$ . Thus, IRLS in general departs from the standard Newton-Raphson. However, if the **canonical link** is used, it can be seen that **the equality of the matrices is guaranteed** and IRLS is exactly the same as Newton-Raphson. In that case,  $w_i = \frac{1}{g'(\mu_i)}$ .

## 5.3 Inference for model parameters

The assumptions on which a generalized linear model is constructed allow us to specify what is the *asymptotic* distribution of the *random vector*  $\hat{\beta}$  through the theory of MLE. Again, the distribution is derived conditionally on the sample predictors  $\mathbf{X}_1, \dots, \mathbf{X}_n$ . In other words, we assume that the randomness of  $Y$  comes only from  $Y|(X_1 = x_1, \dots, X_p = x_p)$  and not from the predictors.

For the ease of exposition, we will **focus on the logistic model** rather than in the general case. The conceptual differences are not so big, but the simplification in terms of notation and the benefits on intuition are important.

There is an important difference between the inference results for the linear model and for logistic regression:

- **In linear regression the inference is exact.** This is due to the nice properties of the normal, least squares estimation, and linearity. As a consequence, the distributions of the coefficients are perfectly known assuming that the assumptions hold.
- **In generalized linear models the inference is asymptotic.** This means that the distributions of the coefficients are unknown except for large sample sizes  $n$ , for which we have *approximations*. The reason is the more complexity of the model in terms of non-linearity. This is the usual situation for the majority of regression models.

### 5.3.1 Distributions of the fitted coefficients

The distribution of  $\hat{\beta}$  is given by the asymptotic theory of MLE:

$$\hat{\beta} \sim \mathcal{N}_{p+1}(\beta, I(\beta)^{-1}) \quad (5.20)$$

where  $\sim [\dots]$  must be understood as *asymptotically distributed as*  $[\dots]$  when  $n \rightarrow \infty$  for the rest of the chapter and

$$I(\beta) := -\mathbb{E} \left[ \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'} \right]$$

is the *Fisher information matrix*. The name comes from the fact that *it measures the information available in the sample for estimating  $\beta$* . The “larger” (large eigenvalues) the matrix is, the more precise the estimation of  $\beta$  is, because that results in smaller variances in (5.20).

The inverse of the Fisher information matrix is can be estimated by

$$I(\beta)^{-1} = (\mathbf{X}' \mathbf{V} \mathbf{X})^{-1}, \quad (5.21)$$

where  $\mathbf{V} = \text{diag}(V_1, \dots, V_n)$  and  $V_i = \text{Var}[Y_i]$ . For the logistic model,  $V_i = \text{logistic}(\eta_i)(1 - \text{logistic}(\eta_i))$ , with  $\eta_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}$ . In the case of the multiple linear regression,  $I(\beta)^{-1} = \sigma^2 (\mathbf{X}' \mathbf{X})^{-1}$  (see (2.11)), so the presence of  $\mathbf{V}$  here is a consequence of the heteroskedasticity of the model.

The interpretation of (5.20) and (5.21) gives some useful insights on what concepts affect the quality of the estimation:

- **Bias.** The estimates are *asymptotically* unbiased.
- **Variance.** It depends on:
  - *Sample size n.* Hidden inside  $\mathbf{X}' \mathbf{V} \mathbf{X}$ . As  $n$  grows, the precision of the estimators increases.
  - *Weighted predictor sparsity*  $(\mathbf{X}' \mathbf{V} \mathbf{X})^{-1}$ . The more *sparse* the predictor is (small eigenvalues of  $(\mathbf{X}' \mathbf{V} \mathbf{X})^{-1}$ ), the more precise  $\hat{\beta}$  is.



**The precision of  $\hat{\beta}$  is affected by the value of  $\beta$** , which is hidden inside  $\mathbf{V}$ . This contrasts sharply with the linear model, where the precision of the least squares estimator was *not* affected by the value of the unknown coefficients (see (2.11)). The reason is partially due to the **heteroskedasticity** of logistic regression, which implies a dependence of the variance of  $Y$  in the logistic curve, hence in  $\beta$ .

Illustration of the randomness of the fitted coefficients  $(\hat{\beta}_0, \hat{\beta}_1)$  and the influence of  $n$ ,  $(\beta_0, \beta_1)$  and  $s_x^2$ . The sample predictors  $x_1, \dots, x_n$  are fixed and new responses  $Y_1, \dots, Y_n$  are generated each time from a logistic model  $Y|X = x \sim \text{Ber}(\text{logistic}(\beta_0 + \beta_1 x))$ . Application also available here.

Similar to linear regression, the problem with (5.20) and (5.21) is that  $\mathbf{V}$  is *unknown* in practice because it depends on  $\beta$ . Plugging-in the estimate  $\hat{\beta}$  to  $\beta$  in  $\mathbf{V}$  results in  $\hat{\mathbf{V}}$ . Now we can use  $\hat{\mathbf{V}}$  to get

$$\frac{\hat{\beta}_j - \beta_j}{\hat{\text{SE}}(\hat{\beta}_j)} \sim \mathcal{N}(0, 1), \quad \hat{\text{SE}}(\hat{\beta}_j)^2 := v_j^2 \quad (5.22)$$

where

$v_j$  is the  $j$ -th element of the diagonal of  $(\mathbf{X}' \hat{\mathbf{V}} \mathbf{X})^{-1}$ .

The LHS of (2.16) is the **Wald statistic** for  $\beta_j$ ,  $j = 0, \dots, p$ . They are employed for building confidence intervals and hypothesis tests in an analogous way to the  $t$ -statistics in linear regression.

### 5.3.2 Confidence intervals for the coefficients

Thanks to (5.22), we can have the  $100(1 - \alpha)\%$  CI for the coefficient  $\beta_j$ ,  $j = 0, \dots, p$ :

$$\left( \hat{\beta}_j \pm \hat{\text{SE}}(\hat{\beta}_j) z_{\alpha/2} \right) \quad (5.23)$$

where  $z_{\alpha/2}$  is the  $\alpha/2$ -upper quantile of the  $\mathcal{N}(0, 1)$ . In case we are interested in the CI for  $e^{\beta_j}$ , we can just simply take the exponential on the above CI. So the  $100(1 - \alpha)\%$  CI for  $e^{\beta_j}$ ,  $j = 0, \dots, p$ , is

$$e^{\left( \hat{\beta}_j \pm \hat{\text{SE}}(\hat{\beta}_j) z_{\alpha/2} \right)}.$$

Of course, this CI is **not** the same as  $\left( e^{\hat{\beta}_j} \pm e^{\hat{\text{SE}}(\hat{\beta}_j) z_{\alpha/2}} \right)$ , which is *not* a CI for  $e^{\hat{\beta}_j}$ .

### 5.3.3 Testing on the coefficients

The distributions in (5.22) also allow us to conduct a formal hypothesis test on the coefficients  $\beta_j$ ,  $j = 0, \dots, p$ . For example, the test for significance:

$$H_0 : \beta_j = 0$$

for  $j = 0, \dots, p$ . The test of  $H_0 : \beta_j = 0$  with  $1 \leq j \leq p$  is especially interesting, since it allows us to answer whether *the variable  $X_j$  has a significant effect on  $Y$* . The statistic used for testing for significance is the Wald statistic

$$\frac{\hat{\beta}_j - 0}{\hat{\text{SE}}(\hat{\beta}_j)},$$

which is asymptotically distributed as a  $\mathcal{N}(0, 1)$  *under the (veracity of) the null hypothesis*.  $H_0$  is tested *against* the *bilateral* alternative hypothesis  $H_1 : \beta_j \neq 0$ .

The tests for significance are built-in in the `summary` function. However, a note of caution is required when applying the rule of thumb:

**Is the CI for  $\beta_j$  below (above) 0 at level  $\alpha$ ?**

- Yes  $\rightarrow$  reject  $H_0$  at level  $\alpha$ .
- No  $\rightarrow$  the criterion is not conclusive.



The significances given in `summary` and the output of `confint` are *slightly* incoherent and the previous rule of thumb **does not apply**. The reason is because MASS's `confint` is using a more sophisticated method (profile likelihood) to estimate the standard error of  $\hat{\beta}_j$ ,  $\hat{\text{SE}}(\hat{\beta}_j)$ , and not the asymptotic distribution behind Wald statistic.

By changing `confint` to R's default `confint.default`, the results of the latter will be completely equivalent to the significances in `summary`, and the rule of thumb still be completely valid. For the contents of this course we prefer `confint.default` due to its better interpretability.

### 5.3.4 Case study application

Let's compute the `summary` of the `nasa` model in order to address the significance of the coefficients. 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 significant negative correlation at level  $\alpha = 0.05$ :

```
# Summary of the model
summary(nasa)
##
## Call:
## glm(formula = fail.field ~ temp, family = "binomial", data = challenger)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q      Max
## -1.0566 -0.7575 -0.3818  0.4571  2.2195
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 7.5837    3.9146  1.937   0.0527 .
## temp        -0.4166    0.1940 -2.147   0.0318 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 28.267 on 22 degrees of freedom
## Residual deviance: 20.335 on 21 degrees of freedom
## AIC: 24.335
##
## Number of Fisher Scoring iterations: 5

# Confidence intervals at 95%
confint.default(nasa)
##             2.5 %      97.5 %
## (Intercept) -0.08865488 15.25614140
## temp        -0.79694430 -0.03634877

# Confidence intervals at other levels
confint.default(nasa, level = 0.90)
##             5 %      95 %
## (Intercept) 1.1448638 14.02262275
## temp        -0.7358025 -0.09749059

# Confidence intervals for the factors affecting the odds
exp(confint.default(nasa))
##             2.5 %      97.5 %
## (Intercept) 0.9151614 4.223359e+06
## temp        0.4507041 9.643039e-01
```

The coefficient for `temp` is significant at  $\alpha = 0.05$  and the intercept is not (it is for  $\alpha = 0.10$ ). The 95% confidence interval for  $\beta_0$  is  $(-0.0887, 15.2561)$  and for  $\beta_1$  is  $(-0.7969, -0.0363)$ . For  $e^{\beta_0}$  and  $e^{\beta_1}$ , the CIs are  $(0.9151, 4.2233 \times 10^6)$  and  $(0.4507, 0.9643)$ , respectively. Therefore, we can say with a 95% confidence that:

- When `temp=0`, the probability of `fail.field=1` is *not* significantly larger than the probability of `fail.field=0` (using the CI for  $\beta_0$ ). `fail.field=1` is between 0.9151 and  $4.2233 \times 10^7$  more likely than `fail.field=0` (using the CI for  $e^{\beta_0}$ ).
- **temp has a significantly negative effect in the probability of fail.field=1** (using the CI for  $\beta_1$ ). Indeed, each unit increase in `temp` produces a reduction of the odds of `fail.field` by a factor between 0.3070 and 0.8967 (using the CI for  $e^{\beta_1}$ ).

This completes the answers to Q1 and Q2.

We conclude illustrating the incoherence of `summary` and `confint`.

```
# Significances with asymptotic approximation for the standard errors
summary(nasa)
##
## Call:
## glm(formula = fail.field ~ temp, family = "binomial", data = challenger)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q      Max
## -1.0566 -0.7575 -0.3818  0.4571  2.2195
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.5837    3.9146   1.937   0.0527 .
## temp        -0.4166    0.1940  -2.147   0.0318 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 28.267 on 22 degrees of freedom
## Residual deviance: 20.335 on 21 degrees of freedom
## AIC: 24.335
##
## Number of Fisher Scoring iterations: 5

# CIs with asymptotic approximation - coherent with summary
confint.default(nasa, level = 0.95)
##      2.5 %      97.5 %
## (Intercept) -0.08865488 15.25614140
## temp        -0.79694430 -0.03634877
confint.default(nasa, level = 0.99)
##      0.5 %      99.5 %
## (Intercept) -2.4994971 17.66698362
## temp        -0.9164425  0.08314945

# CIs with profile likelihood - incoherent with summary
confint(nasa, level = 0.95) # intercept still significant
## Waiting for profiling to be done...
##      2.5 %      97.5 %
## (Intercept)  1.3364047 17.7834329
## temp        -0.9237721 -0.1089953
confint(nasa, level = 0.99) # temp still significant
## Waiting for profiling to be done...
##      0.5 %      99.5 %
```

```
## (Intercept) -0.3095128 22.26687651
## temp       -1.1479817 -0.02994011
```

## 5.4 Prediction

Prediction in general linear models focuses mainly on predicting the values of the **conditional mean**

$$\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = g^{-1}(\eta) = g^{-1}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p),$$

by means of  $\hat{\eta} := \hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_p x_p$  and not on predicting the conditional response. The reason is that confidence intervals, the main difference between both kinds of prediction, depend heavily on the family we are considering for the response.

For the logistic model, the prediction of the conditional response follows immediately from `logistic`( $\hat{\eta}$ ):

$$\hat{Y}|(X_1 = x_1, \dots, X_p = x_p) = \begin{cases} 1, & \text{with probability } \text{logistic}(\hat{\eta}), \\ 0, & \text{with probability } 1 - \text{logistic}(\hat{\eta}). \end{cases}$$

As a consequence, we can predict  $Y$  as 1 if  $\text{logistic}(\hat{\eta}) > \frac{1}{2}$  and as 0 otherwise.

To make predictions and compute CIs in practice we use `predict`. There are **two differences** with respect to its use for `lm`:

- The argument `type`. `type = "link"` returns  $\hat{\eta}$  (the log-odds in the logistic model), `type = "response"` returns  $g^{-1}(\hat{\eta})$  (the probabilities in the logistic model). Observe that `type = "response"` has a different than in `predict` for `lm`, where it returned the predictions for the conditional response.
- There is no `interval` argument for using `predict` with `glm`. That means that there is no straightforward way of computing CIs for prediction.

Figure 5.4 gives an interactive visualization of the CIs for the conditional probability in simple logistic regression. Their interpretation is very similar to the CIs for the conditional mean in the simple linear model, see Section 2.5 and Figure 2.5.

Illustration of the CIs for the conditional probability in the simple logistic regression. Application also available here.

### 5.4.1 Case study application

Let's compute what was the probability of having at least one incident with the O-rings in the launch day (answers Q3):

```
predict(nasa, newdata = data.frame(temp = -0.6), type = "response")
##           1
## 0.999604
```

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.

Since it is a bit cumbersome to compute the CIs for the conditional response, we can code the function `predictCIsLogistic` to do it automatically.

```
# Function for computing the predictions and CIs for the conditional probability
predictCIsLogistic <- function(object, newdata, level = 0.95) {
  # Compute predictions in the log-odds
```

```

pred <- predict(object = object, newdata = newdata, se.fit = TRUE)

# CI in the log-odds
za <- qnorm(p = (1 - level) / 2)
lwr <- pred$fit + za * pred$se.fit
upr <- pred$fit - za * pred$se.fit

# Transform to probabilities
fit <- 1 / (1 + exp(-pred$fit))
lwr <- 1 / (1 + exp(-lwr))
upr <- 1 / (1 + exp(-upr))

# Return a matrix with column names "fit", "lwr" and "upr"
result <- cbind(fit, lwr, upr)
colnames(result) <- c("fit", "lwr", "upr")
return(result)

}

```

Let's apply it to the case study:

```

# Data for which we want a prediction
newdata <- data.frame(temp = -0.6)

# Prediction of the conditional log-odds, the default
predict(nasa, newdata = newdata, type = "link")
##          1
## 7.833731

# Prediction of the conditional probability
predict(nasa, newdata = newdata, type = "response")
##          1
## 0.999604

# Simple call
predictCIsLogistic(nasa, newdata = newdata)
##      fit      lwr      upr
## 1 0.999604 0.4838505 0.9999999
# The CI is large because there is no data around temp = -0.6 and
# that makes the prediction more variable (and also because we only
# have 23 observations)

```



For the `challenger` dataset, do the following:

- Regress `fail.nozzle` on `temp` and `pres.nozzle`.
- Compute the predicted probability of `fail.nozzle=1` for `temp= 15` and `pres.nozzle= 200`. What is the predicted probability for `fail.nozzle=0`?
- Compute the confidence interval for the two predicted probabilities at level 95%.

## 5.5 Deviance

The **deviance** is a key concept in generalized linear models. Intuitively, it measures the *deviance of the fitted generalized linear model with respect to a perfect model for  $\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]$* . This perfect model, known as the *saturated model*, is the model that perfectly fits the data, in the sense that it has as many parameters as observations and the fitted responses ( $\hat{Y}_i$ ) are the same as the observed responses ( $Y_i$ ). For example, in logistic regression this would be the model such that

$$\hat{\mathbb{P}}[Y = 1|X_1 = X_{i1}, \dots, X_k = X_{ip}] = Y_i, \quad i = 1, \dots, n.$$

Figure 5.4 shows a saturated model and a fitted logistic regression to a dataset.

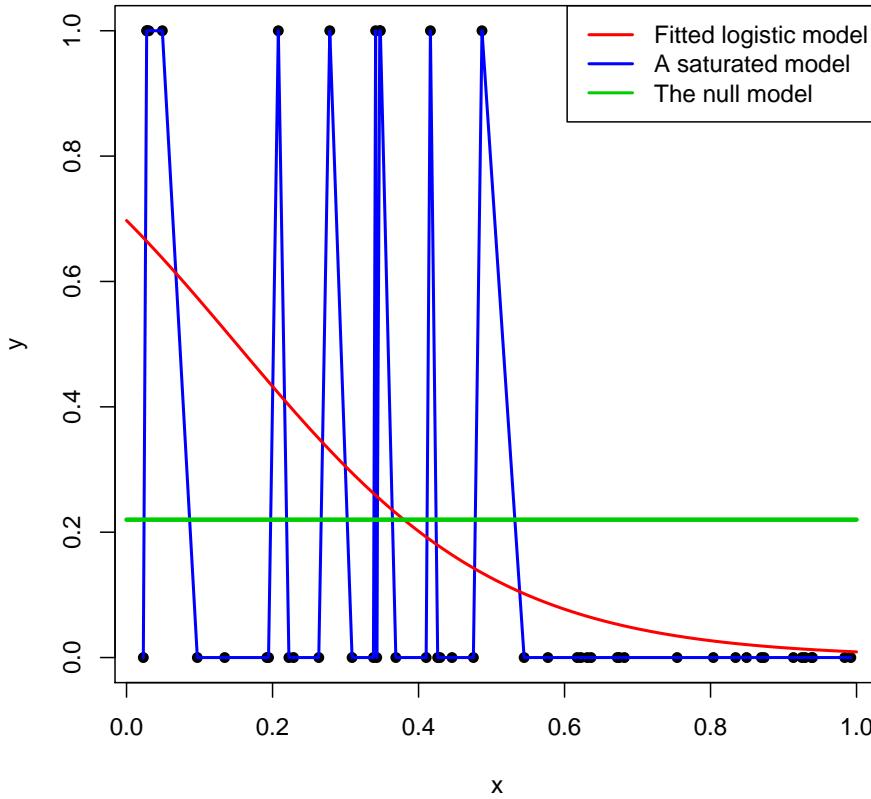


Figure 5.4: Fitted logistic regression versus a saturated model (several are possible depending on the interpolation between points) and the null model.

Formally, the deviance is defined through the difference of the log-likelihoods between the fitted model,  $\ell(\hat{\beta})$ , and the saturated model,  $\ell_s$ . Computing  $\ell_s$  amounts to substitute  $\mu_i$  by  $Y_i$  in (5.15). If the canonical link function is used, this corresponds to setting  $\theta_i = g(Y_i)$  (recall (5.11)). The deviance is then defined as:

$$D := -2 \left[ \ell(\hat{\beta}) - \ell_s \right] \phi.$$

The log-likelihood  $\ell(\hat{\beta})$  is always smaller than  $\ell_s$  (the saturated model is more likely since it has the maximum flexibility). As a consequence, the *deviance is always larger or equal than zero*, being zero only if the fit of the model is perfect.

If the canonical link function is employed, the deviance can be expressed as

$$\begin{aligned}
D &= -\frac{2}{a(\phi)} \sum_{i=1}^n \left( Y_i \hat{\theta}_i - b(\hat{\theta}_i) - Y_i g(Y_i) + b(g(Y_i)) \right) \phi \\
&= \frac{2\phi}{a(\phi)} \sum_{i=1}^n \left( Y_i (Y_i - \hat{\theta}_i) - b(g(Y_i)) + b(\hat{\theta}_i) \right). \tag{5.24}
\end{aligned}$$

In most of the cases,  $a(\phi) \propto \phi$ , so the deviance *does not depend on*  $\phi$ . Expression (5.24) is interesting, since it allows the following key insight:



The deviance is a **generalization of the Residual Sum of Squares (RSS) of the linear model**. The generalization is driven by the likelihood and its equivalence with the RSS in the linear model.

To see it, let consider the linear model in (5.24) by setting  $\phi = \sigma^2$ ,  $a(\phi) = \phi$ ,  $b(\theta) = \frac{\theta^2}{2}$ ,  $c(y, \phi) = -\frac{1}{2} \left\{ \frac{y^2}{\phi} + \log(2\pi\phi) \right\}$  and  $\theta = \mu = \eta$  (the canonical link function  $g$  is the identity, check (5.11) and (5.12)). Then, we have:

$$\begin{aligned}
D &= \frac{2\sigma^2}{\sigma^2} \sum_{i=1}^n \left( Y_i (Y_i - \hat{\eta}_i) - \frac{Y_i^2}{2} + \frac{\hat{\eta}_i^2}{2} \right) \\
&= \sum_{i=1}^n (2Y_i^2 - 2Y_i \hat{\eta}_i - Y_i^2 + \hat{\eta}_i^2) \\
&= \sum_{i=1}^n (Y_i - \hat{\eta}_i)^2 \\
&= \text{RSS}(\hat{\beta}),
\end{aligned}$$

since  $\hat{\eta}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \dots + \hat{\beta}_p x_{ip}$ . Remember that  $\text{RSS}(\hat{\beta})$  is just another name for the SSE.

A benchmark for evaluating the magnitude of the deviance is the **null deviance**,

$$D_0 := -2 \left[ \ell(\hat{\beta}_0) - \ell_s \right] \phi,$$

which is the **deviance of the worst model, the one fitted without any predictor**, to the perfect model. For example, in logistic regression:

$$Y | (X_1 = x_1, \dots, X_p = x_p) \sim \text{Ber}(\text{logistic}(\beta_0)).$$

In this case,  $\hat{\beta}_0 = \text{logit}(\frac{m}{n}) = \log \frac{\frac{m}{n}}{1 - \frac{m}{n}}$  where  $m$  is the number of 1's in  $Y_1, \dots, Y_n$  (see Figure 5.4).

Using again (5.24), we can see that the **null deviance is a generalization of the total sum of squares of the linear model** (see Section 2.6):

$$D_0 = \sum_{i=1}^n (Y_i - \hat{\eta}_i)^2 = \sum_{i=1}^n (Y_i - \hat{\beta}_0)^2 = \text{SST},$$

since  $\hat{\beta}_0 = \bar{Y}$  because there are no predictors.

Using the deviance and the null deviance, we can compare how much the model has improved by adding the predictors  $X_1, \dots, X_p$  and quantify the *percentage of deviance explained*. This can be done by means of the

$R^2$  statistic, which is a generalization of the determination coefficient for linear regression:

$$R^2 := 1 - \frac{D}{D_0} \stackrel{\text{linear model}}{=} 1 - \frac{\text{SSE}}{\text{SST}}.$$

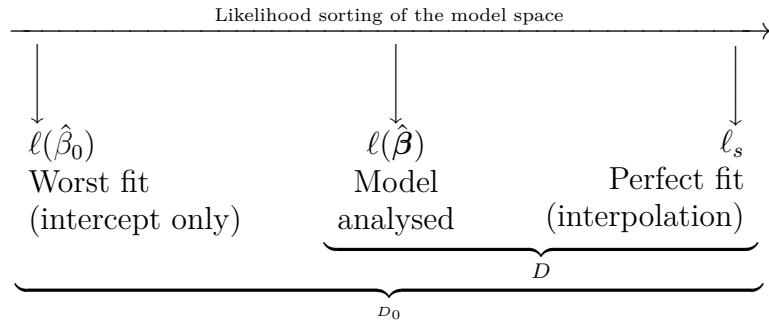


The  $R^2$  for generalized linear models is a global measure of fit that shares the same philosophy with the determination coefficient in linear regression: it is a proportion of how good the fit is. If perfect,  $D = 0$  and  $R^2 = 1$ . If the predictors do not add anything to the regression, then  $D = D_0$  and  $R^2 = 0$ .

However, this  $R^2$  has a different interpretation than the one in linear regression. In particular:

- Is **not the percentage of variance explained by the model**, but rather a ratio indicating how close is the fit to being perfect or the worst.
- It is not related to any correlation coefficient.

An illustrative pictorial representation of the deviance ( $D$ ) and null deviance ( $D_0$ ) is the following:



The deviance is returned by `summary`:

```
# Summary of model
nasa <- glm(fail.field ~ temp, family = "binomial", data = challenger)
summaryLog <- summary(nasa)
summaryLog
##
## Call:
## glm(formula = fail.field ~ temp, family = "binomial", data = challenger)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q      Max
## -1.0566 -0.7575 -0.3818  0.4571  2.2195
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 7.5837    3.9146   1.937   0.0527 .
## temp       -0.4166    0.1940  -2.147   0.0318 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 28.267 on 22 degrees of freedom
## Residual deviance: 20.335 on 21 degrees of freedom
```

```

## AIC: 24.335
##
## Number of Fisher Scoring iterations: 5
# 'Residual deviance' is the deviance; 'Null deviance' is the null deviance

# Null model (only intercept)
null <- glm(fail.field ~ 1, family = "binomial", data = challenger)
summaryNull <- summary(null)
summaryNull
##
## Call:
## glm(formula = fail.field ~ 1, family = "binomial", data = challenger)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q      Max
## -0.8519 -0.8519 -0.8519  1.5425  1.5425
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.8267    0.4532 -1.824   0.0681 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 28.267 on 22 degrees of freedom
## Residual deviance: 28.267 on 22 degrees of freedom
## AIC: 30.267
##
## Number of Fisher Scoring iterations: 4

# Computation of the R^2 with a function - useful for repetitive computations
r2glm <- function(model) {

  summaryLog <- summary(model)
  1 - summaryLog$deviance / summaryLog>null.deviance

}

# R^2
r2glm(nasa)
## [1] 0.280619
r2glm(null)
## [1] -2.220446e-16

```

A related quantity with the deviance is the **scaled deviance**:

$$D^* := \frac{D}{\phi} = -2 \left[ \ell(\hat{\beta}) - \ell_s \right].$$

If  $\phi = 1$ , such as in the logistic or Poisson regression models, then both the deviance and the scaled deviance agree. The scaled deviance has asymptotic distribution

$$D^* \sim \chi^2_{n-p-1}, \quad (5.25)$$

where  $\chi_k^2$  is the *Chi-squared distribution with  $k$  degrees of freedom*. In the case of the linear model,  $D^* = \frac{1}{\sigma^2} \text{RSS}$  is *exactly* distributed as a  $\chi_{n-p-1}^2$ . The result (5.25) provides a way of estimating  $\phi$  when it is unknown: match  $D^* = \frac{D}{\phi}$  with the expectation  $\mathbb{E}[\chi_{n-p-1}^2] = n - p - 1$ . This provides

$$\hat{\phi}_D := \frac{D}{n - p - 1} = \frac{-2\ell(\hat{\beta})}{n - p - 1},$$

which, as expected, in the case of the linear model is equivalent to  $\hat{\sigma}^2$  as given in (2.15). More importantly, the scaled deviance can be used for performing **hypotheses tests on sets of coefficients** of a generalized linear model.

Assume we have one model, say model 2, with  $p_2$  predictors and another model, say model 1, with  $p_1 < p_2$  predictors that are contained in the set of predictors of the model 2. In other words, assume model 1 is **nested** within model 2. Then we can test the null hypothesis that the extra coefficients of model 2 are simultaneously zero. For example, if model 1 has the coefficients  $\{\beta_0, \beta_1, \dots, \beta_{p_1}\}$  and model 2 has  $\{\beta_0, \beta_1, \dots, \beta_{p_1}, \beta_{p_1+1}, \dots, \beta_{p_2}\}$ , we can test

$$H_0 : \beta_{p_1+1} = \dots = \beta_{p_2} = 0 \quad \text{vs.} \quad H_1 : \beta_j \neq 0 \text{ for any } p_1 < j \leq p_2.$$

This can be done by means of the statistic<sup>6</sup>

$$D_{p_1}^* - D_{p_2}^* \stackrel{H_0}{\sim} \chi_{p_2-p_1}^2. \quad (5.26)$$

If  $H_0$  is true, then  $D_{p_1}^* - D_{p_2}^*$  is expected to be *small*, thus we will reject  $H_0$  if the value of the statistic is above the  $\alpha$ -upper quantile of the  $\chi_{p_2-p_1}^2$ , denoted as  $\chi_{\alpha;p_2-p_1}^2$ .

Note that  $D^*$  apparently removes the effects of  $\phi$ , but it is still dependent on  $\phi$ , since this is hidden in the likelihood (see (5.24)). Therefore,  $D^*$  cannot be computed unless  $\phi$  is known, which forbids using (5.26). Hopefully, this dependence is removed by employing (5.25) and (5.26) and assuming that they are asymptotically independent. This gives the  $F$ -test for  $H_0$ :

$$F = \frac{(D_{p_1}^* - D_{p_2}^*)/(p_2 - p_1)}{D_{p_2}^*/(n - p_2 - 1)} = \frac{(D_{p_1} - D_{p_2})/(p_2 - p_1)}{D_{p_2}/(n - p_2 - 1)} = \stackrel{H_0}{\sim} F_{p_2-p_1, n-p_2-1}.$$

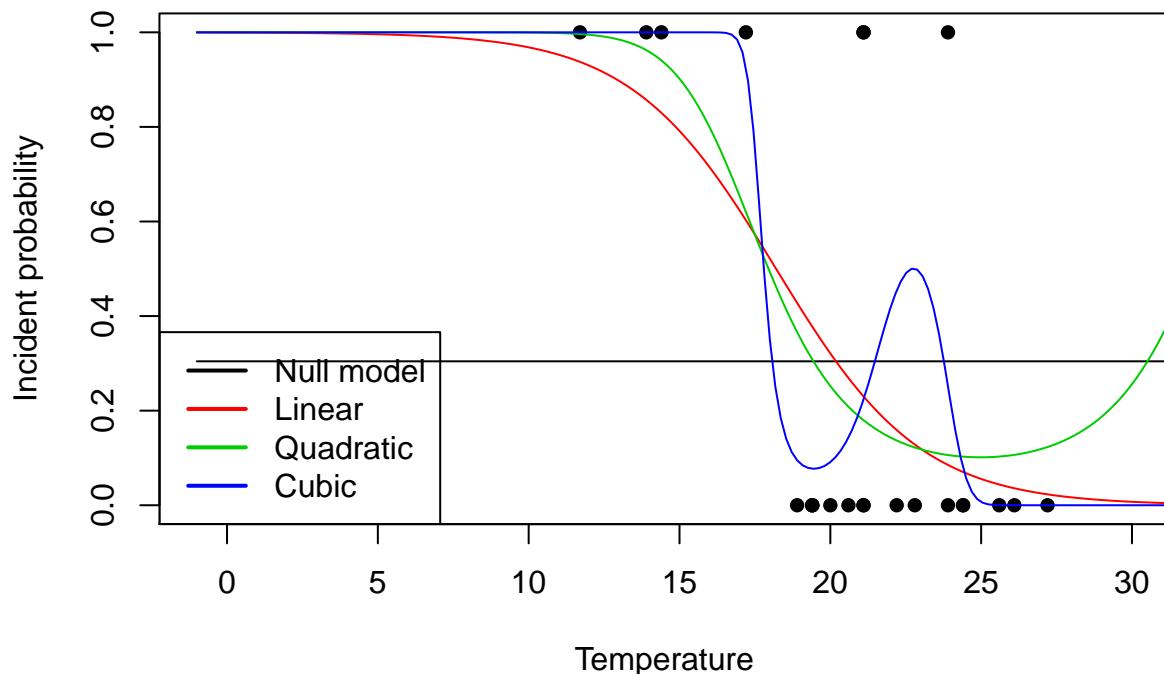
Note that the LHS is perfectly computable, since  $\phi$  cancels due to the quotient (and because we assume that  $a(\phi) \propto \phi$ ). Note also that this is an extension of the  $F$ -test as we saw it in Section 2.6: take  $p_1 = 0$  and  $p_2 = p$  and you test for the significance of all the predictors included in the model (both models contain intercept).

The computation of deviances and associated tests is done through `anova`, which implements the *Analysis of Deviance*.

```
# Polynomial predictors
nasa0 <- glm(fail.field ~ 1, family = "binomial", data = challenger)
nasa1 <- glm(fail.field ~ temp, family = "binomial", data = challenger)
nasa2 <- glm(fail.field ~ poly(temp, degree = 2), family = "binomial",
             data = challenger)
nasa3 <- glm(fail.field ~ poly(temp, degree = 3), family = "binomial",
             data = challenger)
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

<sup>6</sup>Note that  $D_{p_1}^* > D_{p_2}^*$  because  $\ell(\hat{\beta}_{p_1}) < \ell(\hat{\beta}_{p_2})$ .

```
# Plot fits
temp <- seq(-1, 35, l = 200)
tt <- data.frame(temp = temp)
plot(fail.field ~ temp, data = challenger, pch = 16, xlim = c(-1, 30),
      xlab = "Temperature", ylab = "Incident probability")
lines(temp, predict(nasa0, newdata = tt, type = "response"), col = 1)
lines(temp, predict(nasa1, newdata = tt, type = "response"), col = 2)
lines(temp, predict(nasa2, newdata = tt, type = "response"), col = 3)
lines(temp, predict(nasa3, newdata = tt, type = "response"), col = 4)
legend("bottomleft", legend = c("Null model", "Linear", "Quadratic", "Cubic"),
      lwd = 2, col = 1:4)
```



```
# R^2's
r2glm(nasa0)
## [1] -2.220446e-16
r2glm(nasa1)
## [1] 0.280619
r2glm(nasa2)
## [1] 0.3138925
r2glm(nasa3)
## [1] 0.4831863

# Chisq and F tests - same results since phi is known
anova(nasa1, test = "Chisq")
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: fail.field
##
## Terms added sequentially (first to last)
```

```

##  

##  

##      Df Deviance Resid. Df Resid. Dev Pr(>Chi)  

##  NULL              22     28.267  

##  temp   1    7.9323      21     20.335 0.004856 **  

##  ---  

##  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  

anova(nasa1, test = "F")  

## Warning in anova.glm(nasa1, test = "F"): using F test with a 'binomial'  

## family is inappropriate  

## Analysis of Deviance Table  

##  

## Model: binomial, link: logit  

##  

## Response: fail.field  

##  

## Terms added sequentially (first to last)  

##  

##  

##      Df Deviance Resid. Df Resid. Dev      F  Pr(>F)  

##  NULL              22     28.267  

##  temp   1    7.9323      21     20.335 7.9323 0.004856 **  

##  ---  

##  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Incremental comparisons of nested models
anova(nasa1, nasa2, nasa3, test = "Chisq")
## Analysis of Deviance Table
##  

## Model 1: fail.field ~ temp
## Model 2: fail.field ~ poly(temp, degree = 2)
## Model 3: fail.field ~ poly(temp, degree = 3)
##      Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      21     20.335
## 2      20     19.394  1    0.9405  0.3321
## 3      19     14.609  1    4.7855  0.0287 *
##  ---  

##  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# Quadratic effects are not significative

# Cubic vs linear
anova(nasa1, nasa3, test = "Chisq")
## Analysis of Deviance Table
##  

## Model 1: fail.field ~ temp
## Model 2: fail.field ~ poly(temp, degree = 3)
##      Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      21     20.335
## 2      19     14.609  2     5.726   0.0571 .
##  ---  

##  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Example in Poisson regression

```

```

species1 <- glm(Species ~ ., data = species, family = poisson)
species2 <- glm(Species ~ Biomass * pH, data = species, family = poisson)

# Comparison
anova(species1, species2, test = "Chisq")
## Analysis of Deviance Table
##
## Model 1: Species ~ pH + Biomass
## Model 2: Species ~ Biomass * pH
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       86    99.242
## 2       84    83.201  2     16.04 0.0003288 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
r2glm(species1)
## [1] 0.7806071
r2glm(species2)
## [1] 0.8160674

```

## 5.6 Model selection

The same discussion we did in Section 3.2 is applicable to generalized linear models with small changes:

1. The **deviance** of the model (reciprocally the likelihood and the  $R^2$ ) **always decreases** (increase) with the inclusion of more predictors – no matter whether they are significant or not.
2. The **excess of predictors** in the model is paid by a larger variability in the estimation of the model which results in less precise prediction.
3. **Multicollinearity** may hide significant variables, change the sign of them and result in an increase of the variability of the estimation.

Stepwise selection can be done through `stepAIC` as in linear models<sup>7</sup>. Conveniently `summary` also reports the AIC:

```

# Models
nasa1 <- glm(fail.field ~ temp, family = "binomial", data = challenger)
nasa2 <- glm(fail.field ~ temp + pres.field, family = "binomial",
             data = challenger)

# Summaries
summary(nasa1)
##
## Call:
## glm(formula = fail.field ~ temp, family = "binomial", data = challenger)
##
## Deviance Residuals:
##       Min      1Q      Median      3Q      Max
## -1.0566 -0.7575 -0.3818  0.4571  2.2195
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 7.5837    3.9146   1.937   0.0527 .

```

<sup>7</sup>The `leaps` package does not support generalized linear models directly. There are, however, other packages for performing best subset selection with generalized linear models, but we do not cover them here.

```

## temp          -0.4166      0.1940   -2.147   0.0318 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 28.267  on 22  degrees of freedom
## Residual deviance: 20.335  on 21  degrees of freedom
## AIC: 24.335
##
## Number of Fisher Scoring iterations: 5
summary(nasa2)
##
## Call:
## glm(formula = fail.field ~ temp + pres.field, family = "binomial",
##      data = challenger)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q        Max
## -1.2109  -0.6081  -0.4292   0.3498   2.0913
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 6.642709  4.038547  1.645   0.1000
## temp       -0.435032  0.197008 -2.208   0.0272 *
## pres.field  0.009376  0.008821  1.063   0.2878
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 28.267  on 22  degrees of freedom
## Residual deviance: 19.078  on 20  degrees of freedom
## AIC: 25.078
##
## Number of Fisher Scoring iterations: 5

# AICs
AIC(nasa1) # Better
## [1] 24.33485
AIC(nasa2)
## [1] 25.07821

```

stepAIC works analogously to the linear regression situation. Here is an illustration for a binary variable that measures whether a Boston suburb (Boston dataset from Section 3.1) is wealth or not. The binary variable is `medv > 25`: it is TRUE (1) for suburbs with median house value larger than 25000\$) and FALSE (0) otherwise. The cutoff 25000\$ corresponds to the 25% richest suburbs.

```

# Boston dataset
data(Boston)

# Model whether a suburb has a median house value larger than $25000
mod <- glm(I(medv > 25) ~ ., data = Boston, family = "binomial")
summary(mod)

```

```

## Call:
## glm(formula = I(medv > 25) ~ ., family = "binomial", data = Boston)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q      Max
## -3.3498 -0.2806 -0.0932 -0.0006  3.3781
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 5.312511  4.876070  1.090 0.275930
## crim        -0.011101  0.045322 -0.245 0.806503
## zn          0.010917  0.010834  1.008 0.313626
## indus       -0.110452  0.058740 -1.880 0.060060 .
## chas        0.966337  0.808960  1.195 0.232266
## nox         -6.844521  4.483514 -1.527 0.126861
## rm          1.886872  0.452692  4.168 3.07e-05 ***
## age         0.003491  0.011133  0.314 0.753853
## dis         -0.589016  0.164013 -3.591 0.000329 ***
## rad         0.318042  0.082623  3.849 0.000118 ***
## tax         -0.010826  0.004036 -2.682 0.007314 **
## ptratio     -0.353017  0.122259 -2.887 0.003884 **
## black       -0.002264  0.003826 -0.592 0.554105
## lstat       -0.367355  0.073020 -5.031 4.88e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 563.52 on 505 degrees of freedom
## Residual deviance: 209.11 on 492 degrees of freedom
## AIC: 237.11
##
## Number of Fisher Scoring iterations: 7
r2glm(mod)
## [1] 0.628923

# With BIC - ends up with only the significant variables and a similar R^2
modBIC <- stepAIC(mod, trace = 0, k = log(nrow(Boston)))
summary(modBIC)
##
## Call:
## glm(formula = I(medv > 25) ~ indus + rm + dis + rad + tax + ptratio +
##       lstat, family = "binomial", data = Boston)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q      Max
## -3.3077 -0.2970 -0.0947 -0.0005  3.2552
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.556433  3.948818  0.394 0.693469
## indus       -0.143236  0.054771 -2.615 0.008918 **

```

```

##  rm          1.950496  0.441794  4.415 1.01e-05 ***
##  dis         -0.426830  0.111572 -3.826 0.000130 ***
##  rad          0.301060  0.076542  3.933 8.38e-05 ***
##  tax          -0.010240  0.003631 -2.820 0.004800 **
##  ptratio     -0.404964  0.112086 -3.613 0.000303 ***
##  lstat        -0.384823  0.069121 -5.567 2.59e-08 ***
##  ---
##  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  (Dispersion parameter for binomial family taken to be 1)
##
##  Null deviance: 563.52  on 505  degrees of freedom
##  Residual deviance: 215.03  on 498  degrees of freedom
##  AIC: 231.03
##
##  Number of Fisher Scoring iterations: 7
r2glm(modBIC)
## [1] 0.6184273

```

The logistic model is at the intersection between *regression models* and *classification methods*. Therefore, the search for adequate predictors to be included in the model can also be done in terms of the **classification performance**. Although we do not explore in detail this direction, we simply mention how the overall predictive accuracy can be summarized with the *hit matrix* (also called *confusion matrix*)

| Reality vs. classified | $\hat{Y} = 0$           | $\hat{Y} = 1$           |
|------------------------|-------------------------|-------------------------|
| $Y = 0$                | Correct <sub>00</sub>   | Incorrect <sub>01</sub> |
| $Y = 1$                | Incorrect <sub>10</sub> | Correct <sub>11</sub>   |

and with the *hit ratio*,  $\frac{\text{Correct}_0 + \text{Correct}_1}{n}$ , which is the percentage of correct classifications. The hit matrix is easily computed with the `table` function. The function, whenever called with two vectors, computes the cross-table between the two vectors.

```

# Fitted probabilities for Y = 1
nasa$fitted.values
##      1        2        3        4        5        6
## 0.42778935 0.23014393 0.26910358 0.32099837 0.37772880 0.15898364
##      7        8        9       10       11       12
## 0.12833090 0.23014393 0.85721594 0.60286639 0.23014393 0.04383877
##     13       14       15       16       17       18
## 0.37772880 0.93755439 0.37772880 0.08516844 0.23014393 0.02299887
##     19       20       21       22       23
## 0.07027765 0.03589053 0.08516844 0.07027765 0.82977495

# Classified Y's
yHat <- nasa$fitted.values > 0.5

# Hit matrix:
# - 16 correctly classified as 0
# - 4 correctly classified as 1
# - 3 incorrectly classified as 0
tab <- table(challenger$fail.field, yHat)
tab
##      yHat

```

```

##      FALSE TRUE
## 0      16   0
## 1       3   4

# Hit ratio (ratio of correct classification)
sum(diag(tab)) / sum(tab)
## [1] 0.8695652

```

It is important to recall that the hit matrix will be always *biased towards unrealistically good classification rates* if it is computed in the same sample used for fitting the logistic model. An approach based on multi-splitting/cross-validation would be required to handle this problem suitably.



For the `Boston` dataset, do the following:

1. Compute the hit matrix and hit ratio for the regression  $I(\text{medv} > 25) \sim \dots$
2. Fit  $I(\text{medv} > 25) \sim \dots$  but now using only the first 300 observations of `Boston`, the training dataset.
3. For the previous model, predict the probability of the responses and classify them into 0 or 1 in the last 206 observations, the testing dataset.
4. Compute the hit matrix and hit ratio for the new predictions. Check that the hit ratio is smaller than the one in the first point.

## 5.7 Model diagnostics

As it was implicit in Section 5.2, generalized linear models are built on some probabilistic assumptions that are required for performing inference on the model parameters  $\beta$  and  $\phi$ . Unless stated otherwise,  $\sim [\dots]$  denotes *distributed as*  $[\dots]$  along this section.

In general, we assume that the data has been generated from (independence is implicit)

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim E(\theta(x_1, \dots, x_p)), \phi, a, b, c), \quad (5.27)$$

such that

$$\mu = \mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p] = g^{-1}(\eta),$$

where  $\theta$  is given by (check (5.10)–(5.12))

$$\theta = \begin{cases} \mu, & \text{if } g \text{ is the canonical link function,} \\ (b')^{-1}(\mu), & \text{otherwise,} \end{cases}$$

and  $\eta = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$ .

In the case of the logistic and Poisson regressions, both with canonical link functions, the general model takes the form (independence is implicit)

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim \text{Ber}(\text{logistic}(\eta)), \quad (5.28)$$

$$Y|(X_1 = x_1, \dots, X_p = x_p) \sim \text{Pois}(e^\eta). \quad (5.29)$$

The assumptions behind (5.27), (5.28), and (5.29) are the following:

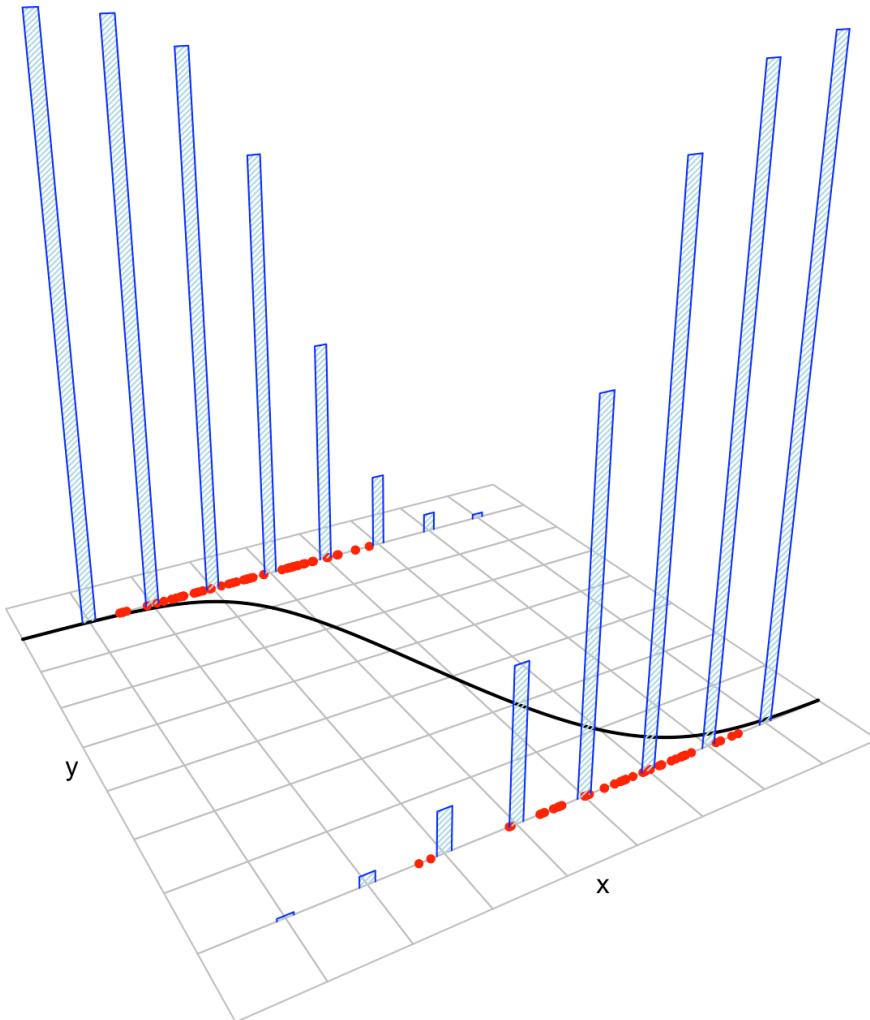


Figure 5.5: The key concepts of the logistic model.

- i. **Linearity** in the transformed expectation:  $g(\mathbb{E}[Y|X_1 = x_1, \dots, X_p = x_p]) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$ .
- ii. **Response distribution**:  $Y|\mathbf{X} = \mathbf{x} \sim E(\theta(\mathbf{x}), \phi, a, b, c)$  ( $\phi, a, b, c$  are *constant* for  $\mathbf{x}$ ).
- iii. **Independence**:  $Y_1, \dots, Y_n$  are independent, conditionally on  $\mathbf{X}_1, \dots, \mathbf{X}_n$ .

There are two important points of the linear model assumptions “missing” here:

- *Where is homoscedasticity?* Homoscedasticity is specific to certain exponential family distributions for whose  $\theta$  does not affect the variance. This is not the case for Bernoulli or Poisson distributions variables, which result in heteroskedastic models. Also, homoscedasticity is the implication of assumption ii in the case of the Normal distribution.
- *Where are the errors?* The errors are not fundamental for building the linear model, but just a helpful concept related to least squares. The linear model can be constructed without errors using (2.8).



Recall that:

- Nothing is said about the distribution of  $X_1, \dots, X_p$ . They could be deterministic or random. They could be discrete or continuous.
- $X_1, \dots, X_p$  are **not required to be independent** between them.

Checking the assumptions of a generalized linear model is more complicated than what we did in 3.5. The reason is the **heterogeneity and heteroskedasticity of the responses**, which makes the inspection of the residuals  $Y_i - \hat{Y}_i$  complicated. The first step is to construct some residuals  $\hat{\varepsilon}_i$  that are simpler to analyze.

The **deviance residuals** are the generalization of the residuals  $\hat{\varepsilon}_i = Y_i - \hat{Y}_i$  from the linear model. They are constructed using the analogy between the deviance and the RSS saw in (??). The deviance can be expressed into a sum of terms associated to each datum (recall e.g. (5.24)):

$$D = \sum_{i=1}^n d_i.$$

For the linear model,  $d_i = \hat{\varepsilon}_i^2$ , since  $D = \text{RSS}(\hat{\beta})$ . Based on this, we can define the deviance residuals as

$$\hat{\varepsilon}_i^D := \text{sign}(Y_i - \hat{\mu}_i) \sqrt{d_i}, \quad i = 1, \dots, n$$

and have a generalization of  $\hat{\varepsilon}_i$  for generalized linear models. This definition has interesting distributional consequences. From (5.25), we know that, asymptotically,  $D^* \sim \chi_{n-p-1}^2$ . This *suggests* that<sup>8</sup> (practical counterexamples are possible, see Figures 5.10 and 5.12)

$$\hat{\varepsilon}_i^D \text{ are approximately normal.}$$

The previous statement is of variable accuracy, depending on the model, sample size, and distribution of the predictors. In the linear model, it is exact and  $(\hat{\varepsilon}_1^D, \dots, \hat{\varepsilon}_n^D)$  are distributed *exactly* as a  $\mathcal{N}_n(\mathbf{0}, \sigma^2 \mathbf{X}'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X})$ .

The deviance residuals are key for the diagnostics of generalized linear models. Whenever we refer to “residuals”, we understand that we refer to the deviance residuals (several are possible). They are also the residuals returned in R, either by `glm$residuals` or by `residuals(glm)`.



This generalization has interesting connections:

- If the canonical function is employed, then  $\sum_{i=1}^n \hat{\varepsilon}_i^D = 0$ , as in linear models.
- The estimate of the scale parameter can be seen as  $\hat{\phi}_D = \frac{\sum_{i=1}^n (\hat{\varepsilon}_i^D)^2}{n-p-1}$ , which is perfectly coherent with  $\hat{\sigma}$  in the linear model.
- Therefore,  $\hat{\phi}_D$  is the sample variance of  $\hat{\varepsilon}_1^D, \dots, \hat{\varepsilon}_n^D$ , which suggests that  $\phi$  is the asymptotic variance of the population deviance residuals, in other words,  $\text{Var}[\varepsilon^D] \approx \phi$ .



When one assumption fails, it is likely that this failure will affect other assumptions. For example, if linearity fails, then most likely the response distribution will fail also. The key point is to identify the **root cause** of the assumptions failure in order to try to find a patch.

The script used for generating the following Figures 5.6–5.17 is available [here](#).

### 5.7.1 Linearity

Linearity between the *transformed expectation* of  $Y$  and the predictors  $X_1, \dots, X_p$  is the building block of generalized linear models. If this assumption fails, then all the conclusions we might extract from the analysis are suspected to be flawed. Therefore it is a **key assumption**.

<sup>8</sup>Remember: a  $\chi_k^2$  random variable has the same distribution as the sum of  $k$  independent squared  $\mathcal{N}(0, 1)$  variables.

### How to check it

We use the *residuals vs. fitted values plot*, which for generalized linear models is the scatterplot of  $(\hat{\eta}_i, \hat{\varepsilon}_i^D)$ ,  $i = 1, \dots, n$ . Recall that it is **not** the scatterplot of  $(\hat{Y}_i, \hat{\varepsilon}_i)$ . Under linearity, we expect that there is *no trend in the residuals*  $\hat{\varepsilon}_i^D$  with respect to  $\hat{\eta}_i$ . If nonlinearities are observed, it is worth plotting the regression terms of the model via `termplot`.

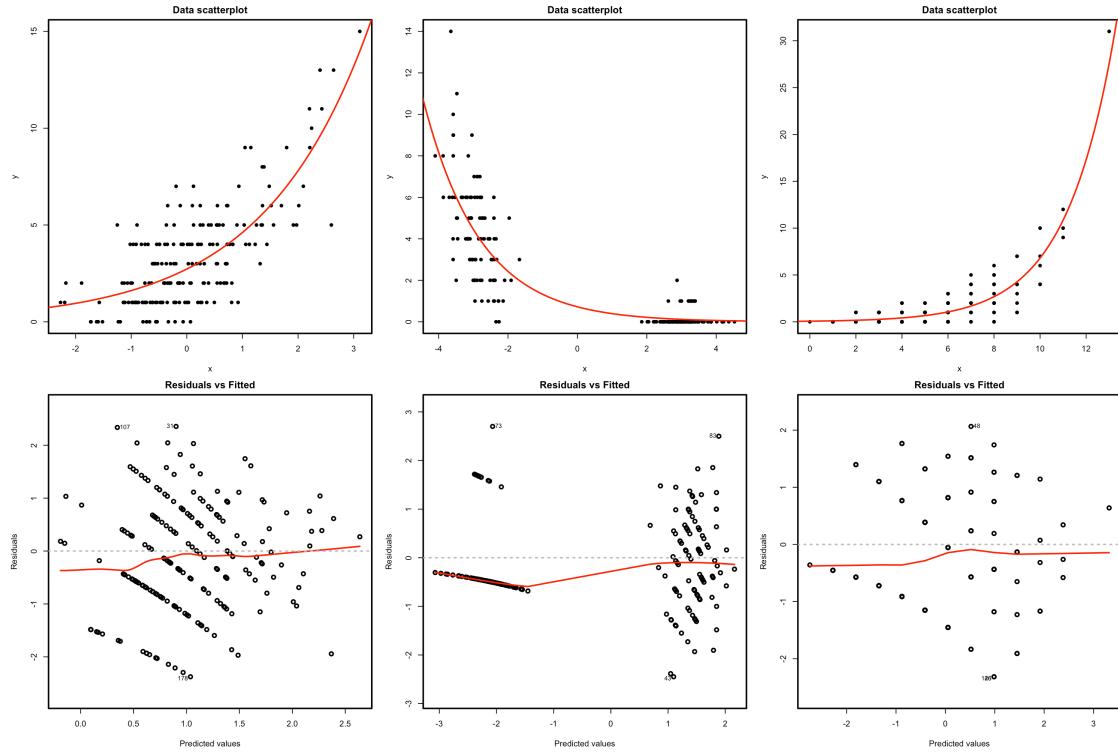


Figure 5.6: Residuals vs. fitted values plots (first row) for datasets (second row) **respecting** the linearity assumption in **Poisson** regression.

### What to do if fails

Using an adequate nonlinear transformation for the problematic predictors or adding interaction terms might be helpful. Alternatively, consider a nonlinear transformation  $f$  for the response  $Y$ .

## 5.7.2 Response distribution

The *approximate* normality in the deviance residuals allows to evaluate how well satisfied the assumption of the response distribution is. The good news is that we can do so **without relying on ad-hoc tools for each distribution**. The bad news is that we have to pay a price in terms of inexactitude, since we employ an **asymptotic distribution**. The speed of this asymptotic convergence largely depends on several aspects (distribution of the response, sample size, distribution of the predictors).

### How to check it

The *QQ-plot* allows us to check if the standardized residuals follow a  $\mathcal{N}(0, 1)$ . **Under the correct distribution of the response, we expect the points to align with the diagonal line.** It is usual to

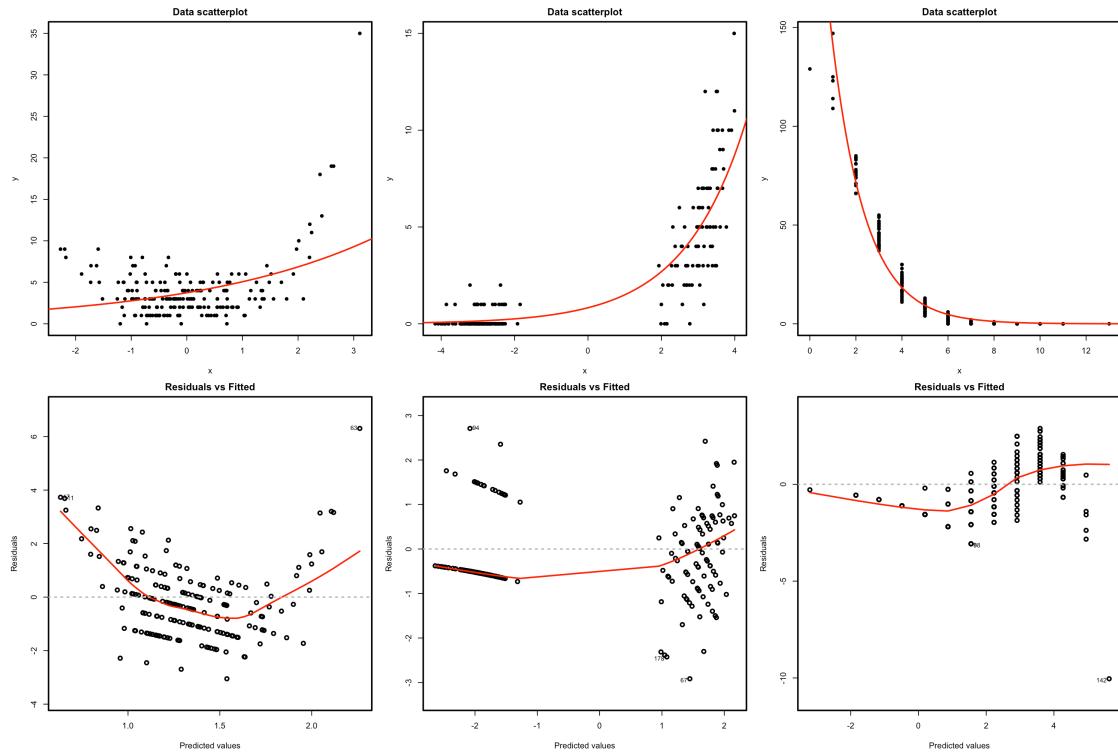


Figure 5.7: Residuals vs. fitted values plots (first row) for datasets (second row) **violating** the linearity assumption in **Poisson** regression.

have departures from the diagonal in the extremes other than in the center, even under normality, although these departures are more evident if the data is non normal. Unfortunately, it is also possible to have severe departures from normality *even* if the model is correct, see below.

Formal normality tests are available through `shapiro.test` and `lillie.test`.

### What to do if fails

Patching the distribution assumption is not easy and requires the consideration of more flexible models. One possibility is to transform  $Y$  by means of a Box-Cox transformation of the form  $\frac{Y^\lambda - 1}{\lambda}$ ,  $\lambda \neq 0$  or  $\log(Y)$ , of course at the price of modelling the transformed response rather than  $Y$ .

### 5.7.3 Independence

Independence is also a key assumption: it guarantees that the amount of information that we have on the relationship between  $Y$  and  $X_1, \dots, X_p$  with  $n$  observations is maximal.

### How to check it

The presence of *autocorrelation* in the residuals can be examined by means of a **serial plot of the residuals**.

**Under uncorrelation, we expect the series to show no tracking of the residuals**, which is a sign of positive serial correlation. Negative serial correlation can be identified in the form of a small-large or positive-negative systematic alternation of the residuals. This can be explored better with `lag.plot`, as saw in Section 3.5.4.

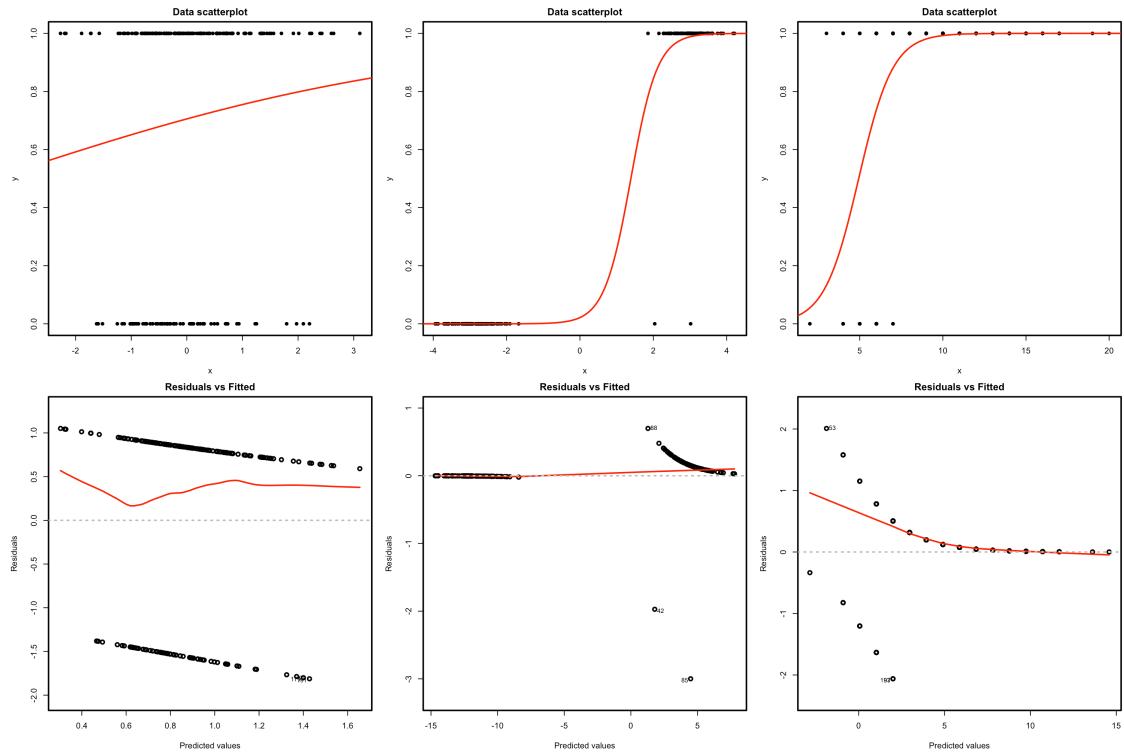


Figure 5.8: Residuals vs. fitted values plots (first row) for datasets (second row) **respecting** the linearity assumption in logistic regression.

### What to do if fails

As in the linear model, little can be done if there is dependence in the data, once this has been collected. If serial dependence is present, a differentiation of the response may lead to independent observations.

#### 5.7.4 Multicollinearity

Multicollinearity can also be present in generalized linear models. Despite the nonlinear logistic curve, **the predictors are combined linearly** in (5.4). Due to this, if two or more predictors are highly correlated between them, the fit of the model will be compromised since the individual linear effect of each predictor is hard to distinguish from the rest of correlated predictors.

The VIF of each coefficient  $\hat{\beta}_j$ . The situation is exactly the same as in linear regression, since VIF looks only into the linear relations of the predictors. Therefore, the rule of thumb is the same as in Section 3.5.5:

- VIF close to 1: absence of multicollinearity.
- **VIF larger than 5 or 10: problematic amount of multicollinearity.** Advised to remove the predictor with largest VIF.

```
# Create predictors with multicollinearity: x4 depends on the rest
set.seed(45678)
x1 <- rnorm(100)
x2 <- 0.5 * x1 + rnorm(100)
x3 <- 0.5 * x2 + rnorm(100)
x4 <- -x1 + x2 + rnorm(100, sd = 0.25)
```

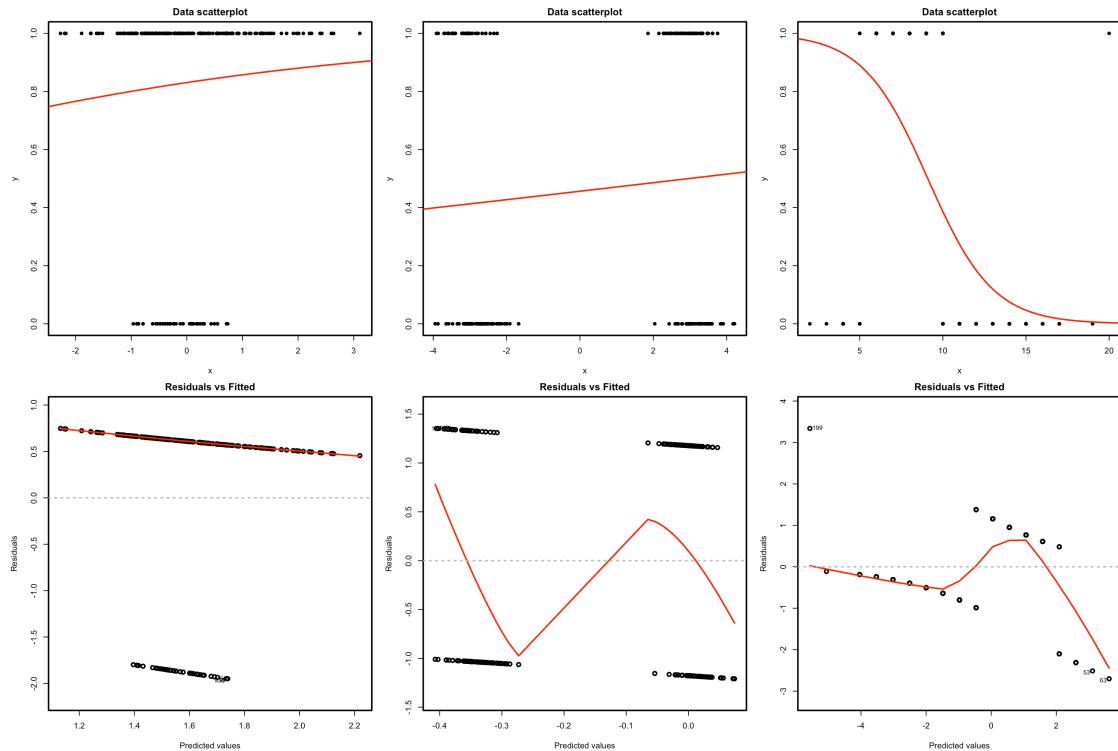


Figure 5.9: Residuals vs. fitted values plots (first row) for datasets (second row) **violating** the linearity assumption in logistic regression.

```

# Response
z <- 1 + 0.5 * x1 + 2 * x2 - 3 * x3 - x4
y <- rbinom(n = 100, size = 1, prob = 1/(1 + exp(-z)))
data <- data.frame(x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Correlations - none seems suspicious
cor(data)
##          x1          x2          x3          x4          y
## x1  1.0000000  0.3825478  0.2142011 -0.5261464  0.20198825
## x2  0.3825478  1.0000000  0.5167341  0.5673174  0.07456324
## x3  0.2142011  0.51673408 1.0000000  0.2500123 -0.49853746
## x4 -0.5261464  0.56731738  0.2500123  1.0000000 -0.11188657
## y   0.2019882  0.07456324 -0.4985375 -0.1118866  1.00000000

# Abnormal generalized variance inflation factors: largest for x4, we remove it
modMultiCo <- glm(y ~ x1 + x2 + x3 + x4, family = "binomial")
vif(modMultiCo)
##          x1          x2          x3          x4
## 27.84756 36.66514  4.94499 36.78817

# Without x4
modClean <- glm(y ~ x1 + x2 + x3, family = "binomial")

# Comparison
summary(modMultiCo)

```

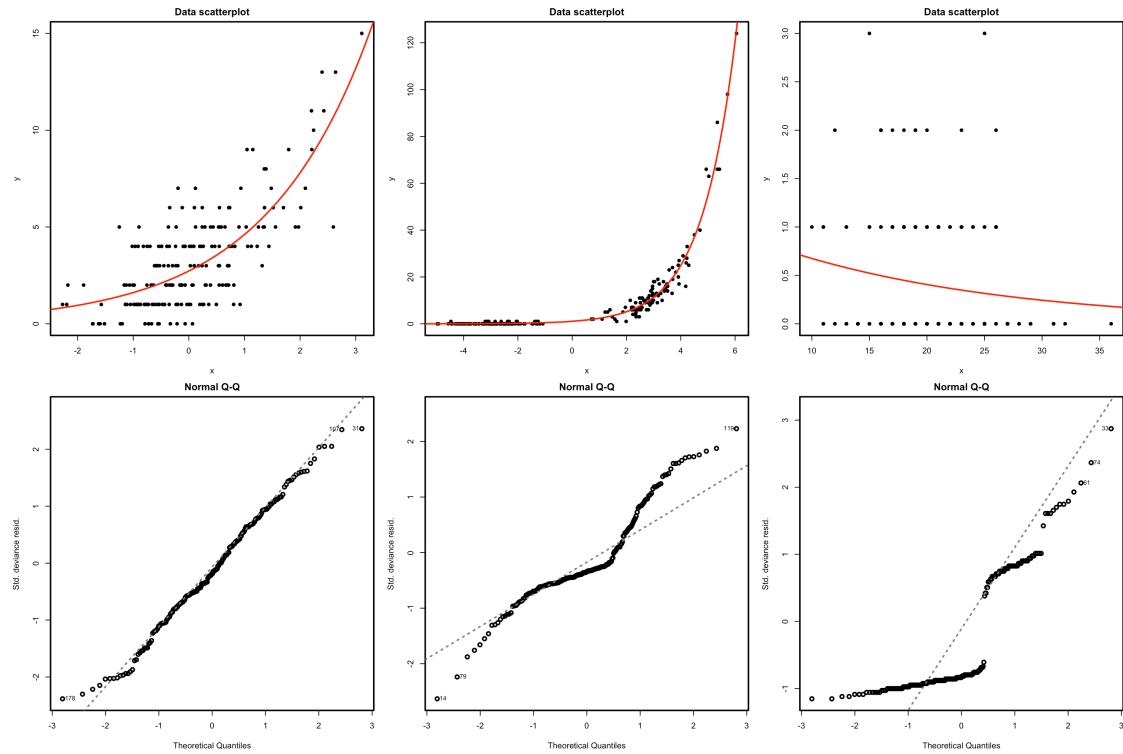


Figure 5.10: QQ-plots for the deviance residuals (first row) for datasets (second row) respecting the response distribution assumption for **Poisson** regression.

```
##  
## Call:  
## glm(formula = y ~ x1 + x2 + x3 + x4, family = "binomial")  
##  
## Deviance Residuals:  
##      Min        1Q     Median        3Q       Max  
## -2.4743  -0.3796   0.1129   0.4052   2.3887  
##  
## Coefficients:  
##             Estimate Std. Error z value Pr(>|z|)  
## (Intercept)  1.2527   0.4008   3.125  0.00178 **  
## x1          -3.4269   1.8225  -1.880  0.06007 .  
## x2           6.9627   2.1937   3.174  0.00150 **  
## x3          -4.3688   0.9312  -4.691 2.71e-06 ***  
## x4          -5.0047   1.9440  -2.574  0.01004 *  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## (Dispersion parameter for binomial family taken to be 1)  
##  
## Null deviance: 132.81  on 99  degrees of freedom  
## Residual deviance: 59.76  on 95  degrees of freedom  
## AIC: 69.76  
##  
## Number of Fisher Scoring iterations: 7
```

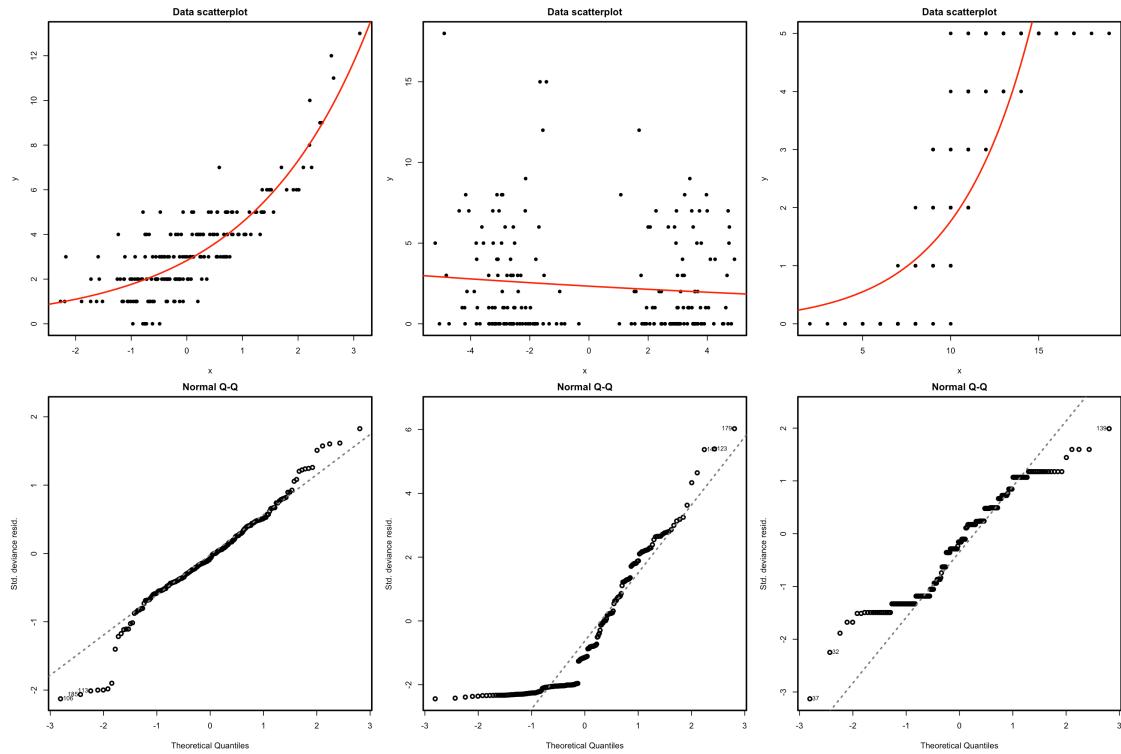


Figure 5.11: QQ-plots for the deviance residuals (first row) for datasets (second row) **violating** the response distribution assumption for **Poisson** regression.

```
summary(modClean)
##
## Call:
## glm(formula = y ~ x1 + x2 + x3, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.0952 -0.4144  0.1839  0.4762  2.5736
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.9237    0.3221  2.868 0.004133 ***
## x1          1.2803    0.4235  3.023 0.002502 ***
## x2          1.7946    0.5290  3.392 0.000693 ***
## x3         -3.4838    0.7491 -4.651 3.31e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 132.813  on 99  degrees of freedom
## Residual deviance: 68.028  on 96  degrees of freedom
## AIC: 76.028
##
## Number of Fisher Scoring iterations: 6
```

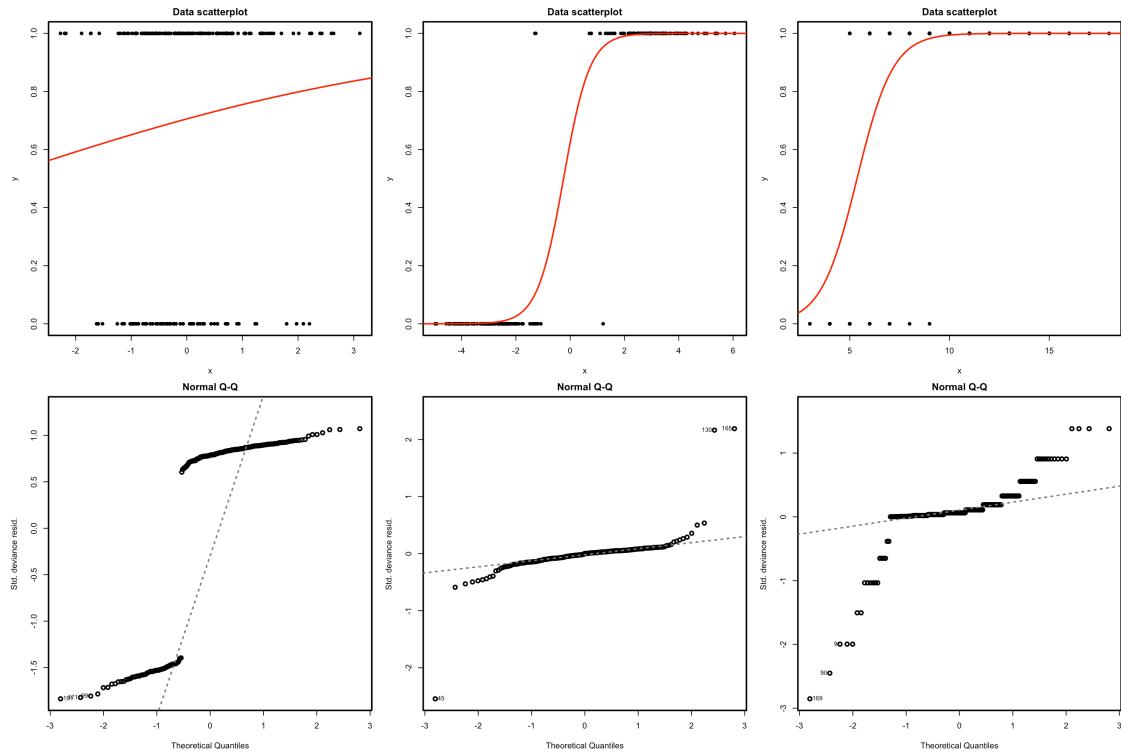


Figure 5.12: QQ-plots for the deviance residuals (first row) for datasets (second row) **respecting** the response distribution assumption for **logistic** regression.

```
# Generalized variance inflation factors normal
vif(modClean)
##      x1      x2      x3
## 1.674300 2.724351 3.743940
```



Performing **PCA on the predictors**, as seen in Section 3.6.2, is a possibility to achieve **uncorrelation** and can be employed straightforwardly in generalized linear models. The story is different for PLS, since it makes use of the linear structure between the response and the predictors and thus is not immediately adaptable to generalized linear models.

## 5.8 Shrinkage

Enforcing sparsity in generalized linear models can be done as it was done with linear models. Ridge regression and Lasso can be generalized with **glmnet** with little differences in practice.

What we want is to bias the estimates of  $\beta$  towards being non-null only in the most important relations between the response and predictors. To achieve that, we add a *penalization* term to the maximum likelihood estimation of  $\beta$ <sup>9</sup>:

<sup>9</sup>We are minimizing the negative log-likelihood  $\ell(\beta)$ .

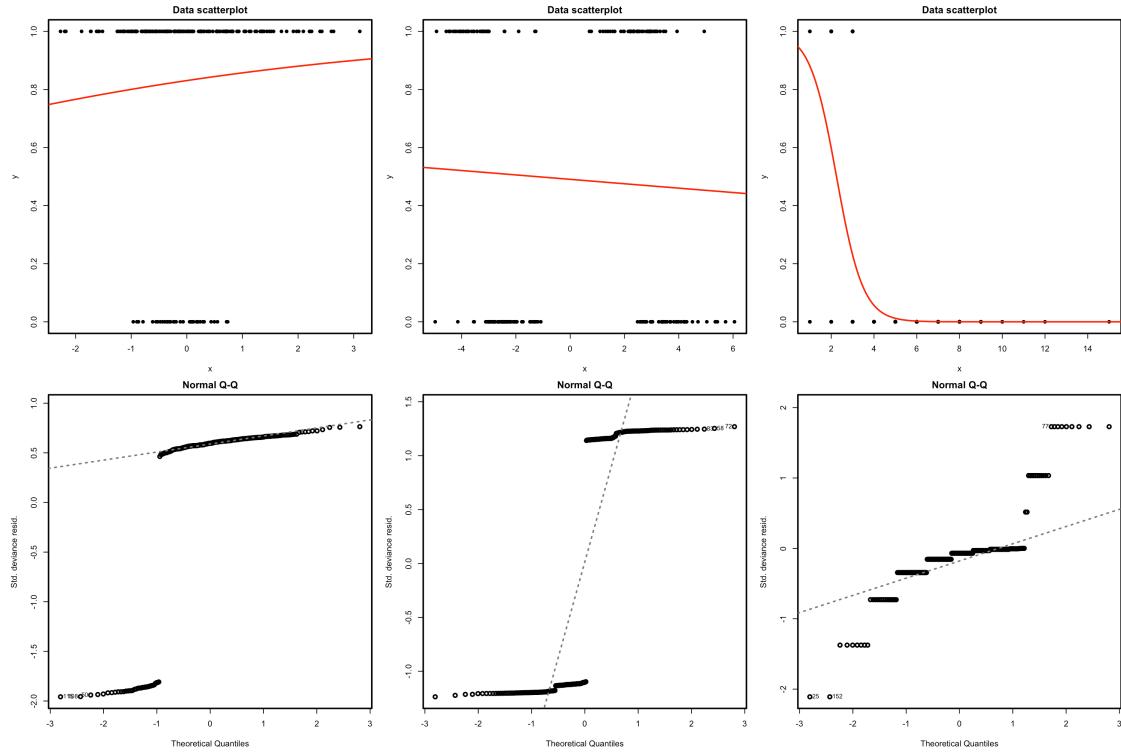


Figure 5.13: QQ-plots for the deviance residuals (first row) for datasets (second row) **violating** the response distribution assumption for **logistic** regression.

$$-\ell(\beta) + \lambda \sum_{j=1}^p |\beta_j|^{(2-\alpha)}. \quad (5.30)$$

As in Section 4.1, **ridge regression corresponds to**  $\alpha = 1$  (quadratic penalty) and **lasso to**  $\alpha = 0$  (linear penalty). Obviously, if  $\lambda = 0$ , we are back to the generalized linear models theory. The optimization of (5.30) gives

$$\hat{\beta}_\lambda := \arg \min_{\beta \in \mathbb{R}^{p+1}} \left\{ -\ell(\beta) + \lambda \sum_{j=1}^p |\beta_j|^{(2-\alpha)} \right\}. \quad (5.31)$$

Note that the sparsity is enforced in the slopes, not in the intercept, and that the link function  $g$  is not affecting the penalization term. Remember that the *predictors need to be standardized* if they are of different nature.

We illustrate the shrinkage in generalized linear models with the **Hitters** dataset, where now the objective will be to predict **NewLeague**, a factor with levels A (standing for *American League*) and N (standing for *National League*). The variable indicates the player's league at the end of 1986. The predictors employed are his statistics during 1986, and the objective is to see whether there is some distinctive pattern between the players in both leagues.

```
# Load data
library(ISLR)
```

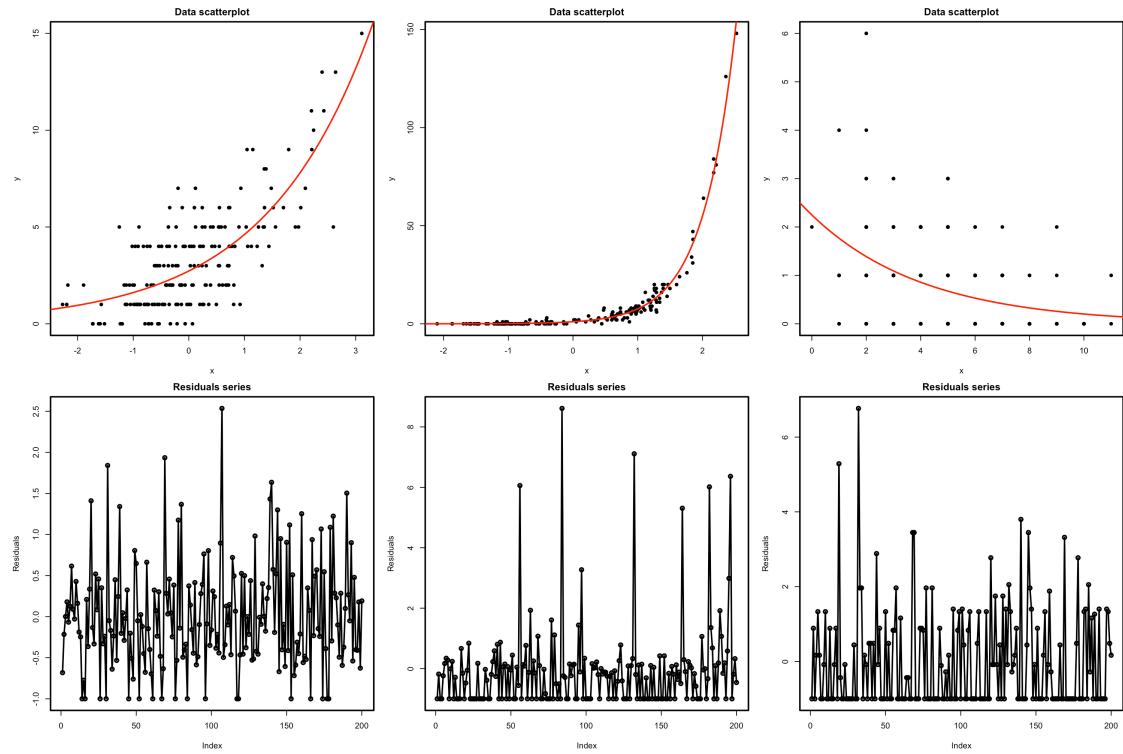


Figure 5.14: Serial plots of the residuals (first row) for datasets (second row) respecting the independence assumption for **Poisson** regression.

```

data(Hitters)

# Include only predictors related with 1986 season and discard NA's
Hitters <- subset(Hitters, select = c(League, AtBat, Hits, HmRun, Runs, RBI,
                                         Walks, Division, PutOuts, Assists, Errors))
Hitters <- na.omit(Hitters)

# Response and predictors
y <- Hitters$League
x <- model.matrix(League ~ ., data = Hitters)[, -1]

```

After preparing the data, we perform the regressions.

```

# Ridge and lasso regressions
library(glmnet)
ridgeMod <- glmnet(x = x, y = y, alpha = 0, family = "binomial")
lassoMod <- glmnet(x = x, y = y, alpha = 1, family = "binomial")

# Solution paths versus lambda
plot(ridgeMod, label = TRUE, xvar = "lambda")

```

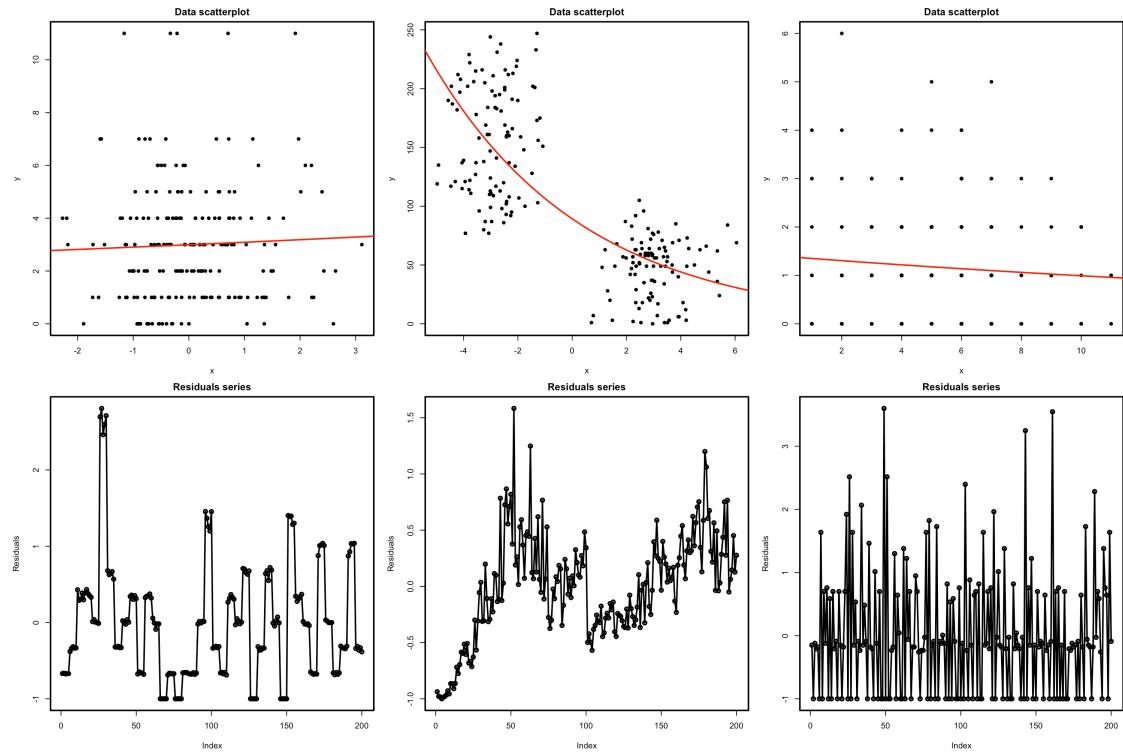
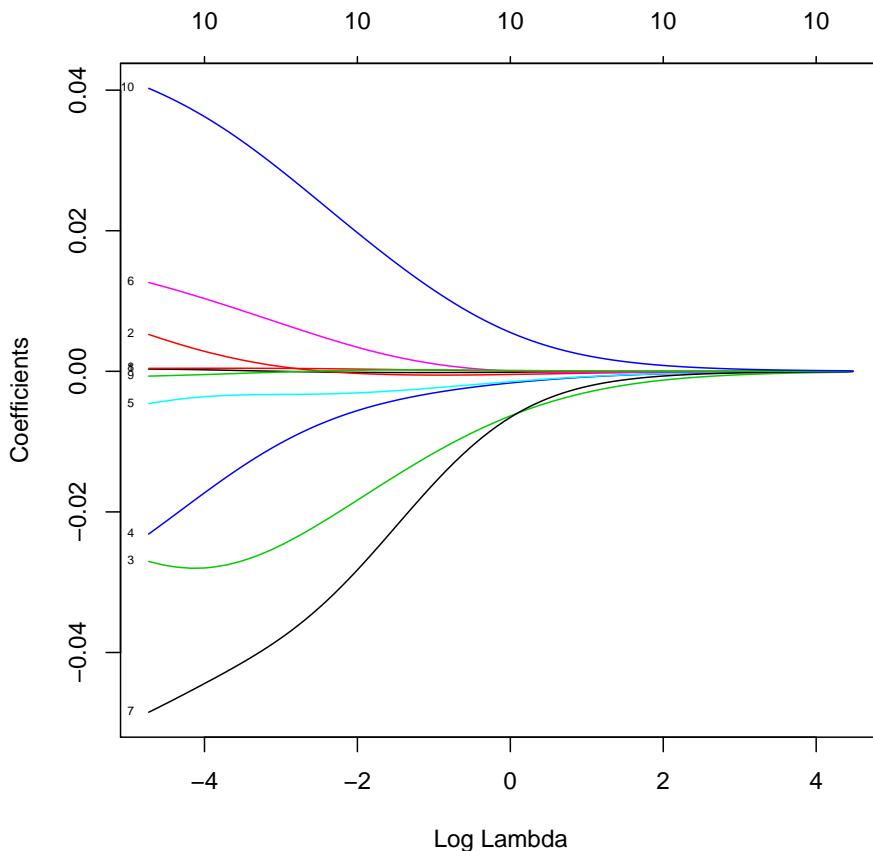


Figure 5.15: Serial plots of the residuals (first row) for datasets (second row) **violating** the independence assumption for **Poisson** regression.



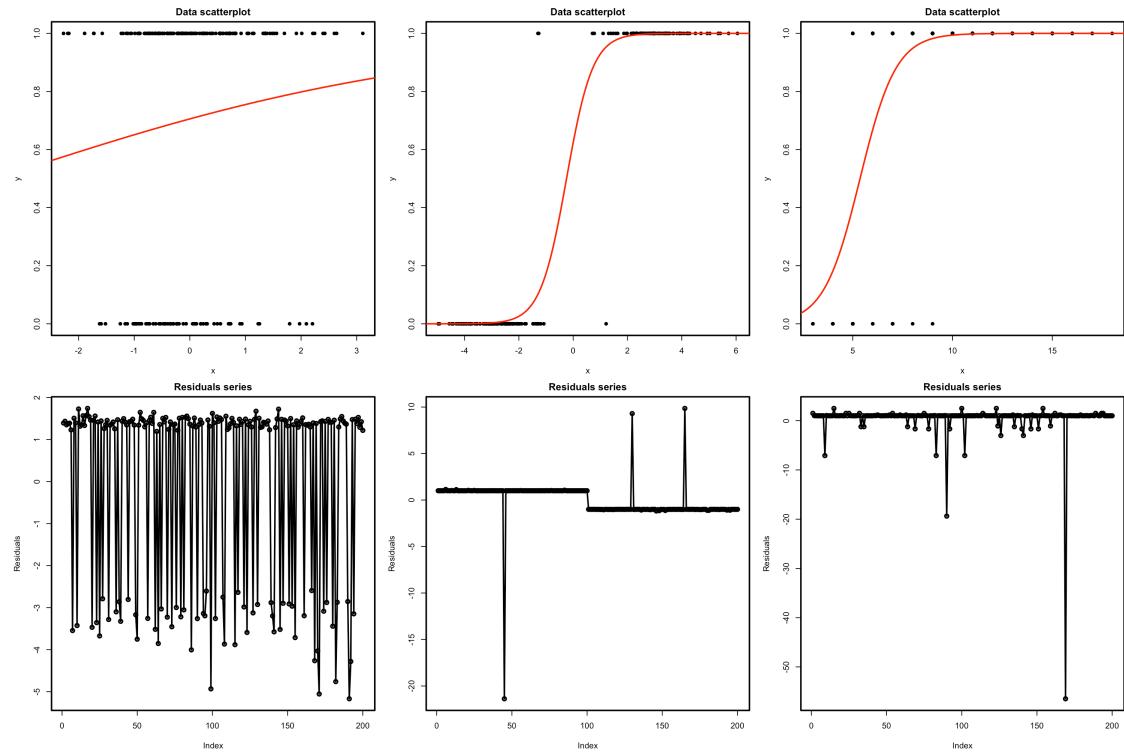


Figure 5.16: Serial plots of the residuals (first row) for datasets (second row) **respecting** the independence assumption for **logistic** regression.

```
plot(lassoMod, label = TRUE, xvar = "lambda")
```

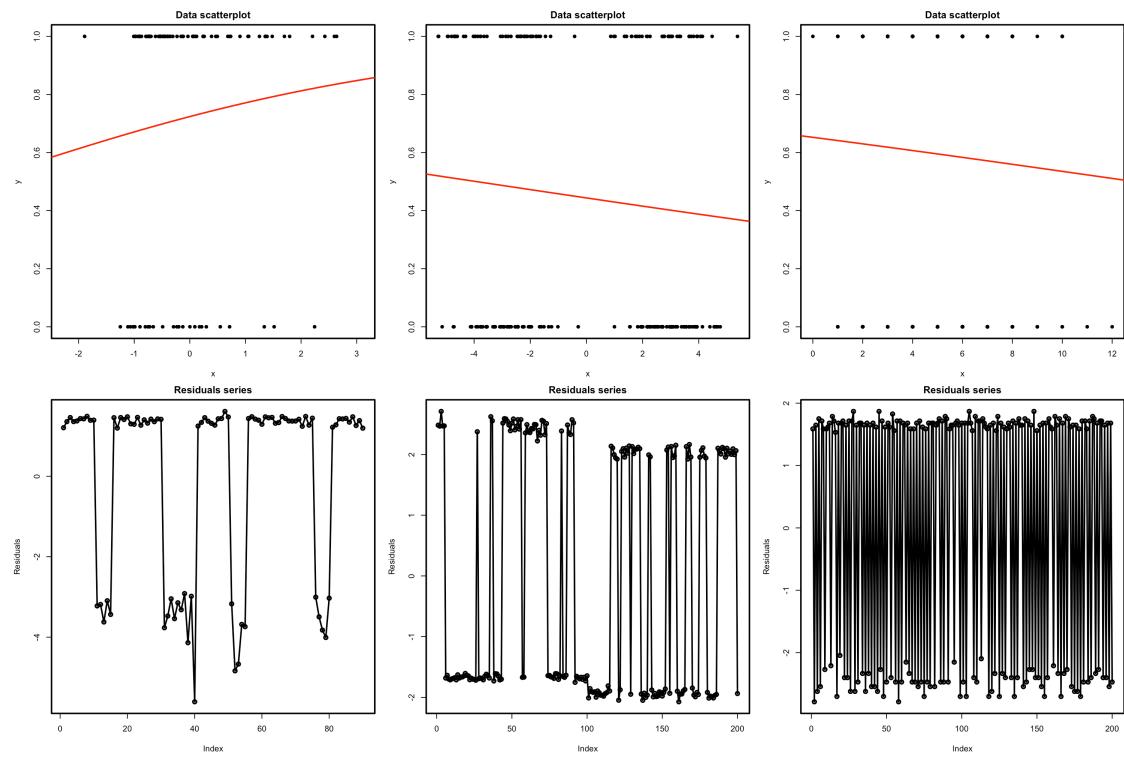
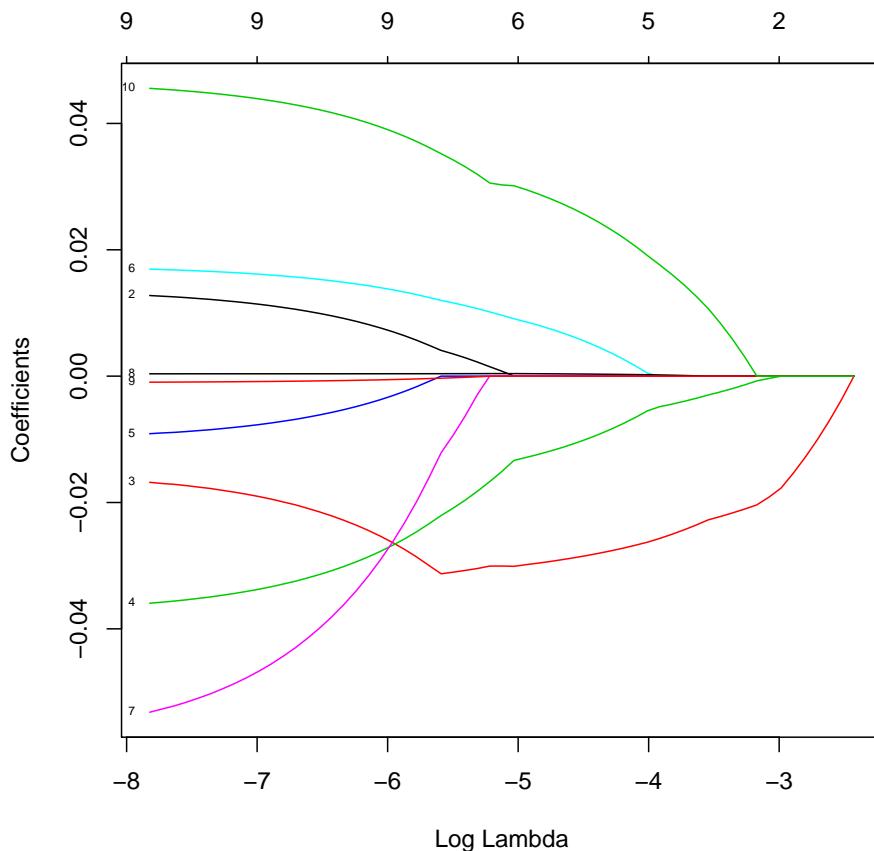
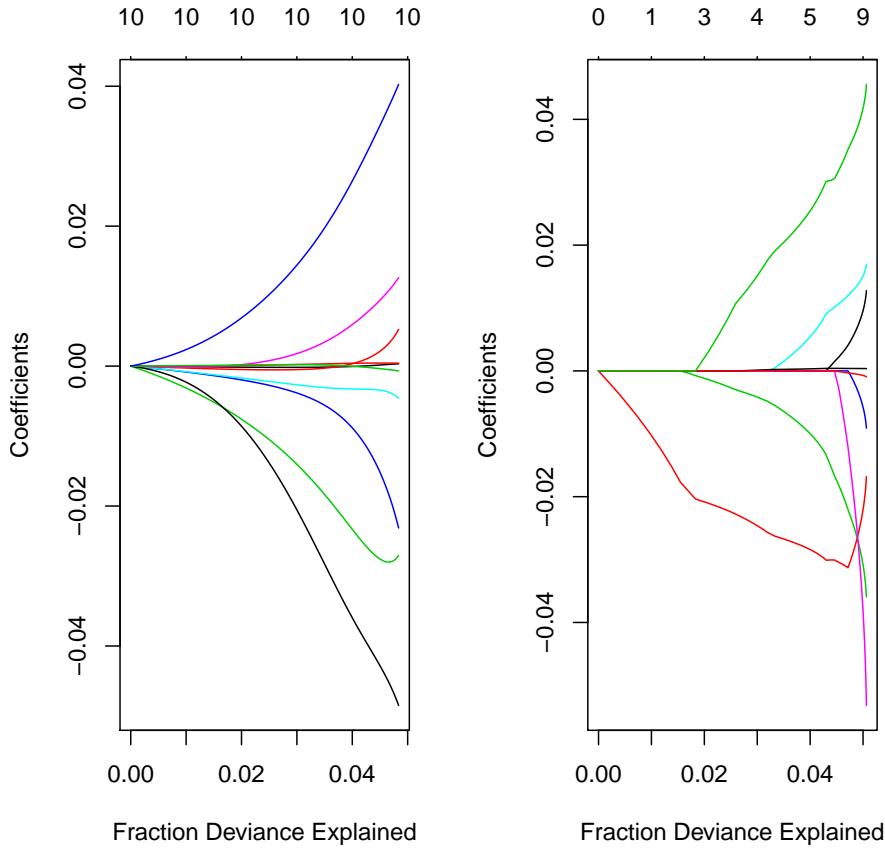


Figure 5.17: Serial plots of the residuals (first row) for datasets (second row) **violating** the independence assumption for **logistic** regression.



```
# Versus the percentage of deviance explained
par(mfrow = c(1, 2))
plot(ridgeMod, label = TRUE, xvar = "dev")
plot(lassoMod, label = TRUE, xvar = "dev")
```

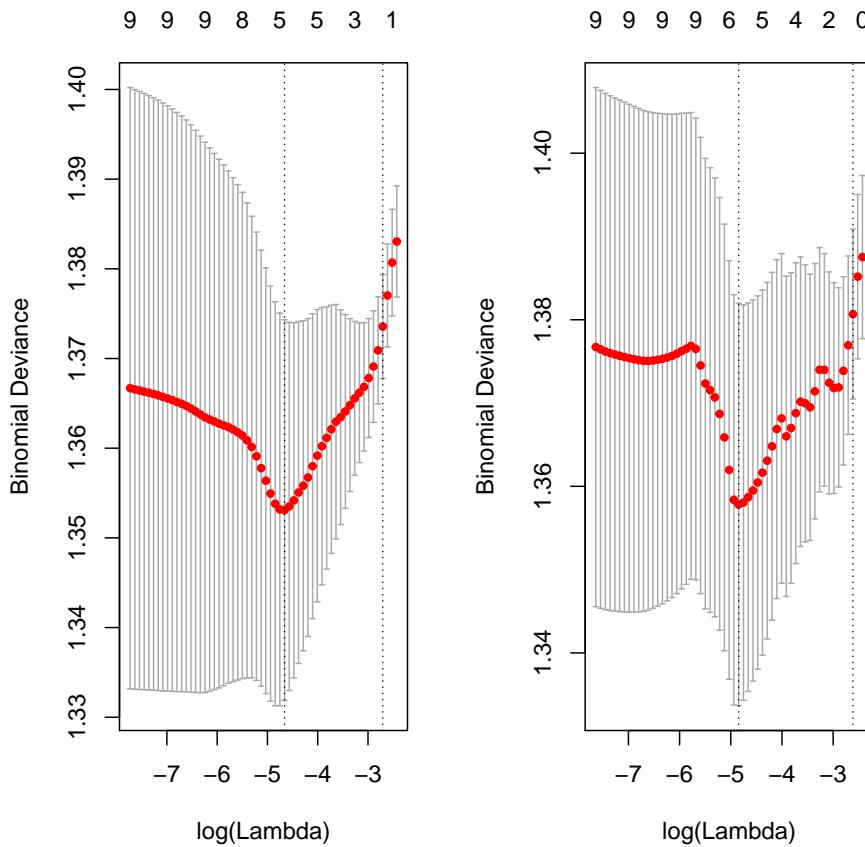


```
# The percentage of deviance explained only goes up to 0.05. There are no
# clear patterns indicating player differences between both leagues

# Let's select the predictors to be included with a 10-fold cross-validation
set.seed(12345)
kcvLasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = 10, family = "binomial")
plot(kcvLasso)

# The lambda that minimises the CV error and "one standard error rule"'s lambda
kcvLasso$lambda.min
## [1] 0.009467416
kcvLasso$lambda.1se
## [1] 0.06679075

# Leave-one-out cross-validation - similar result
ncvLasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = nrow(Hitters),
                       family = "binomial")
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
plot(ncvLasso)
```



```

ncvLasso$lambda.min
## [1] 0.007860015
ncvLasso$lambda.1se
## [1] 0.07330276

# Model selected
predict(ncvLasso, type = "coefficients", s = ncvLasso$lambda.1se)
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept) -0.099447861
## AtBat          .
## Hits          .
## HmRun         -0.006971231
## Runs          .
## RBI           .
## Walks          .
## DivisionW     .
## PutOuts        .
## Assists        .
## Errors         .

```

`HmRun` is selected by leave-one-out cross-validation as the unique predictor to be included in the lasso regression. We know that the model is not good due to the percentage of deviance explained. However, we still want to know whether `HmRun` has any significance at all. When addressing this, we have to take into account Section 4.2 to avoid spurious findings.

```

# Analyse the selected model)
fit <- glm(League ~ HmRun, data = Hitters, family = "binomial")

```

```

summary(fit)
##
## Call:
## glm(formula = League ~ HmRun, family = "binomial", data = Hitters)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q      Max
## -1.2976 -1.1320 -0.8106  1.1686  1.6440
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.27826   0.18086  1.539  0.12392
## HmRun       -0.04290   0.01371 -3.130  0.00175 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 443.95 on 321 degrees of freedom
## Residual deviance: 433.57 on 320 degrees of freedom
## AIC: 437.57
##
## Number of Fisher Scoring iterations: 4
# HmRun is significant - but it may be spurious due to the model selection
# procedure (see Section 4.2)

# Let's split the dataset in two, do model-selection in one part and then
# inference on the selected model in the other, to have an idea of the real
# significance of HmRun
set.seed(123456)
train <- sample(c(FALSE, TRUE), size = nrow(Hitters), replace = TRUE)

# Model selection in training part
ncvLasso <- cv.glmnet(x = x[train, ], y = y[train], alpha = 1, nfolds = sum(train),
                       family = "binomial")
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
predict(ncvLasso, type = "coefficients", s = ncvLasso$lambda.1se)
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                1
## (Intercept) 0.072711333
## AtBat       .
## Hits        .
## HmRun      -0.009980663
## Runs        .
## RBI         .
## Walks       .
## DivisionW   .
## PutOuts     .
## Assists     .
## Errors      .

# Inference in testing part

```

```

summary(glm(League ~ HmRun, data = Hitters[!train, ], family = "binomial"))
##
## Call:
## glm(formula = League ~ HmRun, family = "binomial", data = Hitters[!train,
##   ])
##
## Deviance Residuals:
##       Min      1Q  Median      3Q     Max
## -1.1542 -1.0728 -0.9213  1.2702  1.5269
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.05490   0.26495 -0.207   0.836
## HmRun        -0.02378   0.01829 -1.300   0.194
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 214.74 on 157 degrees of freedom
## Residual deviance: 213.00 on 156 degrees of freedom
## AIC: 217
##
## Number of Fisher Scoring iterations: 4
# HmRun is now not significant...
# We can repeat the analysis for different partitions of the data and we will
# obtain weak significances. Therefore, we can conclude that this is an spurious
# finding and that HmRun is not significant as a single predictor

# Prediction (obviously not trustable, but for illustration)
pred <- predict(ncvLasso, newx = x[!train, ], type = "response",
                 s = ncvLasso$lambda.1se)

# Hit matrix
H <- table(pred > 0.5, y[!train] == "A") # ("A" was the reference level)
H
##
##          FALSE TRUE
## FALSE     34   55
## TRUE      32   37
sum(diag(H)) / sum(H) # Worse than tossing a coin!
## [1] 0.4493671

```



Perform an adequate statistical analysis based on shrinkage of a generalized linear model to reply the following questions:

- What (if any) are the leading factors among the features of a player in season 1986 in order to be in the top 10% of most paid players in season 1987?
- What (if any) are the player features in season 1986 influencing the number of home runs in the same season? And during his career?

*Hint:* you may use the one shown in the section as a template.

## 5.9 Big data considerations

As we saw in Section 5.2.2, fitting a generalized linear model involves fitting a series of linear models. Therefore, all the memory problems that appeared in Section 4.3 are inherited. Worse, computation is now more complicated because:

1. *Computing the likelihood requires reading all the data at once.* Differently from the linear model, updating the model with a new chunk implies re-fitting with all the data due to the nonlinearity of the likelihood.
2. The IRLS algorithm *requires reading the data as many times as iterations.*

These two peculiarities are a game-changer for the approach followed in Section 4.3: `biglm`'s fitting function for generalized linear models, `bigglm`, needs to have access to the full data while performing the fitting. This can be cumbersome.

Hopefully, a neat solution is available using the `ff` and `ffbase` packages, which allow for efficiently *working with data stored in disk that behave (almost) as if they were in RAM*. The function that we will employ is `ffbase`'s `bigglm.ffd`, and requires from an object of the class `ffd` (`ff`'s data frames).

```
# Not really "big data", but for the sake of illustration
set.seed(12345)
n <- 1e6
p <- 10
beta <- seq(-1, 1, length.out = p)^5

x1 <- matrix(rnorm(n * p), nrow = n, ncol = p)
x1[, p] <- 2 * x1[, 1] + rnorm(n, sd = 0.1) # Add some dependence to predictors
x1[, p - 1] <- 2 - x1[, 2] + rnorm(n, sd = 0.5)
y1 <- rbinom(n, size = 1, prob = 1 / (1 + exp(-(1 + x1 %*% beta))))
x2 <- matrix(rnorm(100 * p), nrow = 100, ncol = p)
y2 <- rbinom(100, size = 1, prob = 1 / (1 + exp(-(1 + x2 %*% beta))))
bigData1 <- data.frame("resp" = y1, "pred" = x1)
bigData2 <- data.frame("resp" = y2, "pred" = x2)

# Save files to disk to emulate the situation with big data
write.csv(x = bigData1, file = "bigData1.csv", row.names = FALSE)
write.csv(x = bigData2, file = "bigData2.csv", row.names = FALSE)

# Read files using ff
library(ffbase) # Imports ff
bigData1ff <- read.table.ffd(file = "bigData1.csv", header = TRUE, sep = ",")
bigData2ff <- read.table.ffd(file = "bigData2.csv", header = TRUE, sep = ",")

# Recall: bigData1.csv is not copied into RAM
print(object.size(bigData1), units = "Mb")
## 80.1 Mb
print(object.size(bigData1ff), units = "Kb")
## 38.7 Kb

# Delete the csv files in disk
file.remove(c("bigData1.csv", "bigData2.csv"))
## [1] TRUE TRUE

# Logistic regression
library(biglm)
# Same comments for the formula framework - this is the hack for automatic
```

```

# inclusion of all the predictors
f <- formula(paste("resp ~", paste(names(bigData1)[-1], collapse = " + ")))
bigglmMod <- bigglm.ffdff(formula = f, data = bigData1ff, family = binomial())

# glm's call
glmMod <- glm(formula = resp ~ ., data = bigData1, family = binomial())

# Compare sizes
print(object.size(bigglmMod), units = "Kb")
## 173.1 Kb
print(object.size(glmMod), units = "Mb")
## 679.2 Mb

# Summaries
s1 <- summary(bigglmMod)
s2 <- summary(glmMod)
s1
## Large data regression model: bigglm(formula = f, data = bigData1ff, family = binomial())
## Sample size = 1e+06
##          Coef      (95%      CI)      SE      p
## (Intercept) 1.0177  0.9960  1.0394 0.0109 0.0000
## pred.1      -0.9869 -1.0931 -0.8808 0.0531 0.0000
## pred.2      -0.2927 -0.3046 -0.2808 0.0059 0.0000
## pred.3      -0.0481 -0.0534 -0.0428 0.0027 0.0000
## pred.4      -0.0029 -0.0082  0.0024 0.0026 0.2779
## pred.5      -0.0015 -0.0068  0.0038 0.0027 0.5708
## pred.6      -0.0022 -0.0075  0.0031 0.0026 0.4162
## pred.7      0.0018 -0.0035  0.0071 0.0027 0.4876
## pred.8      0.0582  0.0529  0.0635 0.0027 0.0000
## pred.9      0.2747  0.2640  0.2853 0.0053 0.0000
## pred.10     0.9923  0.9392  1.0454 0.0265 0.0000
s2
##
## Call:
## glm(formula = resp ~ ., family = binomial(), data = bigData1)
##
## Deviance Residuals:
##      Min      1Q      Median      3Q      Max
## -3.4234  0.2112  0.4591  0.6926  2.5853
##
## Coefficients:
##          Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.017719  0.010856  93.744  <2e-16 ***
## pred.1      -0.986934  0.053075 -18.595  <2e-16 ***
## pred.2      -0.292675  0.005947 -49.214  <2e-16 ***
## pred.3      -0.048077  0.002653 -18.120  <2e-16 ***
## pred.4      -0.002875  0.002650  -1.085   0.278
## pred.5      -0.001503  0.002650  -0.567   0.571
## pred.6      -0.002154  0.002650  -0.813   0.416
## pred.7      0.001840  0.002651   0.694   0.488
## pred.8      0.058182  0.002653  21.927  <2e-16 ***
## pred.9      0.274651  0.005320  51.629  <2e-16 ***
## pred.10     0.992299  0.026541  37.388  <2e-16 ***

```

```

## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 1059853  on 999999  degrees of freedom
## Residual deviance:  885895  on 999989  degrees of freedom
## AIC: 885917
##
## Number of Fisher Scoring iterations: 5

# Further information
s1$mat # Coefficients and their inferences
##             Coef      (95%          CI)          SE
## (Intercept) 1.017718995 0.996006224 1.039431766 0.010856386
## pred.1      -0.986933994 -1.093083429 -0.880784558 0.053074718
## pred.2      -0.292674675 -0.304568670 -0.280780679 0.005946998
## pred.3      -0.048076842 -0.053383444 -0.042770241 0.002653301
## pred.4      -0.002875381 -0.008174848  0.002424085 0.002649733
## pred.5      -0.001502598 -0.006803348  0.003798152 0.002650375
## pred.6      -0.002154396 -0.007453619  0.003144826 0.002649611
## pred.7      0.001839965 -0.003462032  0.007141962 0.002650999
## pred.8      0.058182131  0.052875269  0.063488994 0.002653431
## pred.9      0.274650557  0.264011111  0.285290003 0.005319723
## pred.10     0.992299439  0.939217602  1.045381276 0.026540919
##             p
## (Intercept) 0.000000e+00
## pred.1      3.515228e-77
## pred.2      0.000000e+00
## pred.3      2.230685e-73
## pred.4      2.778513e-01
## pred.5      5.707564e-01
## pred.6      4.161613e-01
## pred.7      4.876415e-01
## pred.8      1.431803e-106
## pred.9      0.000000e+00
## pred.10     6.230296e-306
s1$rsq # R^2
## [1] 0.2175508
s1$nullrss # Null deviance
## [1] 1132208

# Extract coefficients
coef(bigglmMod)
## (Intercept)      pred.1      pred.2      pred.3      pred.4
## 1.017718995 -0.986933994 -0.292674675 -0.048076842 -0.002875381
##           pred.5      pred.6      pred.7      pred.8      pred.9
## -0.001502598 -0.002154396  0.001839965  0.058182131  0.274650557
##           pred.10
## 0.992299439

# Prediction works as usual
predict(bigglmMod, newdata = bigData2[1:5, ], type = "response")

```

```

##      [,1]
## 1 0.9603955
## 2 0.7434756
## 3 0.6632871
## 4 0.6188387
## 5 0.6418678
# predict(bigglmMod, newdata = bigData2[1:5, -1]) # Error

# Update the model with training data
update(bigglmMod, moredata = bigData2)
## Large data regression model: bigglm(formula = f, data = bigData1ff, family = binomial())
## Sample size = 1000100

# AIC and BIC
AIC(bigglmMod, k = 2)
## [1] 885917.2
AIC(bigglmMod, k = log(n))
## [1] 886047.2

```



Note that this is also a perfectly valid approach for linear models, we just need to specify `family = gaussian()` in the call to `bigglm.ffdf`.

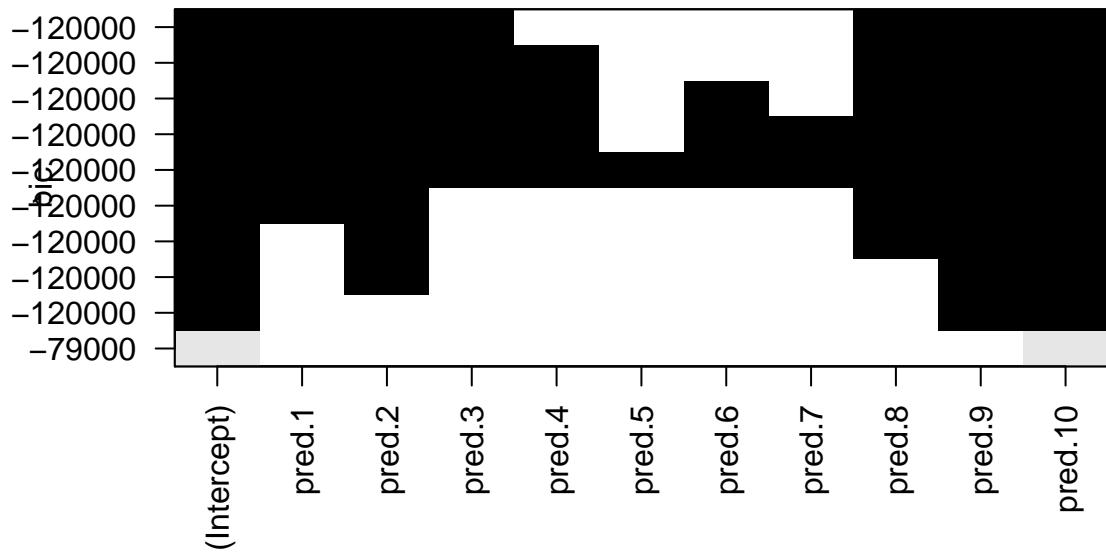
Model selection of `bigglm` is not so straightforward. The trick that `regsubsets` employs for simplifying the model search in linear models (see Section 4.3) does not apply for generalized linear models. However, there is a hack. We can do **best subset selection** in the **linear model associated to the last iteration of the IRLS algorithm** and then refine the search by computing the exact BIC/AIC from a set of candidate models. If we do so, we translate the model selection problem back to the linear case, plus the extra overhead of fitting several generalized linear models. Keep in mind that, albeit useful, this approach is an **approximation**.

```

# Model selection adapted to big data generalized linear models
library(leaps)
reg <- regsubsets(bigglmMod, nvmax = p + 1, method = "exhaustive")
# This takes the QR decomposition, which encodes the linear model associated to
# the last iteration of the IRLS algorithm. However, the reported BICs are *not*
# the true BICs of the generalized linear models, but a sufficient
# approximation to obtain a list of candidate models in a fast way

# Get the model with lowest BIC
plot(reg)

```



```

subs <- summary(reg)
subs$which
##      (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5 pred.6 pred.7 pred.8
## 1      TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2      TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 3      TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## 4      TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## 5      TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## 6      TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
## 7      TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE
## 8      TRUE TRUE TRUE TRUE TRUE FALSE TRUE FALSE TRUE
## 9      TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE
## 10     TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##      pred.9 pred.10
## 1     FALSE    TRUE
## 2     TRUE    TRUE
## 3     TRUE    TRUE
## 4     TRUE    TRUE
## 5     TRUE    TRUE
## 6     TRUE    TRUE
## 7     TRUE    TRUE
## 8     TRUE    TRUE
## 9     TRUE    TRUE
## 10    TRUE    TRUE
subs$bic
## [1] -79219.27 -118848.59 -121238.63 -121703.97 -122033.75 -122347.61
## [7] -122334.97 -122321.82 -122308.48 -122294.99
subs$which[which.min(subs$bic), ]
## (Intercept)      pred.1      pred.2      pred.3      pred.4      pred.5
##      TRUE      TRUE      TRUE      TRUE      FALSE      FALSE
##      pred.6      pred.7      pred.8      pred.9      pred.10
##      FALSE     FALSE      TRUE      TRUE      TRUE

# Let's compute the true BICs for the p models. This implies fitting p bigglm's
bestModels <- list()
for (i in 1:nrow(subs$which)) {

```

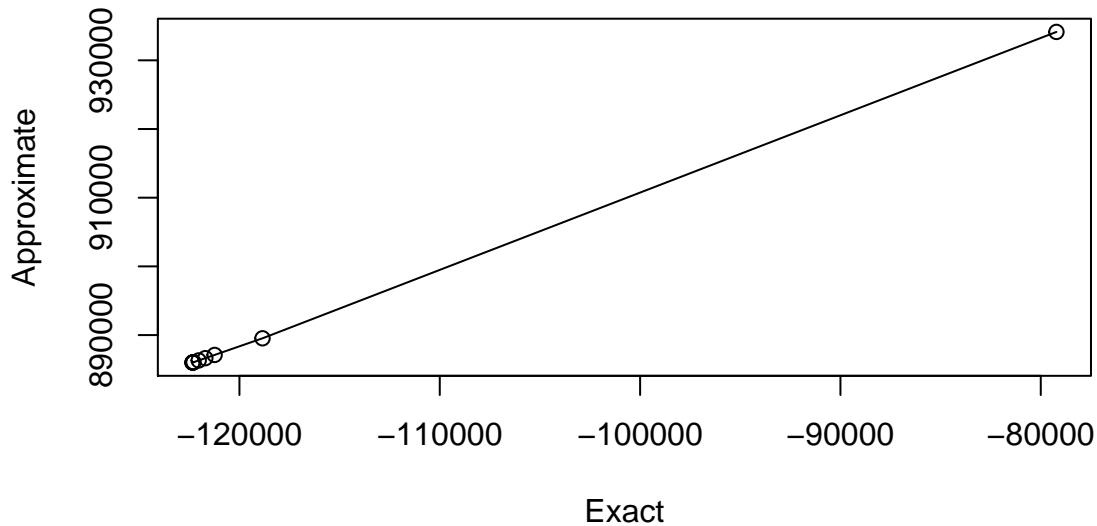
```

f <- formula(paste("resp ~", paste(names(which(subs$which[i, -1])), 
                                collapse = " + ")))
bestModels[[i]] <- bigglm.ffdf(formula = f, data = bigData1ff,
                                family = binomial(), maxit = 20)
# Did not converge with the default iteration limit, maxit = 8

}

# The approximate BICs and the true BICs are very similar (in this example)
exactBICs <- sapply(bestModels, AIC, k = log(n))
plot(subs$bic, exactBICs, type = "o", xlab = "Exact", ylab = "Approximate")

```



```

cor(subs$bic, exactBICs, method = "pearson") # Correlation
## [1] 0.9999708

# Both give the same model selection and same order
subs$which[which.min(subs$bic), ] # Approximate
## (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5
## TRUE TRUE TRUE TRUE FALSE FALSE
## pred.6 pred.7 pred.8 pred.9 pred.10
## FALSE FALSE TRUE TRUE TRUE
subs$which[which.min(exactBICs), ] # Exact
## (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5
## TRUE TRUE TRUE TRUE FALSE FALSE
## pred.6 pred.7 pred.8 pred.9 pred.10
## FALSE FALSE TRUE TRUE TRUE
cor(subs$bic, exactBICs, method = "spearman") # Order correlation
## [1] 1

```



# Chapter 6

## Nonparametric regression

The models we saw in the previous chapters share a common root: all of them are **parametric**. This means that they *assume* a certain structure on the regression function  $m$ , which is controlled by *parameters*. For example, generalized linear models assume that  $m$  is of the form  $m(\mathbf{x}) = g^{-1}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p)$  for some unknown coefficients  $\beta$  that have to be estimated. If this assumption holds (i.e., if  $m$  truly has the assumed form), then parametric methods are the best approach for estimating  $m$ . As we have seen, in practice it is rarely the case where parametric methods work out-of-the-box, and several tricks are needed in order to expand their degree of flexibility in a case-by-case basis. This is the strongest point of **nonparametric** methods: they do not assume major hard-to-satisfy hypothesis on the regression function, but just minimal assumptions. Their weak points: they are more challenging to estimate and to interpret.

In this chapter we will focus mainly on the situation with only **one continuous predictor**  $X$  for predicting the response  $Y$ . The extension to more predictors is possible, but requires some extra work on notation that we do not address in full generality. In this case, the complete knowledge of  $Y$  when  $X = x$  is given by the conditional pdf  $f_{Y|X=x}(y) = \frac{f(x,y)}{f_X(x)}$ . While this pdf provides full knowledge about  $Y|X = x$ , it is also a challenging task to estimate it: for each  $x$  we have to estimate a *curve*! A simpler approach, yet still challenging, is to estimate the conditional mean (a scalar) for each  $x$  through the regression function

$$m(x) = \mathbb{E}[Y|X = x] = \int y f_{Y|X=x}(y) dy.$$

This density-based view of the regression function is useful in order to construct estimators, as we will see.

Finally, recall that  $Y$  can be expressed in terms of  $m$  by means of the *location-scale model*:

$$Y = m(X) + \sigma(X)\varepsilon,$$

where  $\sigma^2(x) := \text{Var}[Y|X = x]$  is the *conditional variance* and  $\varepsilon$  is independent from  $X$  and such that  $\mathbb{E}[\varepsilon] = 0$  and  $\text{Var}[\varepsilon] = 1$ .

### 6.1 Kernel density estimation

In order to introduce a nonparametric estimator for the  $m$ , we need to introduce first a nonparametric *density* estimator for the *density* of the predictor  $X$ . This estimator is aimed to estimate  $f$ , the density of  $X$ , from a sample  $X_1, \dots, X_n$  and without assuming any specific form for  $f$ . The **kernel density estimator**<sup>1</sup> does this job with the estimate

---

<sup>1</sup> Also known as the *Parzen-Rosenblatt estimator* to honor the proposals by Parzen (1962) and Rosenblatt (1956).

$$\hat{f}(x; h) := \frac{1}{nh} \sum_{i=1}^n K \left( \frac{x - X_i}{h} \right), \quad (6.1)$$

where  $K$  is known as a *kernel*, a density that is typically symmetric and unimodal at 0, and  $h > 0$  is the *bandwidth*. A common notation is  $K_h(z) := \frac{1}{h} K \left( \frac{z}{h} \right)$ , so  $\hat{f}(x; h) = \frac{1}{n} \sum_{i=1}^n K_h(x - X_i)$ . It is useful to recall (6.1) with the normal kernel. If that is the case, then  $K_h(x - X_i) = \phi_h(x - X_i)$  and the kernel is the density of a  $\mathcal{N}(X_i, h^2)$ . Thus the bandwidth  $h$  can be thought as the *standard deviation of the normal densities that have mean zero*. Figure 6.1 illustrates the construction of the kernel density estimator, and the bandwidth and kernel effects.

Construction of the kernel density estimator. The animation shows how the bandwidth and kernel affect the density estimate, and how the kernels are rescaled densities with modes at the data points. Application also available [here](#).

Several types of kernels are possible. The most popular is the *normal kernel*  $K(z) = \phi(z)$ , although the *Epanechnikov kernel*,  $K(z) = \frac{3}{4}(1 - z^2)1_{\{|z| < 1\}}$ , is the most efficient. The *rectangular kernel*  $K(z) = \frac{1}{2}1_{\{|z| < 1\}}$  yields the moving histogram as a particular case. The kernel density estimator inherits the smoothness properties of the kernel. That means, for example, (6.1) with a normal kernel is infinitely differentiable. But with an Epanechnikov kernel, (6.1) is not differentiable, and with a rectangular kernel is not even continuous. However, if a certain smoothness is guaranteed (continuity at least), the *choice of the kernel has little importance in practice* (at least compared with the choice of  $h$ ).

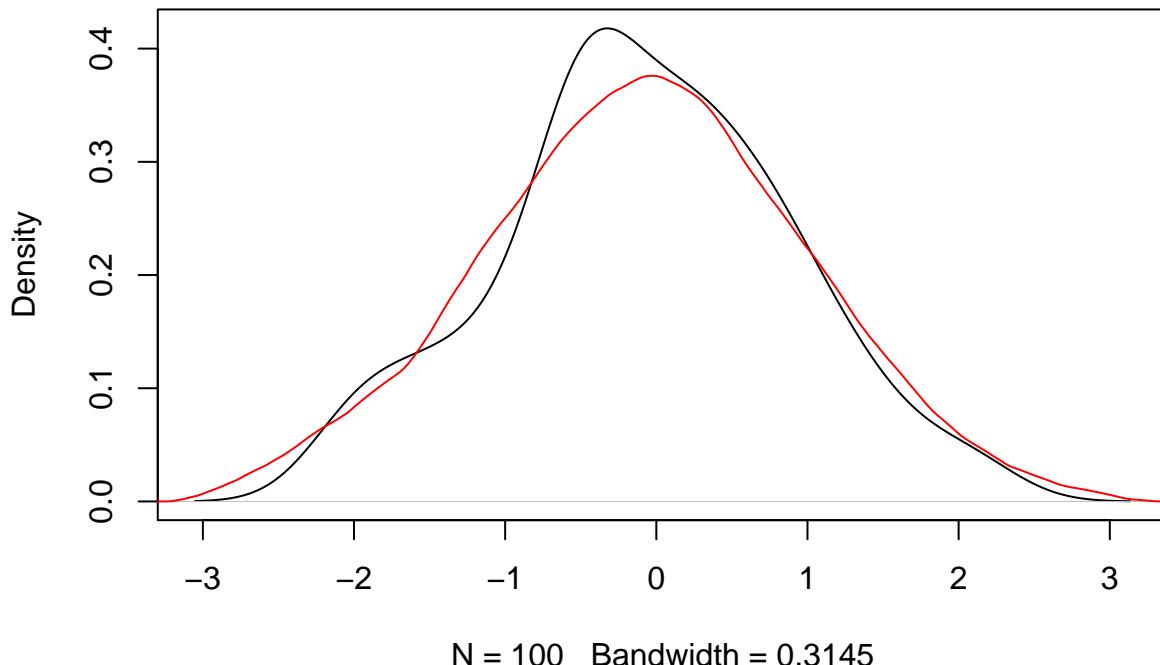
Implementation of the kernel density estimator in R is built-in through the `density` function. The function automatically chooses the bandwidth  $h$  using a *bandwidth selector*.

```
# Sample 100 points from a N(0, 1)
set.seed(1234567)
samp <- rnorm(n = 100, mean = 0, sd = 1)

# Quickly compute a kernel density estimator and plot the density object
# Automatically chooses bandwidth and uses normal kernel
plot(density(x = samp))

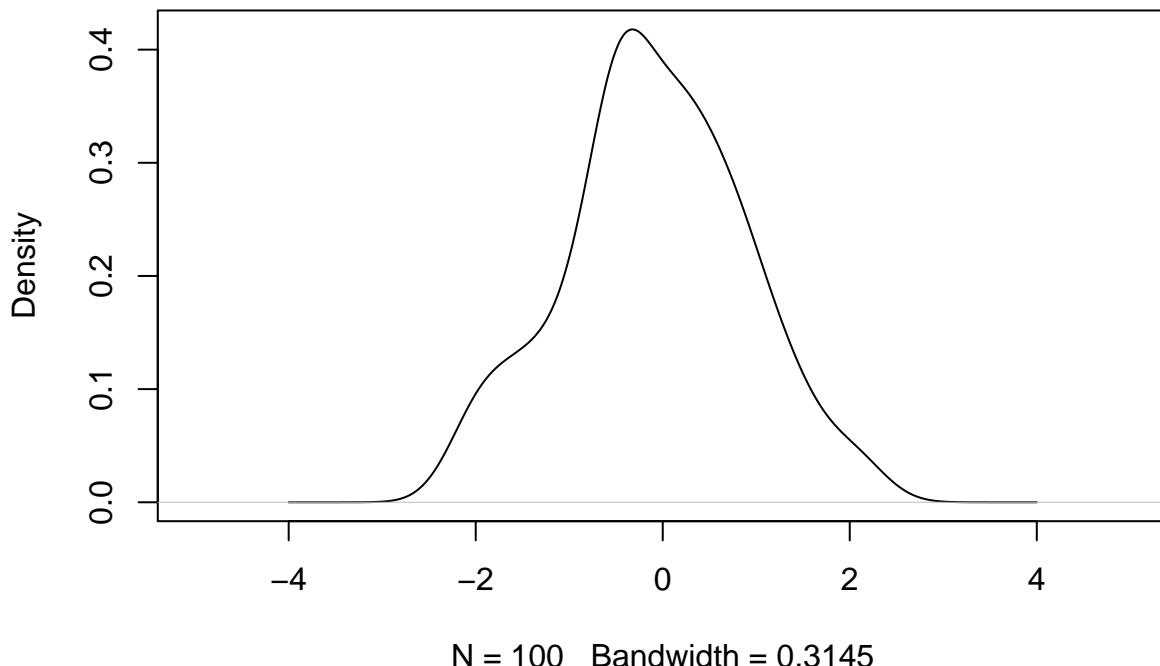
# Select a particular bandwidth (0.5) and kernel (Epanechnikov)
lines(density(x = samp, bw = 0.5, kernel = "epanechnikov"), col = 2)
```

**density.default(x = samp)**



```
# density automatically chooses the interval for plotting the kernel density
# estimator (observe that the black line goes to roughly between -3 and 3)
# This can be tuned using "from" and "to"
plot(density(x = samp, from = -4, to = 4), xlim = c(-5, 5))
```

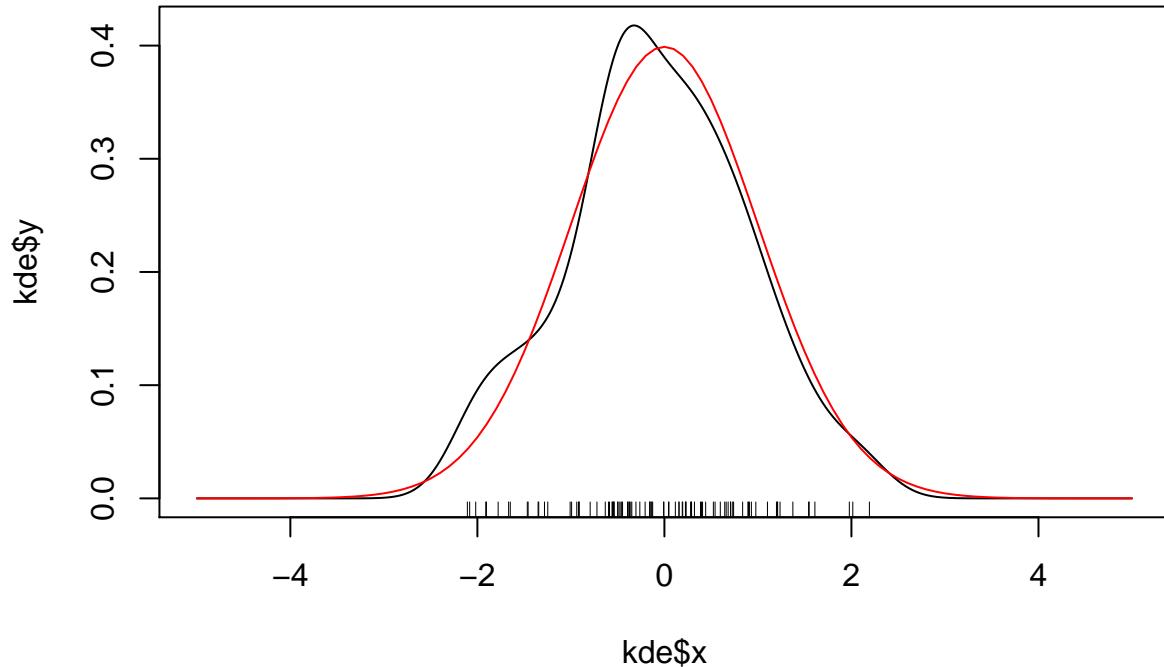
**density.default(x = samp, from = -4, to = 4)**



```

# The density object is a list
kde <- density(x = samp, from = -5, to = 5, n = 1024)
str(kde)
## List of 7
## $ x      : num [1:1024] -5 -4.99 -4.98 -4.97 -4.96 ...
## $ y      : num [1:1024] 5.98e-17 3.46e-17 2.56e-17 3.84e-17 4.50e-17 ...
## $ bw     : num 0.315
## $ n      : int 100
## $ call   : language density.default(x = samp, n = 1024, from = -5, to = 5)
## $ data.name: chr "samp"
## $ has.na  : logi FALSE
## - attr(*, "class")= chr "density"
# Note that the evaluation grid "x"" is not directly controlled, only through
# "from", "to", and "n" (better use powers of 2)
plot(kde$x, kde$y, type = "l")
curve(dnorm(x), col = 2, add = TRUE) # True density
rug(samp)

```



Load the dataset `faithful`. Then:

- Estimate and plot the density of `faithful$eruptions`.
- Create a new plot and superimpose different density estimations with bandwidths equal to 0.1, 0.5, and 1.
- Get the density estimate at *exactly* the point  $x = 3.1$  using  $h = 0.15$  and the Epanechnikov kernel.

The kernel density estimator can be extended to the multivariate setting by using *product kernels*. For a sample  $\mathbf{X}_1, \dots, \mathbf{X}_n$  in  $\mathbb{R}^p$ , the multivariate kernel density estimator employing product kernels is defined as

$$\hat{f}(\mathbf{x}; \mathbf{h}) = \frac{1}{n} \sum_{i=1}^n K_{h_1}(x_1 - X_{i,1}) \times \dots \times K_{h_p}(x_p - X_{i,p}), \quad (6.2)$$

where  $\mathbf{x} = (x_1, \dots, x_p)$ ,  $\mathbf{X}_i = (X_{i,1}, \dots, X_{i,p})$ , and  $\mathbf{h} = (h_1, \dots, h_p)$  is a vector of bandwidths (if the variables  $X_1, \dots, X_p$  have been standardized, then a simple choice is to consider  $h = h_1 = \dots = h_p$ ). The interpretation of (6.2) is analogous to the one of (6.1): build a mixture of densities with each density centered at the each data point.

## 6.2 Kernel regression estimation

### 6.2.1 Nadaraya-Watson estimator

Our objective is to estimate the regression function  $m$  nonparametrically. Due to its definition, we can rewrite it as follows:

$$\begin{aligned} m(x) &= \mathbb{E}[Y|X=x] \\ &= \int y f_{Y|X=x}(y) dy \\ &= \frac{\int y f(x, y) dy}{f_X(x)}. \end{aligned}$$

This expression shows an interesting point: the regression function can be computed from the joint density  $f$  and the marginal  $f_X$ . Therefore, given a sample  $(X_1, Y_1), \dots, (X_n, Y_n)$ , an estimate of  $m$  follows by replacing the previous densities by their kernel density estimators. To that aim, recall that in (6.2) we defined a multivariate extension of (6.1) based on product kernels. For the two dimensional case, the kernel density estimator with equal bandwidths  $\mathbf{h} = (h, h)$  is

$$\hat{f}(x, y; h) = \frac{1}{n} \sum_{i=1}^n K_h(x - X_i) K_h(y - Y_i). \quad (6.3)$$

Using (6.3),

$$\begin{aligned} m(x) &\approx \frac{\int y \hat{f}(x, y; h) dy}{\hat{f}_X(x; h)} \\ &= \frac{\int y \hat{f}(x, y; h) dy}{\hat{f}_X(x; h)} \\ &= \frac{\int y \frac{1}{n} \sum_{i=1}^n K_h(x - X_i) K_h(y - Y_i) dy}{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i)} \\ &= \frac{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i) \int y K_h(y - Y_i) dy}{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i)} \\ &= \frac{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i) Y_i}{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i)}. \end{aligned}$$

This yields the so-called **Nadaraya-Watson**<sup>2</sup> estimate:

---

<sup>2</sup>Termed due to the coetaneous proposals by Nadaraya (1964) and Watson (1964).

$$\hat{m}(x; 0, h) := \frac{1}{n} \sum_{i=1}^n \frac{K_h(x - X_i)}{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i)} Y_i = \sum_{i=1}^n W_i^0(x) Y_i, \quad (6.4)$$

where  $W_i^0(x) := \frac{K_h(x - X_i)}{\sum_{i=1}^n K_h(x - X_i)}$ . This estimate can be seen as a weighted average of  $Y_1, \dots, Y_n$  by means of the set of weights  $\{W_{n,i}(x)\}_{i=1}^n$  (check that they add to one). The set of weights depends on the evaluation point  $x$ . That means that the Nadaraya-Watson estimator is a **local mean of  $Y_1, \dots, Y_n$  around  $X = x$**  (see Figure 6.2.2).

Let's implement the Nadaraya-Watson estimate to get a feeling of how it works in practice.

```
# Nadaraya-Watson
mNW <- function(x, X, Y, h, K = dnorm) {

  # Arguments
  # x: evaluation points
  # X: vector (size n) with the predictors
  # Y: vector (size n) with the response variable
  # h: bandwidth
  # K: kernel

  # Matrix of size n x length(x)
  Kx <- sapply(X, function(Xi) K((x - Xi) / h) / h)

  # Weights
  W <- Kx / rowSums(Kx) # Column recycling!

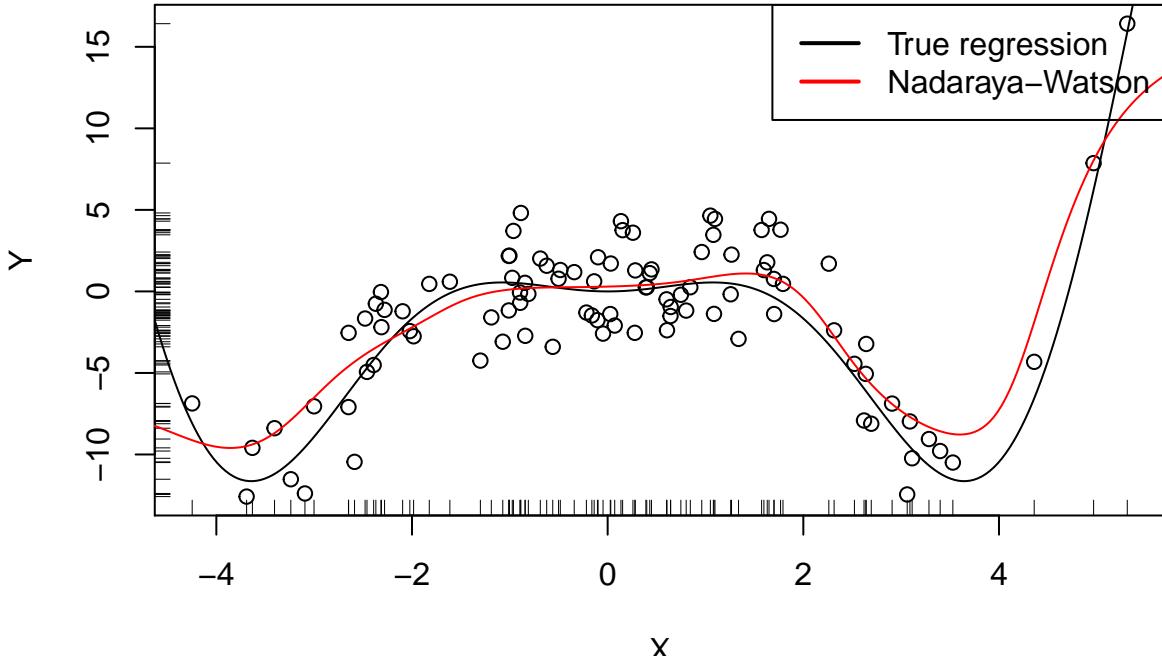
  # Means at x ("drop" to drop the matrix attributes)
  drop(W %*% Y)

}

# Generate some data to test the implementation
set.seed(12345)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) x^2 * cos(x)
X <- rnorm(n, sd = 2)
Y <- m(X) + eps
xGrid <- seq(-10, 10, l = 500)

# Bandwidth
h <- 0.5

# Plot data
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(xGrid, mNW(x = xGrid, X = X, Y = Y, h = h), col = 2)
legend("topright", legend = c("True regression", "Nadaraya-Watson"),
lwd = 2, col = 1:2)
```



Implement your own version of the Nadaraya-Watson estimator in R and compare it with `mNW`. You may focus only on the normal kernel and reduce the accuracy of the final computation up to `1e-7` to achieve better efficiency. Are you able to improve the speed of `mNW`? Use `system.time` or the `microbenchmark` package to measure the running times for a sample size of  $n = 10000$ .

The code below illustrates the effect of varying  $h$  for the Nadaraya-Watson estimator using `manipulate`.

```
# Simple plot of N-W for varying h's
library(manipulate)
manipulate({
  # Plot data
  plot(X, Y)
  rug(X, side = 1); rug(Y, side = 2)
  lines(xGrid, m(xGrid), col = 1)
  lines(xGrid, mNW(x = xGrid, X = X, Y = Y, h = h), col = 2)
  legend("topright", legend = c("True regression", "Nadaraya-Watson"),
         lwd = 2, col = 1:2)
}, h = slider(min = 0.01, max = 2, initial = 0.5, step = 0.01))
```

### 6.2.2 Local polynomial regression

Nadaraya-Watson can be seen as a particular case of a *local polynomial fit*, specifically, the one corresponding to a *local constant fit*. The motivation for the local polynomial fit comes from attempting to minimize the RSS

$$\sum_{i=1}^n (Y_i - m(X_i))^2. \quad (6.5)$$

This is not achievable directly, since no knowledge on  $m$  is available. However, by a  $p$ -th order Taylor expression it is possible to obtain that, for  $x$  close to  $X_i$ ,

$$\begin{aligned} m(X_i) &\approx m(x) + m'(x)(X_i - x) + \frac{m''(x)}{2}(X_i - x)^2 \\ &\quad + \cdots + \frac{m^{(p)}(x)}{p!}(X_i - x)^p. \end{aligned} \quad (6.6)$$

Replacing (6.6) in (6.5), we have that

$$\sum_{i=1}^n \left( Y_i - \sum_{j=0}^p \frac{m^{(j)}(x)}{j!} (X_i - x)^j \right)^2. \quad (6.7)$$

This expression is still not workable: it depends on  $m^{(j)}(x)$ ,  $j = 0, \dots, p$ , which of course are unknown! The *great idea* is to set  $\beta_j := \frac{m^{(j)}(x)}{j!}$  and turn (6.7) into a linear regression problem where the unknown parameters are  $\beta = (\beta_0, \beta_1, \dots, \beta_p)'$ :

$$\sum_{i=1}^n \left( Y_i - \sum_{j=0}^p \beta_j (X_i - x)^j \right)^2. \quad (6.8)$$

While doing so, an estimate of  $\beta$  automatically will yield estimates for  $m^{(j)}(x)$ ,  $j = 0, \dots, p$ , and we know how to obtain  $\hat{\beta}$  by minimizing (6.8). The final touch is to make the contributions of  $X_i$  dependent on the distance to  $x$  by weighting with kernels:

$$\hat{\beta} := \arg \min_{\beta \in \mathbb{R}^{p+1}} \sum_{i=1}^n \left( Y_i - \sum_{j=0}^p \beta_j (X_i - x)^j \right)^2 K_h(x - X_i). \quad (6.9)$$

Denoting

$$\mathbf{X} := \begin{pmatrix} 1 & X_1 - x & \cdots & (X_1 - x)^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_n - x & \cdots & (X_n - x)^p \end{pmatrix}_{n \times (p+1)}$$

and

$$\mathbf{W} := \text{diag}(K_h(X_1 - x), \dots, K_h(X_n - x)), \quad \mathbf{Y} := \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}_{n \times 1},$$

we can re-express (6.9) into a *weighted least squares problem* whose exact solution is

$$\hat{\beta} = \arg \min_{\beta \in \mathbb{R}^{p+1}} (\mathbf{Y} - \mathbf{X}\beta)' \mathbf{W} (\mathbf{Y} - \mathbf{X}\beta) \quad (6.10)$$

$$= (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W} \mathbf{Y}. \quad (6.11)$$

The estimate for  $m(x)$  is then computed as

$$\begin{aligned}\hat{m}(x; p, h) &:= \hat{\beta}_0 \\ &= \mathbf{e}'_1(\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{Y} \\ &= \sum_{i=1}^n W_i^p(x)Y_i,\end{aligned}$$

where  $W_i^p(x) := \mathbf{e}'_1(\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{e}_i$  and  $\mathbf{e}_i$  is the  $i$ -th canonical vector. Just as the Nadaraya-Watson, the local polynomial estimator is a *linear combination of the responses*. Two cases deserve special attention:

- $p = 0$  is the *local constant estimator* or the Nadaraya-Watson estimator. In this situation, the estimator has explicit weights, as we saw before:

$$W_i^0(x) = \frac{K_h(x - X_i)}{\sum_{j=1}^n K_h(x - X_j)}.$$

- $p = 1$  is the *local linear estimator*, which has weights equal to:

$$W_i^1(x) = \frac{\hat{s}_2(x; h) - \hat{s}_1(x; h)(X_i - x)}{\hat{s}_2(x; h)\hat{s}_0(x; h) - \hat{s}_1(x; h)^2} K_h(x - X_i), \quad (6.12)$$

where  $\hat{s}_r(x; h) := \frac{1}{n} \sum_{i=1}^n (X_i - x)^r K_h(x - X_i)$ .

Figure 6.2.2 illustrates the construction of the local polynomial estimator (up to cubic degree) and shows how  $\hat{\beta}_0 = \hat{m}(x; p, h)$ , the intercept of the local fit, estimates  $m$  at  $x$ .

Construction of the local polynomial estimator. The animation shows how local polynomial fits in a neighborhood of  $x$  are combined to provide an estimate of the regression function, which depends on the polynomial degree, bandwidth, and kernel (gray density at the bottom). The data points are shaded according to their weights for the local fit at  $x$ . Application also available here.

KernSmooth's `locpoly` implements the local polynomial estimator. Below are some examples of its usage.

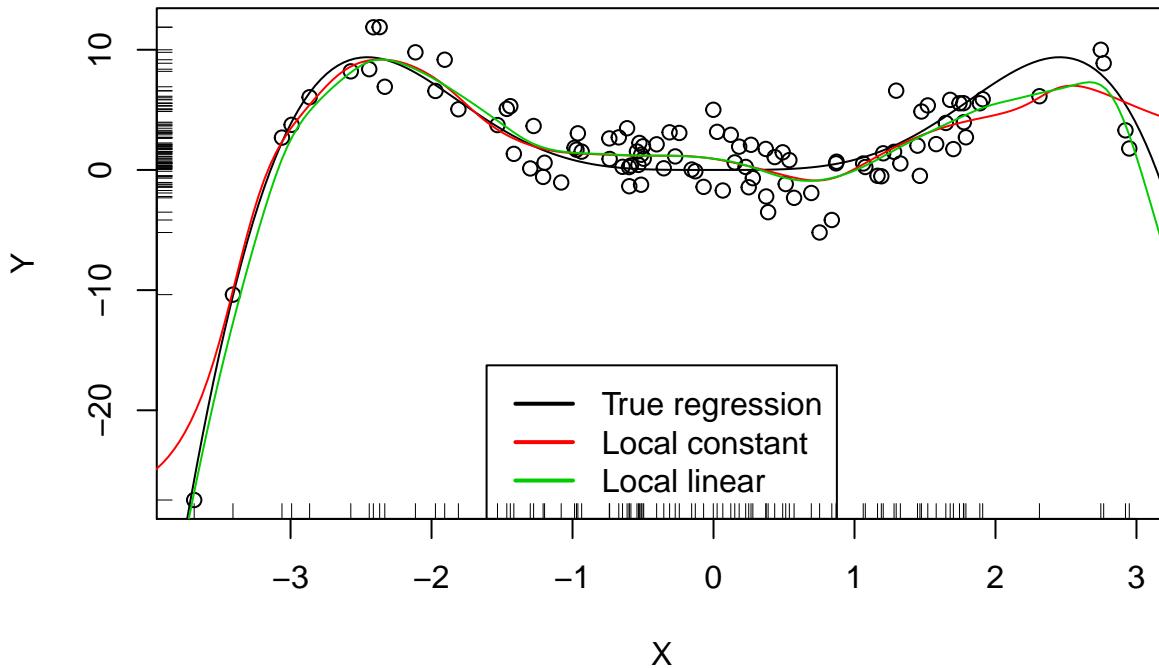
```
# Generate some data to test the implementation
set.seed(123456)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) x^3 * sin(x)
X <- rnorm(n, sd = 1.5)
Y <- m(X) + eps
xGrid <- seq(-10, 10, l = 500)

# Fits
h <- 0.25
lp0 <- locpoly(x = X, y = Y, bandwidth = h, degree = 0, range.x = c(-10, 10),
                 gridsize = 500)
lp1 <- locpoly(x = X, y = Y, bandwidth = h, degree = 1, range.x = c(-10, 10),
                 gridsize = 500)

# Prediction at x = 2
x <- 2
lp1$y[which.min(abs(lp1$x - x))] # Prediction
## [1] 5.445975
m(x) # Reality
```

```
## [1] 7.274379

# Plot data
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(lp0$x, lp0$y, col = 2)
lines(lp1$x, lp1$y, col = 3)
legend("bottom", legend = c("True regression", "Local constant",
                            "Local linear"),
       lwd = 2, col = 1:3)
```



```
# Simple plot of local polynomials for varying h's
library(manipulate)
manipulate({
  # Plot data
  lpp <- locpoly(x = X, y = Y, bandwidth = h, degree = p, range.x = c(-10, 10),
                  gridsize = 500)
  plot(X, Y)
  rug(X, side = 1); rug(Y, side = 2)
  lines(xGrid, m(xGrid), col = 1)
  lines(lpp$x, lpp$y, col = p + 2)
  legend("bottom", legend = c("True regression", "Local polynomial fit"),
         lwd = 2, col = c(1, p + 2))

}, h = slider(min = 0.01, max = 2, initial = 0.5, step = 0.01),
p = slider(min = 0, max = 4, initial = 0, step = 1))
```

A more sophisticated framework for performing nonparametric estimation of the regression function, for **multiple predictors** that are continuous or **discrete**, is the **np** package. The code below illustrates the usage of the function that implements nonparametric regression, **npreg**.

# TODO

## 6.3 Asymptotic properties

The purpose of this section is to provide some highlights on the asymptotic bias and variance of the local linear and local constant estimators. These provide useful insights on the effect of  $p$ ,  $m$ ,  $f$ , and  $\sigma^2$  in the performance of the estimators.

Along this section we will make the following assumptions<sup>3</sup>:

- **A1.**  $m$  is twice continuously differentiable.
- **A2.**  $\sigma^2$  is continuous and positive.
- **A3.**  $f$  is continuously differentiable and positive.
- **A4.** The kernel  $K$  is a symmetric and bounded pdf with finite second moment and is square integrable.
- **A5.**  $h = h_n$  is a deterministic sequence of bandwidths such that, when  $n \rightarrow \infty$ ,  $h \rightarrow 0$  and  $nh \rightarrow \infty$ .

The bias and variance are expanded in their *conditional* versions on the predictor's sample  $X_1, \dots, X_n$ . The reason for analyzing the conditional instead of the *unconditional* versions is avoiding technical difficulties that integration with respect to the predictor's density may pose. The main result is the following.

**Theorem 6.1.** *Under A1–A5, the conditional bias and variance of the local constant ( $p = 0$ ) and local linear ( $p = 1$ ) estimators are<sup>4</sup>*

$$\text{Bias}[\hat{m}(x; p, h)|X_1, \dots, X_n] = B_p(x)h^2 + o_{\mathbb{P}}(h^2), \quad (6.13)$$

$$\text{Var}[\hat{m}(x; p, h)|X_1, \dots, X_n] = \frac{R(K)}{nhf(x)}\sigma^2(x) + o_{\mathbb{P}}((nh)^{-1}), \quad (6.14)$$

where

$$B_p(x) = \frac{\mu_2(K)}{2} \left\{ m''(x) + 2 \frac{m'(x)f'(x)}{f(x)} 1_{\{p=0\}} \right\}.$$

The bias and variance expressions (6.13) and (6.14) yield very interesting insights:

- The bias decreases with  $h$  *quadratically* for  $p = 0, 1$ . The bias at  $x$  is directly proportional to  $m''(x)$  if  $p = 1$  and affected by  $m''(x)$  if  $p = 0$ . Therefore:
  - The bias is negative in concave regions, *i.e.*  $\{x \in \mathbb{R} : m(x)'' < 0\}$ . These regions correspond to *peaks and modes of  $m$*
  - Conversely, the bias is positive in convex regions, *i.e.*  $\{x \in \mathbb{R} : m(x)'' > 0\}$ . These regions correspond to *valleys of  $m$* .
  - **The wilder the curvature  $m''$ , the harder to estimate  $m$ .**
- The bias for  $p = 0$  at  $x$  is affected by  $m'(x)$ ,  $f'(x)$ , and  $f(x)$ . Precisely, **the lower the density  $f(x)$ , the larger the bias**. And **the faster  $m$  and  $f$  change at  $x$ , the larger the bias**. Thus the bias of the local constant estimator is much more sensible to  $m(x)$  and  $f(x)$  than the local linear (which is only sensible to  $m''(x)$ ). Particularly, the fact that it depends on  $f'(x)$  and  $f(x)$  is referred as the *design bias* since it depends merely on the predictor's distribution.
- The variance depends directly on  $\frac{\sigma^2(x)}{f(x)}$  for  $p = 0, 1$ . As a consequence, **the lower the density and larger the conditional variance, the more variable is  $\hat{m}(\cdot; p, h)$** . The variance decreases at a factor of  $(nh)^{-1}$  due to the *effective sample size*, which can be thought as the amount of data in the neighborhood of  $x$  for performing the estimation.

<sup>3</sup>These are the only assumptions done so far in the model (in addition to  $\varepsilon$  independent from  $X$ ).

<sup>4</sup>The notation  $o_{\mathbb{P}}(a_n)$  stands for a random variable that converges in probability to zero at a rate faster than  $a_n$ . It is mostly employed for denoting non-important terms in asymptotic expansions, like the ones in (6.13)–(6.14).

An extended version of Theorem 6.1, given in Theorem 3.1 of Fan and Gijbels (1996), shows that **odd order polynomial fits are preferable to even order polynomial fits**. The reason is that odd orders introduce an extra coefficient for the polynomial fit that allows them to reduce the bias, while at the same time they keep the variance unchanged. In summary, the conclusions of the above analysis of  $p = 0$  vs.  $p = 1$ , namely that  $p = 1$  has smaller bias than  $p = 0$  (but of the same order) and the same variance as  $p = 0$ , extend to the case  $p = 2\nu$  vs.  $p = 2\nu + 1$ ,  $\nu \in \mathbb{N}$ . This allows us to claim that local polynomial fitting *is an odd world* (Fan and Gijbels (1996)).

## 6.4 Bandwidth selection

Bandwidth selection is of key practical importance. Several bandwidth selectors have been proposed for kernel regression but, for simplicity, we will focus only on the **cross-validation** bandwidth selector.

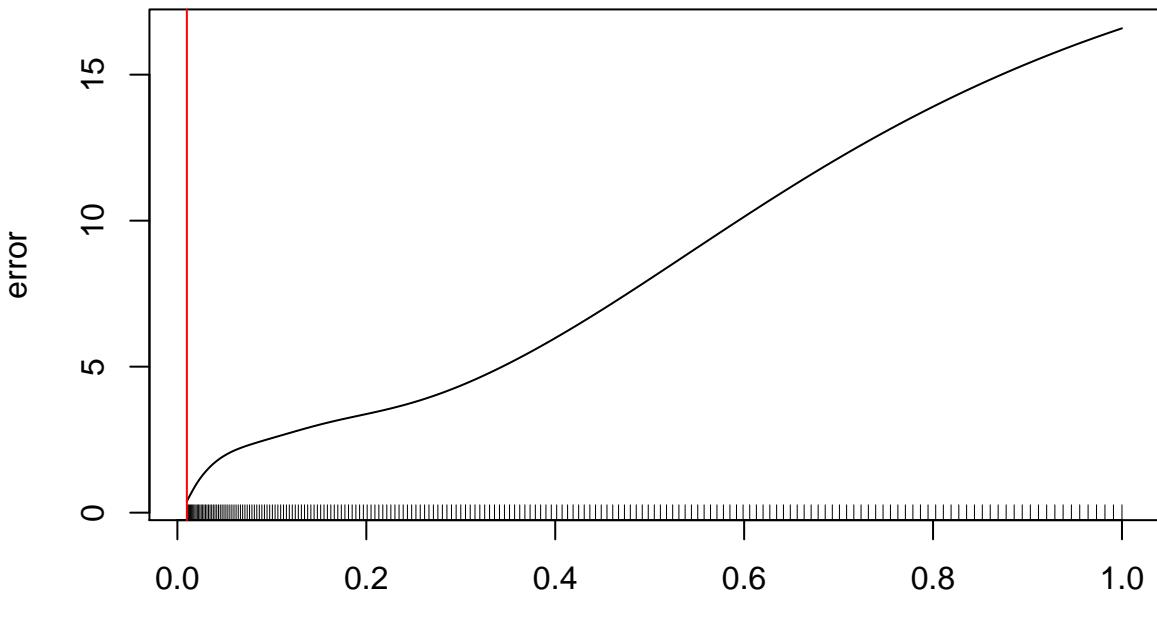
Following an analogy with the fit of the linear model, we could look for the bandwidth  $h$  such that it minimizes the RSS of the form

$$\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{m}(X_i; p, h))^2. \quad (6.15)$$

However, this is a bad idea. Attempting to minimize (6.15) always leads to  $h \approx 0$  that results in a useless interpolation of the data. Let's see an example.

```
# Grid for representing (3.22)
hGrid <- seq(0.1, 1, l = 200)^2
error <- sapply(hGrid, function(h)
  mean((Y - mNW(x = X, X = X, Y = Y, h = h))^2))

# Error curve
plot(hGrid, error, type = "l")
rug(hGrid)
abline(v = hGrid[which.min(error)], col = 2)
```



The

root of the problem is the comparison of  $Y_i$  with  $\hat{m}(X_i; p, h)$ , since there is nothing that forbids  $h \rightarrow 0$  and as a consequence  $\hat{m}(X_i; p, h) \rightarrow Y_i$ . We can change this behavior if we compare  $Y_i$  with  $\hat{m}_{-i}(X_i; p, h)$ , the **leave-one-out estimate** computed without the  $i$ -th datum  $(X_i, Y_i)$ :

$$\begin{aligned} \text{CV}(h) &:= \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{m}_{-i}(X_i; p, h))^2, \\ h_{\text{CV}} &:= \arg \min_{h>0} \text{CV}(h). \end{aligned}$$

The optimization of the above criterion might seem to be computationally expensive, since it is required to compute  $n$  regressions for a *single* evaluation of the objective function.

**Proposition 6.1.** The weights of the leave-one-out estimator  $\hat{m}_{-i}(x; p, h) = \sum_{\substack{j=1 \\ j \neq i}}^n W_{-i,j}^p(x) Y_j$  can be obtained from  $\hat{m}(x; p, h) = \sum_{i=1}^n W_i^p(x) Y_i$ :

$$W_{-i,j}^p(x) = \frac{W_j^p(x)}{\sum_{\substack{k=1 \\ k \neq i}}^n W_k^p(x)} = \frac{W_j^p(x)}{1 - W_i^p(x)}.$$

This implies that

$$\text{CV}(h) = \frac{1}{n} \sum_{i=1}^n \left( \frac{Y_i - \hat{m}(X_i; p, h)}{1 - W_i^p(X_i)} \right)^2.$$

Let's implement this simple bandwidth selector in R.

```
# Generate some data to test the implementation
set.seed(12345)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) x^2 + sin(x)
X <- rnorm(n, sd = 1.5)
Y <- m(X) + eps
xGrid <- seq(-10, 10, l = 500)

# Objective function
cvNW <- function(X, Y, h, K = dnorm) {
  sum(((Y - mNW(x = X, X = X, Y = Y, h = h, K = K)) /
       (1 - K(0) / colSums(K(outer(X, X, "-") / h))))^2)
}

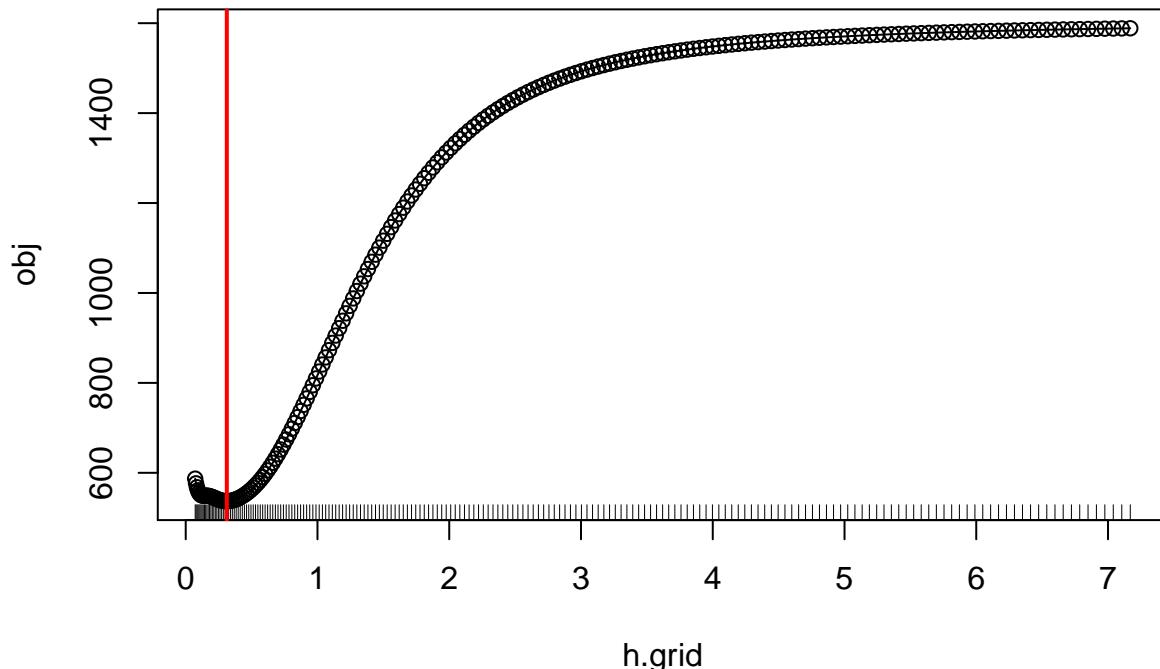
# Find optimum CV bandwidth, with sensible grid
bw.cv.grid <- function(X, Y,
                        h.grid = diff(range(X)) * (seq(0.1, 1, l = 200))^2,
                        K = dnorm, plot.cv = FALSE) {
  obj <- sapply(h.grid, function(h) cvNW(X = X, Y = Y, h = h, K = K))
  h <- h.grid[which.min(obj)]
  if (plot.cv) {
    plot(h.grid, obj, type = "o")
    rug(h.grid)
    abline(v = h, col = 2, lwd = 2)
  }
}
```

```

}
h
}

# Bandwidth
h <- bw.cv.grid(X = X, Y = Y, plot.cv = TRUE)

```

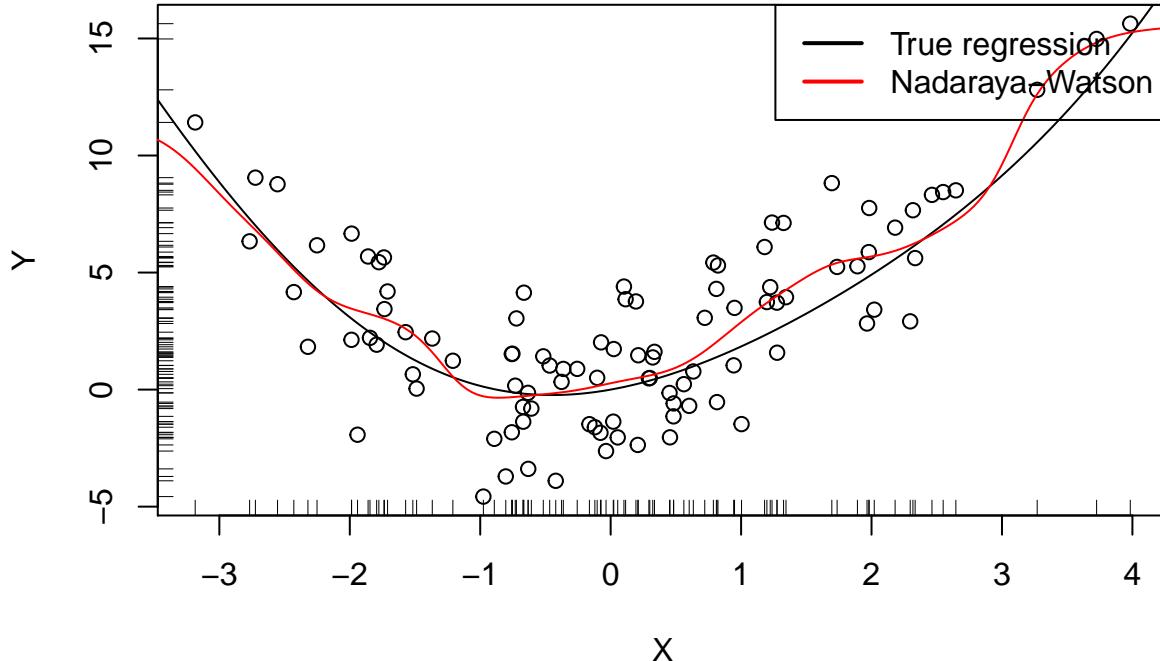


```

h
## [1] 0.3117806

# Plot result
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(xGrid, mNW(x = xGrid, X = X, Y = Y, h = h), col = 2)
legend("topright", legend = c("True regression", "Nadaraya-Watson"),
lwd = 2, col = 1:2)

```



A more sophisticated bandwidth selection can be achieved by `npregbw` in the `np` package. The code below illustrates the usage of this function.

```
# TODO
```

## 6.5 Local likelihood

We explore in this section an extension of the local polynomial estimator. This extension aims to estimate the regression function by relying in the likelihood, rather than the least squares. Thus, the idea behind the local likelihood is to **fit, locally, parametric models by maximum likelihood**.

We begin by seeing that local likelihood using the linear model is equivalent to local polynomial modelling. Theorem E.1 showed that, under the assumptions given in Section 2.3, the maximum likelihood estimate of  $\beta$  in the linear model

$$Y|(X_1, \dots, X_p) \sim \mathcal{N}(\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p, \sigma^2) \quad (6.16)$$

was equivalent to the least squares estimate,  $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{Y}$ . The reason was the form of the conditional (on  $X_1, \dots, X_p$ ) likelihood:

$$\begin{aligned} \ell(\beta) &= -\frac{n}{2} \log(2\pi\sigma^2) \\ &\quad - \frac{1}{2\sigma^2} \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \dots - \beta_p X_{ip})^2. \end{aligned}$$

If there is a single predictor  $X$ , polynomial fitting of order  $p$  of the conditional mean can be achieved by the well-known trick of identifying the  $j$ -th predictor  $X_j$  in (6.16) by  $X^j$ . This results in

$$Y|X \sim \mathcal{N}(\beta_0 + \beta_1 X + \dots + \beta_p X^p, \sigma^2). \quad (6.17)$$

Therefore, maximizing with respect to  $\beta$  the **weighted log-likelihood of the linear model around  $x$**  of (6.17),

$$\begin{aligned} \ell_{x,h}(\beta) &= -\frac{n}{2} \log(2\pi\sigma^2) \\ &\quad - \frac{1}{2\sigma^2} \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_i - \dots - \beta_p X_i^p)^2 K_h(x - X_i), \end{aligned}$$

provides  $\hat{\beta}_0 = \hat{m}(x; p, h)$ , the **local polynomial estimator**, as it was obtained in (6.9), but from a likelihood-based perspective. The same idea can be applied to the family of **generalized linear models**.

Construction of the local likelihood estimator. The animation shows how local likelihood fits in a neighborhood of  $x$  are combined to provide an estimate of the regression function for binary response, which depends on the polynomial degree, bandwidth, and kernel (gray density at the bottom). The data points are shaded according to their weights for the local fit at  $x$ . Application also available here.

We illustrate the local likelihood principle for the logistic regression. In this case,  $(X_1, Y_1), \dots, (X_n, Y_n)$  with

$$Y_i|X_i \sim \text{Ber}(\text{logistic}(\eta(X_i))), \quad i = 1, \dots, n.$$

with the polynomial term<sup>5</sup>

$$\eta(x) := \beta_0 + \beta_1 x + \dots + \beta_p x^p.$$

The log-likelihood of  $\beta$  is

$$\begin{aligned} \ell(\beta) &= \sum_{i=1}^n \{Y_i \log(\text{logistic}(\eta(X_i))) + (1 - Y_i) \log(1 - \text{logistic}(\eta(X_i)))\} \\ &= \sum_{i=1}^n \ell(Y_i, \eta(X_i)), \end{aligned}$$

where we consider the *log-likelihood addend*  $\ell(y, \eta) = y\eta - \log(1 + e^\eta)$ , and make explicit the dependence on  $\eta(x)$  for clarity in the next developments, and implicit the dependence on  $\beta$ .

The **local log-likelihood of  $\beta$  around  $x$**  is then

$$\ell_{x,h}(\beta) := \sum_{i=1}^n \ell(Y_i, \eta(X_i - x)) K_h(x - X_i). \quad (6.18)$$

Maximizing<sup>6</sup> the local log-likelihood (6.18) with respect to  $\beta$  provides

$$\hat{\beta} = \arg \max_{\beta \in \mathbb{R}^{p+1}} \ell_{x,h}(\beta).$$

The local likelihood estimate of  $\eta(x)$  is

$$\hat{\eta}(x) := \hat{\beta}_0.$$

<sup>5</sup>If  $p = 1$ , then we have the usual logistic model.

<sup>6</sup>No analytical solution for the optimization problem, numerical optimization is needed.

Note that the dependence of  $\hat{\beta}_0$  on  $x$  and  $h$  is omitted. From  $\hat{\eta}(x)$ , we can obtain the *local logistic regression* evaluated at  $x$  as

$$\hat{m}_\ell(x; h, p) := g^{-1}(\hat{\eta}(x)) = \text{logistic}(\hat{\beta}_0). \quad (6.19)$$

Each evaluation of  $\hat{m}_\ell(x; h, p)$  in a different  $x$  requires, thus, a fit of the underlying logistic model.

The code below shows three different ways of implementing the local logistic regression (of first degree) in R.

```
# Simulate some data
n <- 200
logistic <- function(x) 1 / (1 + exp(-x))
p <- function(x) logistic(1 - 3 * sin(x))
set.seed(123456)
X <- runif(n = n, -3, 3)
Y <- rbinom(n = n, size = 1, prob = p(X))

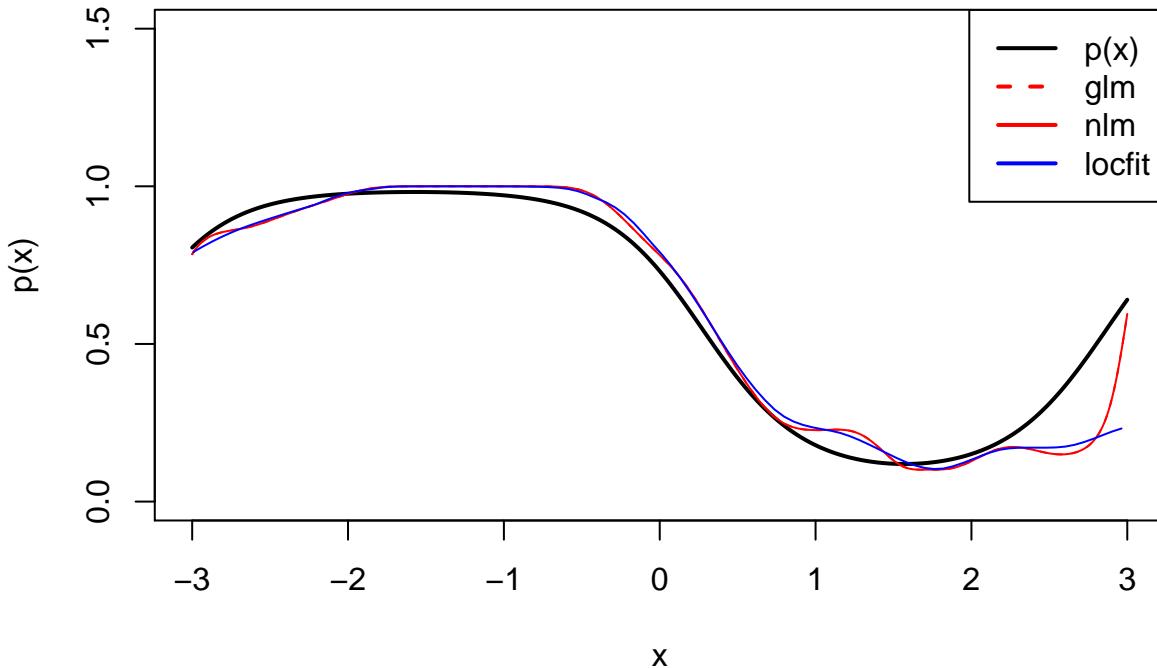
# Set bandwidth and evaluation grid
h <- 0.25
x <- seq(-3, 3, l = 501)

# Optimize the weighted log-likelihood through glm's built in procedure
suppressWarnings(
  fitGlm <- sapply(x, function(x) {
    K <- dnorm(x = x, mean = X, sd = h)
    glm.fit(x = cbind(1, X - x), y = Y, weights = K,
             family = binomial())$coefficients[1]
  })
)

# Optimize the weighted log-likelihood explicitly
suppressWarnings(
  fitNlm <- sapply(x, function(x) {
    K <- dnorm(x = x, mean = X, sd = h)
    nlm(f = function(beta) {
      -sum(K * (Y * (beta[1] + beta[2] * (X - x)) -
                 log(1 + exp(beta[1] + beta[2] * (X - x)))))
    }, p = c(0, 0))$estimate[1]
  })
)

# Using locfit
# Bandwidth can not be controlled explicitly - only though nn in ?lp
library(locfit)
fitLocfit <- locfit(Y ~ lp(X, deg = 1, nn = 0.25), family = "binomial",
                      kern = "gauss")

# Compare fits
plot(x, p(x), ylim = c(0, 1.5), type = "l", lwd = 2)
lines(x, logistic(fitGlm), col = 2)
lines(x, logistic(fitNlm), col = 2, lty = 2)
plot(fitLocfit, add = TRUE, col = 4)
legend("topright", legend = c("p(x)", "glm", "nlm", "locfit"), lwd = 2,
       col = c(1, 2, 2, 4), lty = c(1, 2, 1, 1))
```



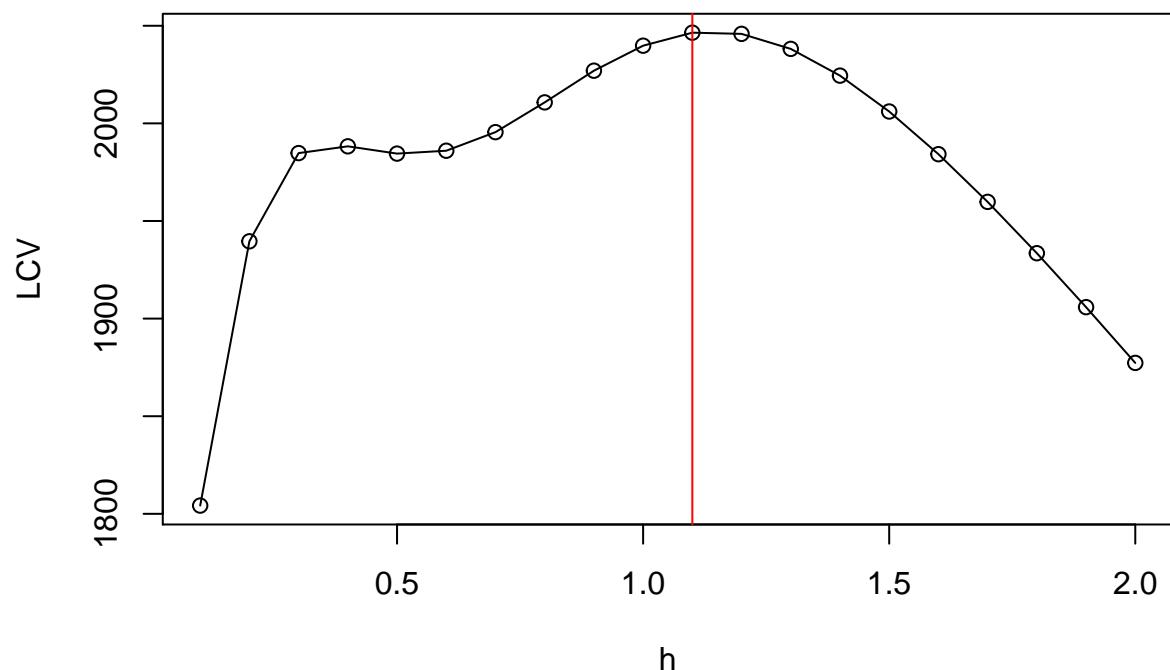
Bandwidth selection can be done by means of *likelihood cross-validation*. The objective is to maximize the local likelihood fit at  $(X_i, Y_i)$  but removing the influence by the datum itself. That is, maximizing

$$\text{LCV}(h) = \sum_{i=1}^n \ell(Y_i, \hat{\eta}_{-i}(X_i)), \quad (6.20)$$

where  $\hat{\eta}_{-i}(X_i)$  represents the local fit at  $X_i$  without the  $i$ -th datum  $(X_i, Y_i)$ . Unfortunately, the nonlinearity of (6.19) forbids a simplifying result as Proposition 6.1. Thus, in principle, it is required to fit  $n$  local likelihoods for sample size  $n - 1$  for obtaining a single evaluation of (6.20). The interested reader is referred to Sections 4.3.3 and 4.4.3 of Loader (1999) for an approximation of (6.20) that only requires a local likelihood fit for a single sample.

We conclude by illustrating how to compute the LCV function and optimize it (keep in mind that much more efficient implementations are possible!).

```
# Exact LCV - recall that we *maximize* the LCV!
h <- seq(0.1, 2, by = 0.1)
suppressWarnings(
  LCV <- sapply(h, function(h) {
    sum(sapply(1:n, function(i) {
      K <- dnorm(x = X[i], mean = X[-i], sd = h)
      nlm(f = function(beta) {
        -sum(K * (Y[-i] * (beta[1] + beta[2] * (X[-i] - X[i])) -
              log(1 + exp(beta[1] + beta[2] * (X[-i] - X[i])))))
      }, p = c(0, 0)$minimum
    }))
  })
)
plot(h, LCV, type = "o")
abline(v = h[which.max(LCV)], col = 2)
```





## Appendix A

# Installation of R and RStudio

This is what you have to do in order to install R and RStudio in your own computer:

1. In Mac OS X, download and install first **XQuartz** and log out and back on your Mac OS X account (this is an **important** step that is required for 3D graphics to work). Be sure that your Mac OS X system is up-to-date.
2. Download the latest version of R at <https://cran.r-project.org/>. For Windows, you can download it directly here. For Mac OS X you can download the latest version (at the time of writing this, 3.4.1) here.
3. Install R. In Windows, be sure to select the 'Startup options' and then choose 'SDI' in the 'Display Mode' options. Leave the rest of installation options as default.
4. Download the latest version of RStudio for your system at <https://www.rstudio.com/products/rstudio/download/#download> and install it.

If there is any Linux user, kindly follow the corresponding instructions here for installing R, download RStudio (only certain Ubuntu and Fedora versions are supported), and install it using your package manager.



## Appendix B

# 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 B.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 B.1);
2. for running a line or code selection from the script in the *console* (first tab in the lower-right panel in Figure B.1), you do it with the keyboard shortcut 'Ctrl+Enter' (Windows and Linux) or 'Cmd+Enter' (Mac OS X).

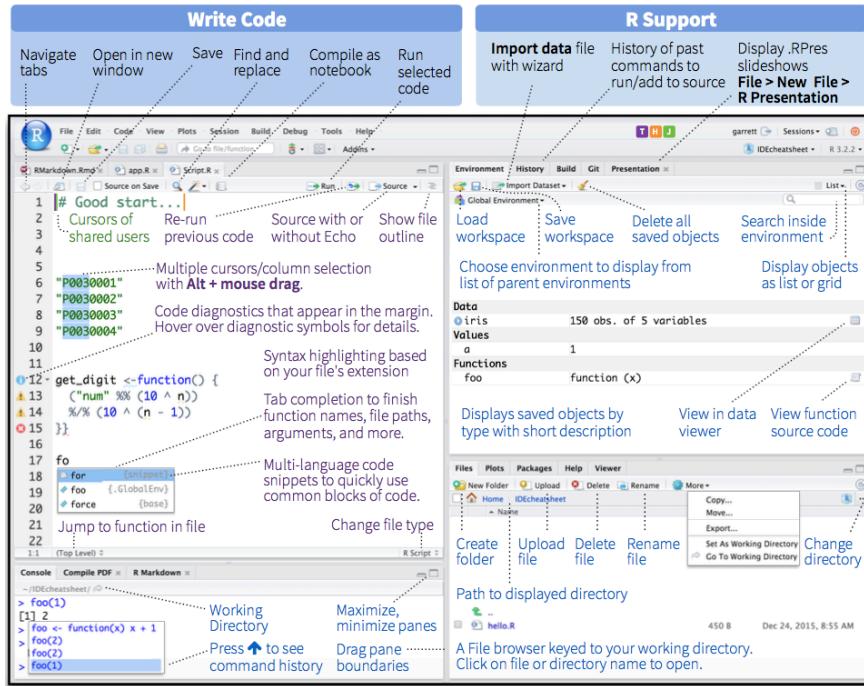


Figure B.1: Main window of 'RStudio'. The red shows the code panel and the yellow shows the console output. Extracted from [here](<https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf>).



# Appendix C

## Introduction to R

This section provides a collection of self-explainable snippets of the programming language R (R Core Team, 2017) that show the very basics of the language. It is not meant to be an exhaustive introduction to R, but rather a reminder/panoramic of a collection of basic functions and methods.

In the following, `#` denotes comments to the code and `##` outputs of the code.

### Simple computations

```
# The console can act as a simple calculator
1.0 + 1.1
## [1] 2.1
2 * 2
## [1] 4
3/2
## [1] 1.5
2^3
## [1] 8
1/0
## [1] Inf
0/0
## [1] NaN

# Use ";" for performing several operations in the same line
(1 + 3) * 2 - 1; 3 + 2
## [1] 7
## [1] 5

# Elemental mathematical functions
sqrt(2); 2^0.5
## [1] 1.414214
## [1] 1.414214
exp(1)
## [1] 2.718282
log(10) # Neperian logarithm
## [1] 2.302585
log10(10); log2(10) # Logs in base 10 and 2
## [1] 1
```

```
## [1] 3.321928
sin(pi); cos(0); asin(0)
## [1] 1.224647e-16
## [1] 1
## [1] 0
tan(pi/3)
## [1] 1.732051
sqrt(-1)
## Warning in sqrt(-1): NaNs produced
## [1] NaN
```

```
# Remember to close the parenthesis
1 +
(1 + 3
## Error: <text>:4:0: unexpected end of input
## 2: 1 +
## 3: (1 + 3
##     ^
```



Compute:

- $\frac{e^2 + \sin(2)}{\cos^{-1}\left(\frac{1}{2}\right) + 2}$ . Answer: 2.723274.
- $\sqrt{3^{2.5} + \log(10)}$ . Answer: 4.22978.
- $(2^{0.93} - \log_2(3 + \sqrt{2 + \sin(1)}))10^{\tan(1/3)}\sqrt{3^{2.5} + \log(10)}$ . Answer: -3.032108.

## Variables and assignment

```
# Any operation that you perform in R can be stored in a variable (or object)
# with the assignment operator "<-" 
x <- 1

# To see the value of a variable, simply type it
x
## [1] 1

# A variable can be overwritten
x <- 1 + 1

# Now the value of x is 2 and not 1, as before
x
## [1] 2

# Careful with capitalization
X
## Error in eval(expr, envir, enclos): object 'X' not found

# Different
X <- 3
x; X
## [1] 2
```

```
## [1] 3

# The variables are stored in your workspace (a .RData file)
# A handy tip to see what variables are in the workspace
ls()
## [1] "x" "X"
# Now you know which variables can be accessed!

# Remove variables
rm(x)
x
## Error in eval(expr, envir, enclos): object 'X' not found
```



Do the following:

- Store  $-123$  in the variable  $y$ .
- Store the log of the square of  $y$  in  $z$ .
- Store  $\frac{y-z}{y+z^2}$  in  $y$  and remove  $z$ .
- Output the value of  $y$ . *Answer: 4.366734.*

## Vectors

```
# These are vectors - arrays of numbers
# We combine numbers with the function "c"
c(1, 3)
## [1] 1 3
c(1.5, 0, 5, -3.4)
## [1] 1.5 0.0 5.0 -3.4

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

# Storing some vectors
myData <- c(1, 2)
myData2 <- c(-4.12, 0, 1.1, 1, 3, 4)
myData
## [1] 1 2
myData2
## [1] -4.12 0.00 1.10 1.00 3.00 4.00

# Entrywise operations
myData + 1
## [1] 2 3
myData^2
## [1] 1 4

# If you want to access a position of a vector, use [position]
myData[1]
## [1] 1
```

```

myData2[6]
## [1] 4

# You also can change elements
myData[1] <- 0
myData
## [1] 0 2

# Think on what you want to access...
myData2[7]
## [1] NA
myData2[0]
## numeric(0)

# If you want to access all the elements except a position, use [-position]
myData2[-1]
## [1] 0.0 1.1 1.0 3.0 4.0
myData2[-2]
## [1] -4.12 1.10 1.00 3.00 4.00

# Also with vectors as indexes
myData2[1:2]
## [1] -4.12 0.00
myData2[myData]
## [1] 0

# And also
myData2[-c(1, 2)]
## [1] 1.1 1.0 3.0 4.0

# But do not mix positive and negative indexes!
myData2[c(-1, 2)]
## Error in myData2[c(-1, 2)]: only 0's may be mixed with negative subscripts

# Remove the first element
myData2 <- myData2[-1]

```



Do the following:

- Create the vector  $x = (1, 7, 3, 4)$ .
- Create the vector  $y = (100, 99, 98, \dots, 2, 1)$ .
- Create the vector  $z = c(4, 8, 16, 32, 96)$ .
- Compute  $x_2 + y_4$  and  $\cos(x_3) + \sin(x_2)e^{-y_2}$ . *Answers:* 104 and  $-0.9899925$ .
- Set  $x_2 = 0$  and  $y_2 = -1$ . Recompute the previous expressions. *Answers:* 97 and 2.785875.
- Index  $y$  by  $x + 1$  and store it as  $z$ . What is the output? *Answer:*  $z$  is  $c(-1, 100, 97, 96)$ .

## Some functions

```

# Functions take arguments between parenthesis and transform them into an output
sum(myData)
## [1] 2

```

```

prod(myData)
## [1] 0

# Summary of an object
summary(myData)
##    Min. 1st Qu. Median    Mean 3rd Qu.    Max.
##    0.0    0.5    1.0    1.0    1.5    2.0

# Length of the vector
length(myData)
## [1] 2

# Mean, standard deviation, variance, covariance, correlation
mean(myData)
## [1] 1
var(myData)
## [1] 2
cov(myData, myData^2)
## [1] 4
cor(myData, myData * 2)
## [1] 1
quantile(myData)
##    0% 25% 50% 75% 100%
##  0.0  0.5  1.0  1.5  2.0

# Maximum and minimum of vectors
min(myData)
## [1] 0
which.min(myData)
## [1] 1

# Usually the functions have several arguments, which are set by "argument = value"
# In this case, the second argument is a logical flag to indicate the kind of sorting
sort(myData) # If nothing is specified, decreasing = FALSE is assumed
## [1] 0 2
sort(myData, decreasing = TRUE)
## [1] 2 0

# Do not know what are the arguments of a function? Use args and help!
args(mean)
## function (x, ...)
## NULL
?mean

```



Do the following:

- Compute the mean, median and variance of  $y$ . *Answers: 49.5, 49.5, 843.6869.*
- Do the same for  $y + 1$ . What are the differences?
- What is the maximum of  $y$ ? Where is it placed?
- Sort  $y$  increasingly and obtain the 5th and 76th positions. *Answer: c(4,75).*
- Compute the covariance between  $y$  and  $y$ . Compute the variance of  $y$ . Why do you get the same result?

## Matrices, data frames, and lists

```

# A matrix is an array of vectors
A <- matrix(1:4, nrow = 2, ncol = 2)
A
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

# Another matrix
B <- matrix(1, nrow = 2, ncol = 2, byrow = TRUE)
B
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1

# Matrix is a vector with dimension attributes
dim(A)
## [1] 2 2

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

# Entrywise operations
A + 1
##      [,1] [,2]
## [1,]    2    4
## [2,]    3    5
A * B
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

# Accessing elements
A[2, 1] # Element (2, 1)
## [1] 2
A[1, ] # First row - this is a vector
## [1] 1 3
A[, 2] # First column - this is a vector
## [1] 3 4

# Obtain rows and columns as matrices (and not as vectors)
A[1, , drop = FALSE]
##      [,1] [,2]
## [1,]    1    3

```

```

A[, 2, drop = FALSE]
##      [,1]
## [1,]    3
## [2,]    4

# Matrix transpose
t(A)
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4

# Matrix multiplication
A %*% B
##      [,1] [,2]
## [1,]    4    4
## [2,]    6    6
A %*% B[, 1]
##      [,1]
## [1,]    4
## [2,]    6
A %*% B[1, ]
##      [,1]
## [1,]    4
## [2,]    6

# Care is needed
A %*% B[1, , drop = FALSE] # Incompatible product
## Error in A %*% B[1, , drop = FALSE]: non-conformable arguments

# Compute inverses with "solve"
solve(A) %*% A
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1

# 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
##   var1 var2
## 1     1     3
## 2     2     4

# You can change names
names(myDf) <- c("newname1", "newname2")
myDf
##   newname1 newname2
## 1         1         3
## 2         2         4

# The nice thing is that you can access variables by its name with
# the "$" operator
myDf$newname1

```

```

## [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
##   newname1 newname2 myNewVariable
## 1         1         3         0
## 2         2         4         1

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

# Access elements by names
myList$myData
## [1] 0 2
myList$A
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
myList$myDf
##   newname1 newname2 myNewVariable
## 1         1         3         0
## 2         2         4         1

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

# A less lengthy output
names(myList)
## [1] "myData" "A"      "myDf"

```



Do the following:

- Create a matrix called `M` with rows given by `y[3:5]`, `y[3:5]^2` and `log(y[3:5])`.
- Create a data frame called `myDataFrame` with column names “`y`”, “`y2`” and “`logy`” containing the vectors `y[3:5]`, `y[3:5]^2` and `log(y[3:5])`, respectively.
- Create a list, called `l`, with entries for `x` and `M`. Access the elements by their names.
- Compute the squares of `myDataFrame` and save the result as `myDataFrame2`.

- Compute the log of the sum of `myDataFrame` and `myDataFrame2`. *Answer:*

```
##          y      y2    logy
## 1 9.180087 18.33997 3.242862
## 2 9.159678 18.29895 3.238784
## 3 9.139059 18.25750 3.234656
```

## More on data frames

```
# The iris dataset is already imported in R
# (beware: locfit has also an iris dataset, with different names and shorter)

# The beginning of the data
head(iris)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

# "names" gives you the variables in the data frame
names(iris)
## [1] "Sepal.Length" "Sepal.Width"   "Petal.Length" "Petal.Width"
## [5] "Species"

# So we can access variables by "$" or as in a matrix
iris$Sepal.Length[1:10]
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
iris[1:10, 1]
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
iris[3, 1]
## [1] 4.7

# Information on the dimension of the data frame
dim(iris)
## [1] 150   5

# "str" gives the structure of any object in R
str(iris)
## 'data.frame': 150 obs. of  5 variables:
##   $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   $ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

# Recall the species variable: it is a categorical variable (or factor),  
# not a numeric variable

```
iris$Species[1:10]
## [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa
```

```
## Levels: setosa versicolor virginica

# Factors can only take certain values
levels(iris$Species)
## [1] "setosa"    "versicolor" "virginica"

# If a file contains a variable with character strings as observations (either
# encapsulated by quotation marks or not), the variable will become a factor
# when imported into R
```



Do the following:

- Load the `faithful` dataset into R.
- Get the dimensions of `faithful` and show beginning of the data.
- Retrieve the fifth observation of `eruptions` in two different ways.
- Obtain a summary of `waiting`.

## Vector-related functions

```
# The function "seq" creates sequences of numbers equally separated
seq(0, 1, by = 0.1)
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
seq(0, 1, length.out = 5)
## [1] 0.00 0.25 0.50 0.75 1.00

# You can short the latter argument
seq(0, 1, l = 5)
## [1] 0.00 0.25 0.50 0.75 1.00

# Repeat number
rep(0, 5)
## [1] 0 0 0 0 0

# Reverse a vector
myVec <- c(1:5, -1:3)
rev(myVec)
## [1] 3 2 1 0 -1 5 4 3 2 1

# Another way
myVec[length(myVec):1]
## [1] 3 2 1 0 -1 5 4 3 2 1

# Count repetitions in your data
table(iris$Sepal.Length)
##
## 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6
## 1 3 1 4 2 5 6 10 9 4 1 6 7 6 8 7 3 6
## 6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7 7.1 7.2 7.3 7.4 7.6 7.7 7.9
## 6 4 9 7 5 2 8 3 4 1 1 3 1 1 1 4 1
table(iris$Species)
##
```

```
##      setosa  versicolor  virginica
##      50          50          50
```



Do the following:

- Create the vector  $x = (0.3, 0.6, 0.9, 1.2)$ .
- Create a vector of length 100 ranging from 0 to 1 with entries equally separated.
- Compute the amount of zeros and ones in  $x \leftarrow c(0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0)$ . Check that they are the same as in `rev(x)`.
- Compute the vector  $(0.1, 1.1, 2.1, \dots, 100.1)$  in four different ways using `seq` and `rev`. Do the same but using `:` instead of `seq`. (*Hint*: add 0.1)

## Logical conditions and subsetting

```
# Relational operators: x < y, x > y, x <= y, x >= y, x == y, x != y
# They return TRUE or FALSE

# Smaller than
0 < 1
## [1] TRUE

# Greater than
1 > 1
## [1] FALSE

# Greater or equal to
1 >= 1 # Remember: ">=" and not ">=" !
## [1] TRUE

# Smaller or equal to
2 <= 1 # Remember: "<=" and not "<=" !
## [1] FALSE

# Equal
1 == 1 # Tests equality. Remember: "==" and not "=" !
## [1] TRUE

# Unequal
1 != 0 # Tests inequality
## [1] TRUE

# TRUE is encoded as 1 and FALSE as 0
TRUE + 1
## [1] 2
FALSE + 1
## [1] 1

# In a vector-like fashion
x <- 1:5
y <- c(0, 3, 1, 5, 2)
x < y
```

```

## [1] FALSE TRUE FALSE TRUE FALSE
x == y
## [1] FALSE FALSE FALSE FALSE FALSE
x != y
## [1] TRUE TRUE TRUE TRUE TRUE

# Subsetting of vectors
x
## [1] 1 2 3 4 5
x[x >= 2]
## [1] 2 3 4 5
x[x < 3]
## [1] 1 2

# Easy way of work with parts of the data
data <- data.frame(x = c(0, 1, 3, 3, 0), y = 1:5)
data
##   x y
## 1 0 1
## 2 1 2
## 3 3 3
## 4 3 4
## 5 0 5

# Data such that x is zero
data0 <- data[data$x == 0, ]
data0
##   x y
## 1 0 1
## 5 0 5

# Data such that x is larger than 2
data2 <- data[data$x > 2, ]
data2
##   x y
## 3 3 3
## 4 3 4

# In an example
iris$Sepal.Width[iris$Sepal.Width > 3]
##  [1] 3.5 3.2 3.1 3.6 3.9 3.4 3.4 3.1 3.7 3.4 4.0 4.4 3.9 3.5 3.8 3.8 3.4
## [18] 3.7 3.6 3.3 3.4 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2 3.5 3.6 3.4
## [35] 3.5 3.2 3.5 3.8 3.8 3.2 3.7 3.3 3.2 3.2 3.1 3.3 3.1 3.2 3.4 3.1 3.3
## [52] 3.6 3.2 3.2 3.8 3.2 3.3 3.2 3.8 3.4 3.1 3.1 3.1 3.1 3.2 3.3 3.4

# Problem - what happened?
data[x > 2, ]
##   x y
## 3 3 3
## 4 3 4
## 5 0 5

# In an example

```

```

summary(iris)
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##   Median :5.800   Median :3.000   Median :4.350   Median :1.300
##   Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##   Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##           Species
##   setosa      :50
##   versicolor:50
##   virginica  :50
##
##
##
summary(iris[iris$Sepal.Width > 3, ])
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.   :4.400   Min.   :3.100   Min.   :1.000   Min.   :0.1000
##   1st Qu.:5.000   1st Qu.:3.200   1st Qu.:1.450   1st Qu.:0.2000
##   Median :5.400   Median :3.400   Median :1.600   Median :0.4000
##   Mean   :5.684   Mean   :3.434   Mean   :2.934   Mean   :0.9075
##   3rd Qu.:6.400   3rd Qu.:3.600   3rd Qu.:5.000   3rd Qu.:1.8000
##   Max.   :7.900   Max.   :4.400   Max.   :6.700   Max.   :2.5000
##           Species
##   setosa      :42
##   versicolor: 8
##   virginica  :17
##
##
##
# On the factor variable only makes sense == and !=
summary(iris[iris$Species == "setosa", ])
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.   :4.300   Min.   :2.300   Min.   :1.000   Min.   :0.100
##   1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400   1st Qu.:0.200
##   Median :5.000   Median :3.400   Median :1.500   Median :0.200
##   Mean   :5.006   Mean   :3.428   Mean   :1.462   Mean   :0.246
##   3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575   3rd Qu.:0.300
##   Max.   :5.800   Max.   :4.400   Max.   :1.900   Max.   :0.600
##           Species
##   setosa      :50
##   versicolor: 0
##   virginica  : 0
##
##
##
# Subset argument in lm
lm(Sepal.Width ~ Petal.Length, data = iris, subset = Sepal.Width > 3)
##
## Call:
## lm(formula = Sepal.Width ~ Petal.Length, data = iris, subset = Sepal.Width >

```

```

##      3)
##
## Coefficients:
## (Intercept)  Petal.Length
##      3.59439     -0.05455
lm(Sepal.Width ~ Petal.Length, data = iris, subset = iris$Sepal.Width > 3)
##
## Call:
## lm(formula = Sepal.Width ~ Petal.Length, data = iris, subset = iris$Sepal.Width >
##      3)
##
## Coefficients:
## (Intercept)  Petal.Length
##      3.59439     -0.05455
# Both iris$Sepal.Width and Sepal.Width in subset are fine: data = iris
# tells R to look for Sepal.Width in the iris dataset

# AND operator "&"
TRUE & TRUE
## [1] TRUE
TRUE & FALSE
## [1] FALSE
FALSE & FALSE
## [1] FALSE

# OR operator "|"
TRUE | TRUE
## [1] TRUE
TRUE | FALSE
## [1] TRUE
FALSE | FALSE
## [1] FALSE

# Both operators are useful for checking for ranges of data
y
## [1] 0 3 1 5 2
index1 <- (y <= 3) & (y > 0)
y[index1]
## [1] 3 1 2
index2 <- (y < 2) | (y > 4)
y[index2]
## [1] 0 1 5

# In an example
summary(iris[iris$Sepal.Width > 3 & iris$Sepal.Width < 3.5, ])
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.   :4.400   Min.   :3.100   Min.   :1.200   Min.   :0.100
##   1st Qu.:4.925   1st Qu.:3.125   1st Qu.:1.500   1st Qu.:0.200
##   Median  :5.950   Median  :3.200   Median  :4.450   Median  :1.400
##   Mean    :5.781   Mean    :3.245   Mean    :3.460   Mean    :1.145
##   3rd Qu.:6.700   3rd Qu.:3.400   3rd Qu.:5.375   3rd Qu.:2.075
##   Max.    :7.200   Max.    :3.400   Max.    :6.000   Max.    :2.500
##               Species

```

```
##  setosa      :20
##  versicolor: 8
##  virginica :14
##
```

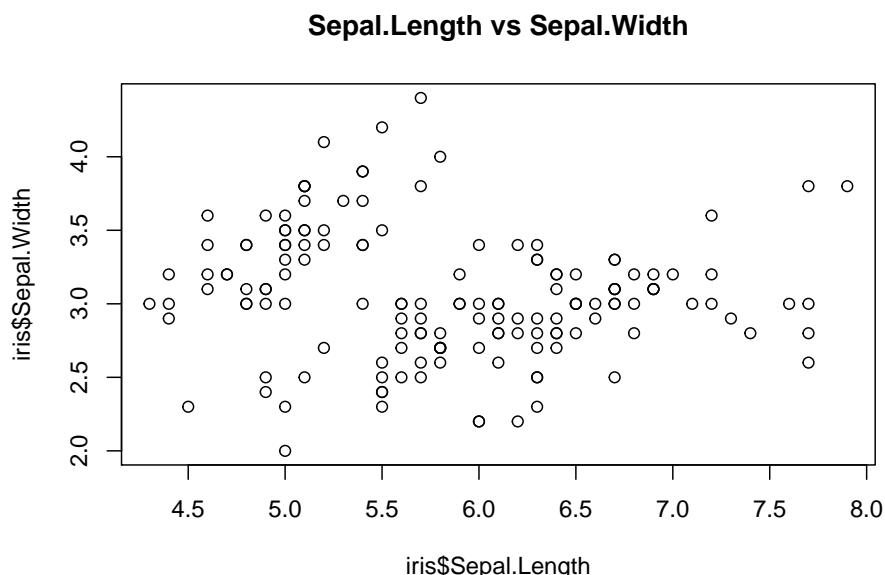
 Do the following for the `iris` dataset:

- Compute the subset corresponding to `Petal.Length` either smaller than 1.5 or larger than 2. Save this dataset as `irisPetal`.
- Compute and summarize a linear regression of `Sepal.Width` into `Petal.Width + Petal.Length` for the dataset `irisPetal`. What is the  $R^2$ ? *Solution: 0.101*.
- Check that the previous model is the same as regressing `Sepal.Width` into `Petal.Width + Petal.Length` for the dataset `iris` with the appropriate subset expression.
- Compute the variance for `Petal.Width` when `Petal.Width` is smaller or equal than 1.5 and larger than 0.3. *Solution: 0.1266541*.

## Plotting functions

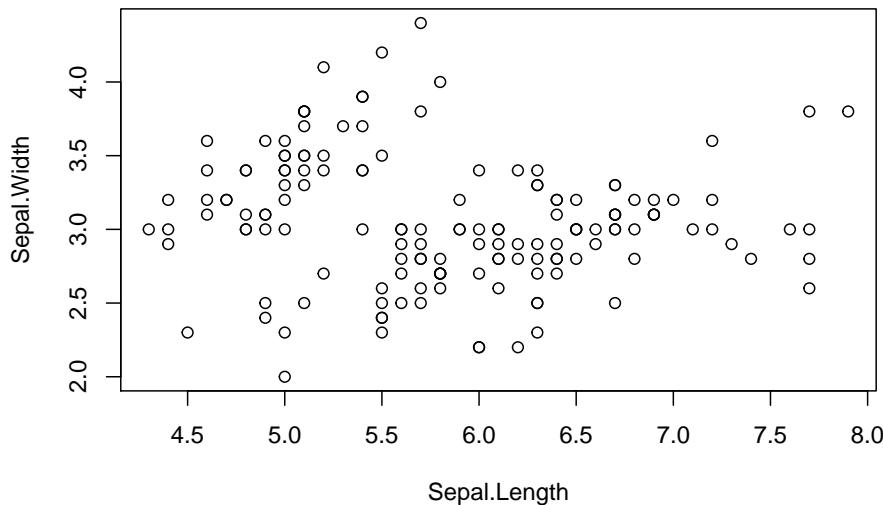
```
# "plot" is the main function for plotting in R
# It has a different behaviour depending on the kind of object that it receives

# How to plot some data
plot(iris$Sepal.Length, iris$Sepal.Width, main = "Sepal.Length vs Sepal.Width")
```

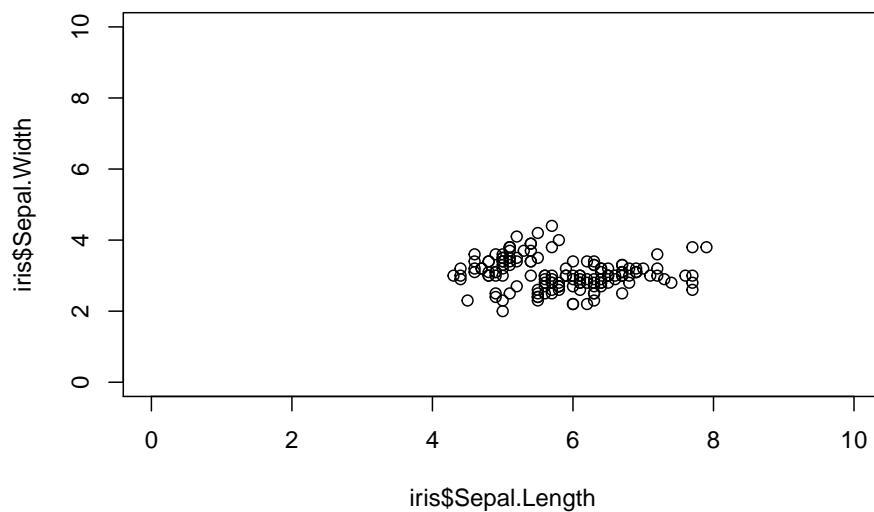


```
# Alternatively
plot(iris[, 1:2], main = "Sepal.Length vs Sepal.Width")
```

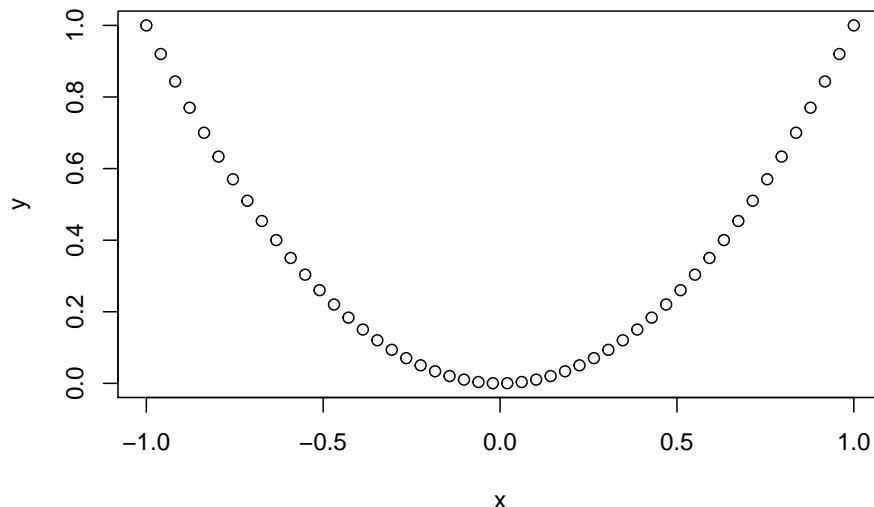
### Sepal.Length vs Sepal.Width



```
# Change the axis limits
plot(iris$Sepal.Length, iris$Sepal.Width, xlim = c(0, 10), ylim = c(0, 10))
```

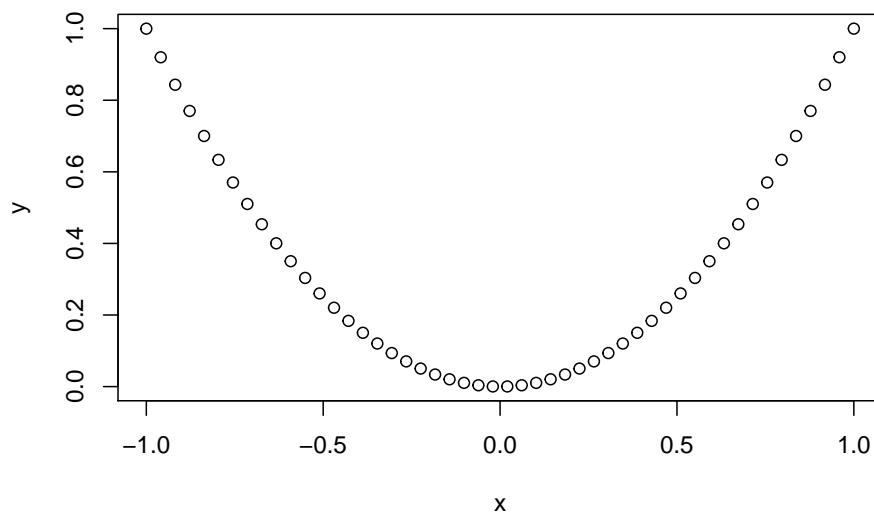


```
# How to plot a curve (a parabola)
x <- seq(-1, 1, l = 50)
y <- x^2
plot(x, y)
```

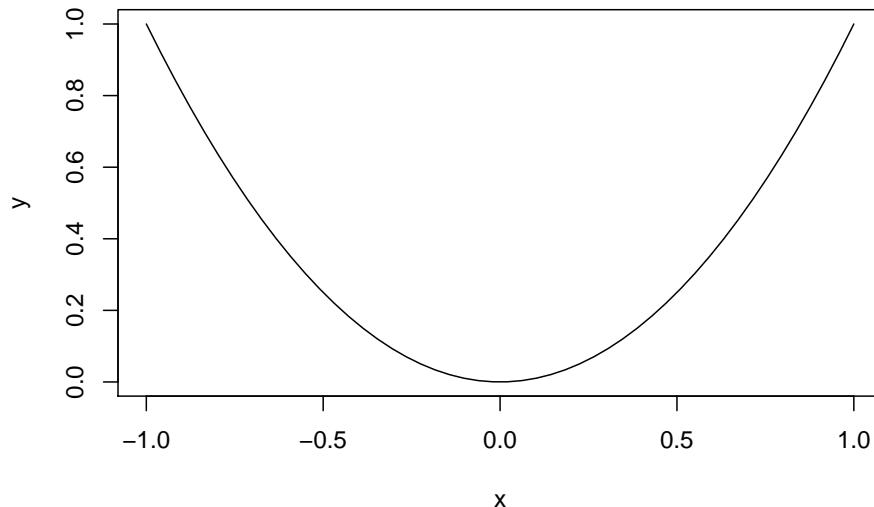


```
plot(x, y, main = "A dotted parabola")
```

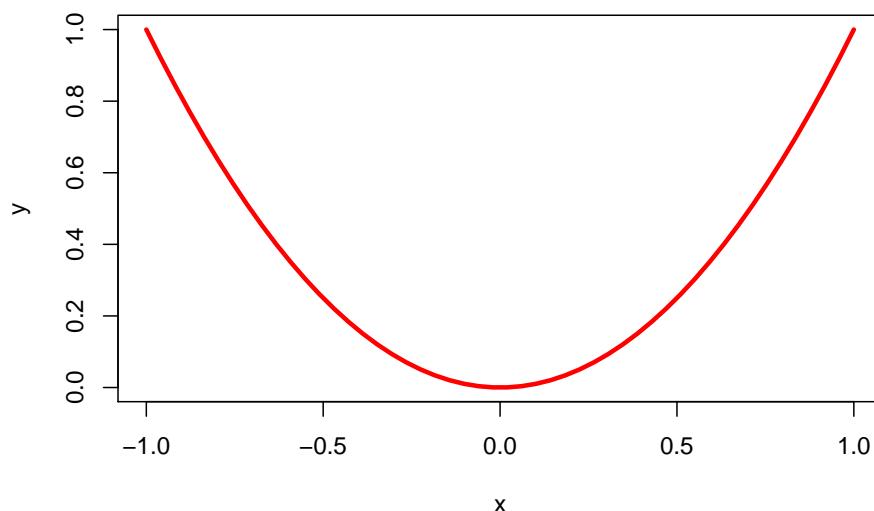
**A dotted parabola**



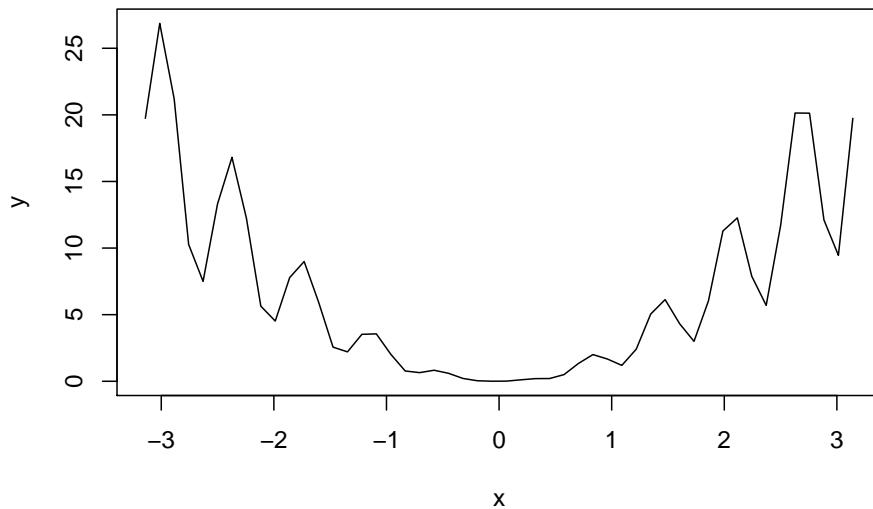
```
plot(x, y, main = "A parabola", type = "l")
```

**A parabola**

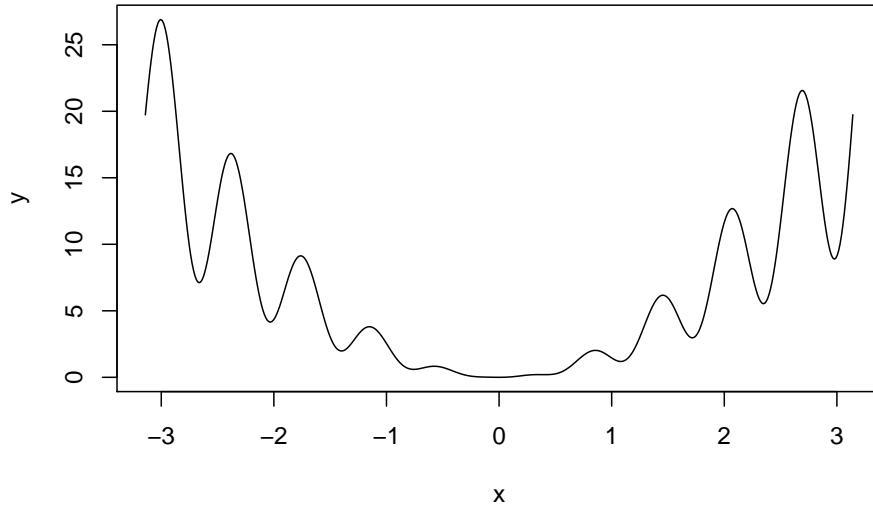
```
plot(x, y, main = "A red and thick parabola", type = "l", col = "red", lwd = 3)
```

**A red and thick parabola**

```
# Plotting a more complicated curve between -pi and pi
x <- seq(-pi, pi, l = 50)
y <- (2 + sin(10 * x)) * x^2
plot(x, y, type = "l") # Kind of rough...
```



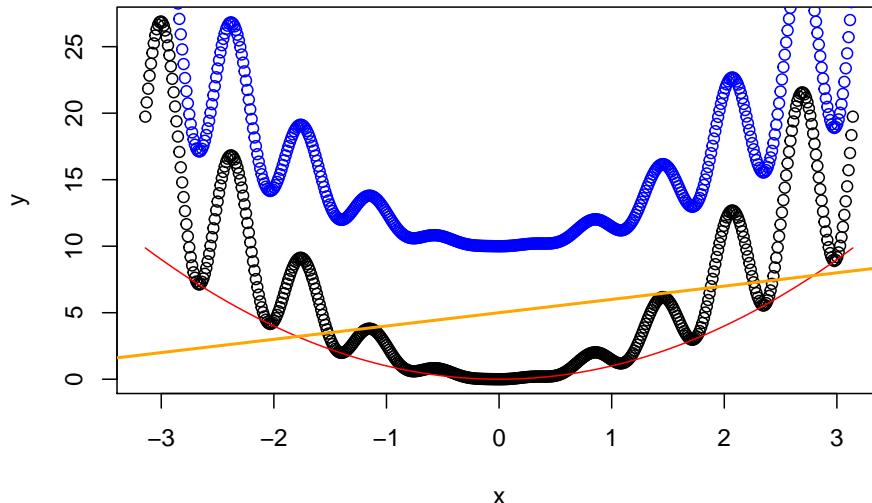
```
# Remember that we are joining points for creating a curve!
# More detailed plot
x <- seq(-pi, pi, l = 500)
y <- (2 + sin(10 * x)) * x^2
plot(x, y, type = "l")
```



```
# For more options in the plot customization see
?plot
?par

# "plot" is a first level plotting function. That means that whenever is called,
# it creates a new plot. If we want to add information to an existing plot, we
# have to use a second level plotting function such as "points", "lines" or "abline"

plot(x, y) # Create a plot
lines(x, x^2, col = "red") # Add lines
points(x, y + 10, col = "blue") # Add points
abline(a = 5, b = 1, col = "orange", lwd = 2) # Add a straight line  $y = a + b * x$ 
```



Do the following:

- Plot the `faithful` dataset.
- Add the straight line  $y = 110 - 15x$  (red).
- Make a new plot for the function  $y = \sin(x)$  (black). Add  $y = \sin(2x)$  (red),  $y = \sin(3x)$  (blue), and  $y = \sin(4x)$  (orange).

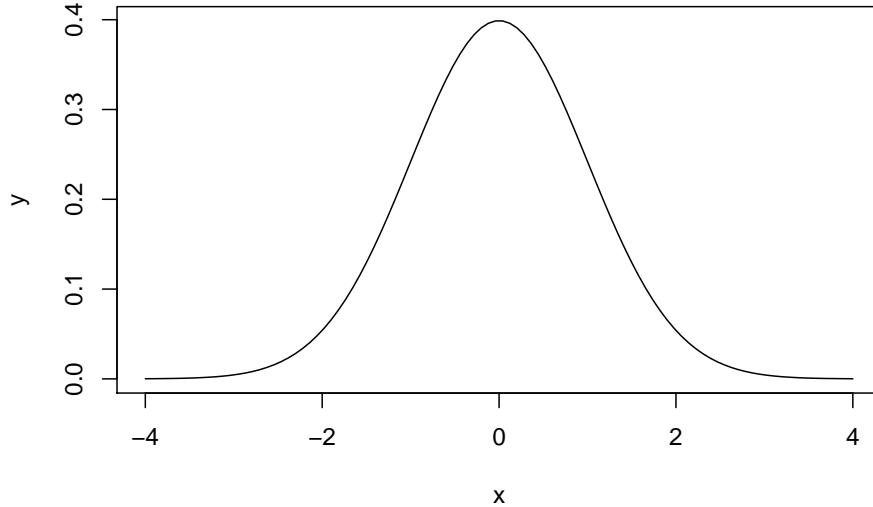
## Distributions

```
# R allows to sample [r], compute density/probability mass functions [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
# - chisq -> Chi squared
# - pois -> Poisson
# - binom -> Binomial
# More distributions:
?Distributions

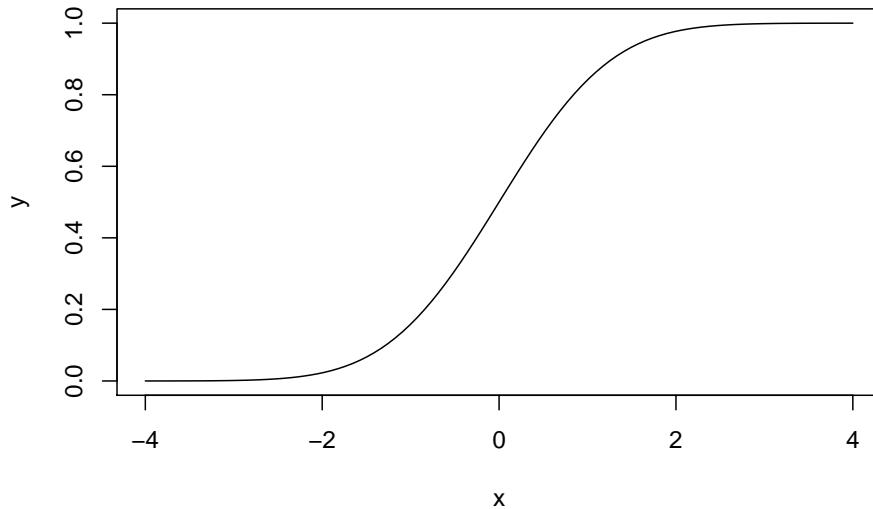
# Sampling from a Normal - 100 random points from a N(0, 1)
rnorm(n = 10, mean = 0, sd = 1)
## [1] -1.12055305 -0.06232391  0.71414179  0.77304492  1.98370559
## [6]  0.88127216 -1.15256313  0.55526743  0.26835801  1.26793195

# 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)
## [1]  1.4404800 -0.7195761  0.6709784 -0.4219485  0.3782196 -1.6665864
## [7] -0.5082030  0.4433822 -1.7993868 -0.6179521
```

```
# 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")
```



```
# Computing the 95% quantile for a  $N(0, 1)$ 
qnorm(p = 0.95, mean = 0, sd = 1)
## [1] 1.644854

# 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

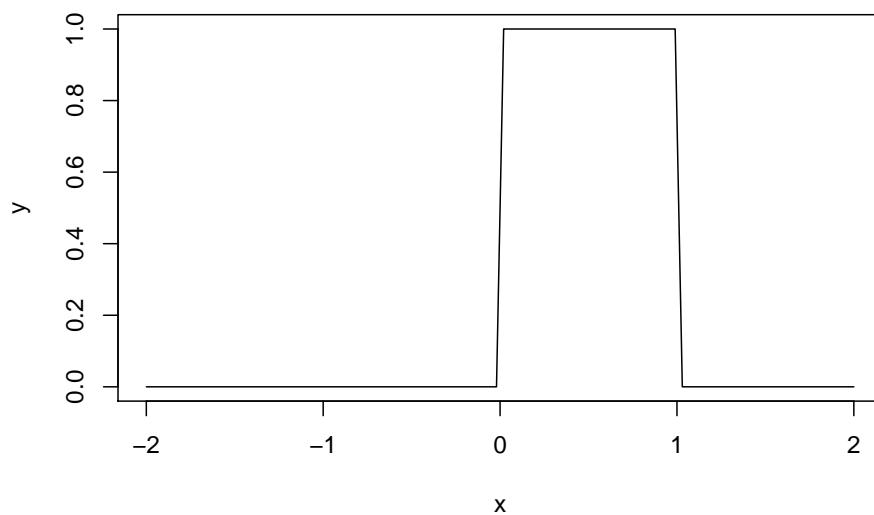
# Sampling from a  $U(0, 1)$ 
set.seed(45678)
```

```

runif(n = 10, min = 0, max = 1)
## [1] 0.9251342 0.3339988 0.2358930 0.3366312 0.7488829 0.9327177 0.3365313
## [8] 0.2245505 0.6473663 0.0807549

# 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")

```



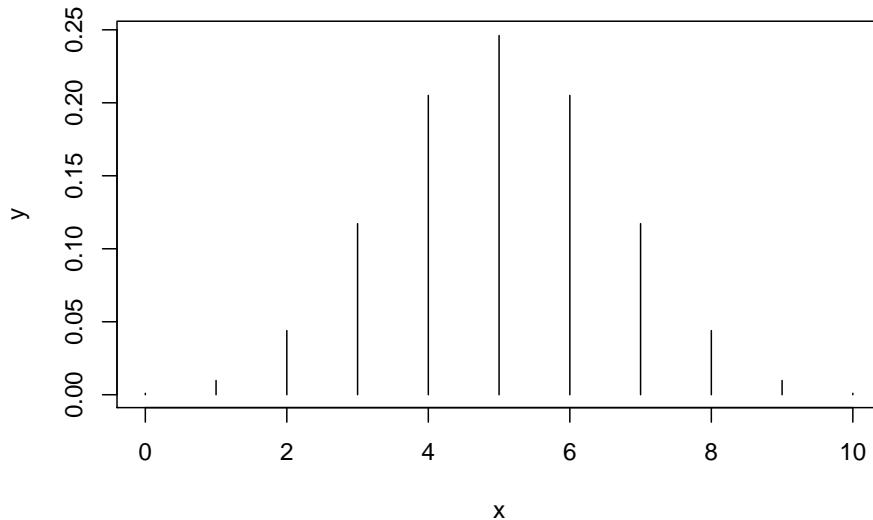
```

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

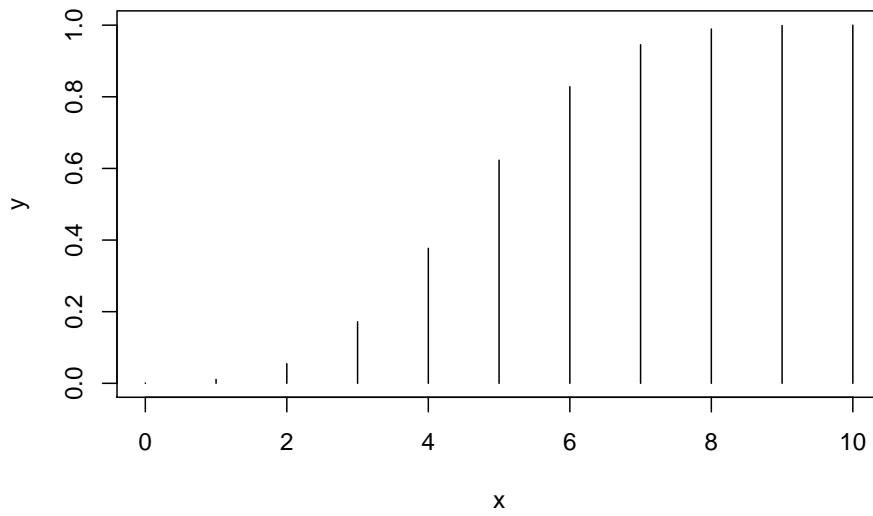
# Sampling from a  $Bi(10, 0.5)$ 
set.seed(45678)
samp <- rbinom(n = 200, size = 10, prob = 0.5)
table(samp) / 200
## samp
##    1     2     3     4     5     6     7     8     9
## 0.010 0.060 0.115 0.220 0.210 0.215 0.115 0.045 0.010

# Plotting the probability mass of a  $Bi(10, 0.5)$ 
x <- 0:10
y <- dbinom(x = x, size = 10, prob = 0.5)
plot(x, y, type = "h") # Vertical bars

```



```
# Plotting the distribution function of a Bi(10, 0.5)
x <- 0:10
y <- pbinom(q = x, size = 10, prob = 0.5)
plot(x, y, type = "h")
```



Do the following:

- Compute the 90%, 95% and 99% quantiles of a  $F$  distribution with  $df1 = 1$  and  $df2 = 5$ .  
*Answer: c(4.060420, 6.607891, 16.258177).*
- Plot the distribution function of a  $U(0,1)$ . Does it make sense with its density function?
- Sample 100 points from a Poisson with  $\lambda = 5$ .
- Sample 100 points from a  $U(-1,1)$  and compute its mean.
- Plot the density of a  $t$  distribution with  $df = 1$  (use a sequence spanning from -4 to 4). Add lines of different colors with the densities for  $df = 5$ ,  $df = 10$ ,  $df = 50$  and  $df = 100$ . Do you see any pattern?

## Functions

```

# A function is a way of encapsulating a block of code so it can be reused easily
# They are useful for simplifying repetitive tasks and organize the analysis

# This is a silly function that takes x and y and returns its sum
# Note the use of "return" to indicate what should be returned
add <- function(x, y) {
  z <- x + y
  return(z)
}

# Calling add - you need to run the definition of the function first!
add(x = 1, y = 2)
## [1] 3
add(1, 1) # Arguments names can be omitted
## [1] 2

# A more complex function: computes a linear model and its posterior summary.
# Saves us a few keystrokes when computing a lm and a summary
lmSummary <- function(formula, data) {
  model <- lm(formula = formula, data = data)
  summary(model)
}
# If no return(), the function returns the value of the last expression

# Usage
lmSummary(Sepal.Length ~ Petal.Width, iris)
##
## Call:
## lm(formula = formula, data = data)
##
## Residuals:
##       Min     1Q Median     3Q    Max
## -1.38822 -0.29358 -0.04393  0.26429  1.34521
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 4.77763   0.07293  65.51   <2e-16 ***
## Petal.Width  0.88858   0.05137  17.30   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.478 on 148 degrees of freedom
## Multiple R-squared:  0.669,  Adjusted R-squared:  0.6668 
## F-statistic: 299.2 on 1 and 148 DF,  p-value: < 2.2e-16

# Recall: there is no variable called model in the workspace.
# The function works on its own workspace!
model
## Error in eval(expr, envir, enclos): object 'model' not found

# Add a line to a plot

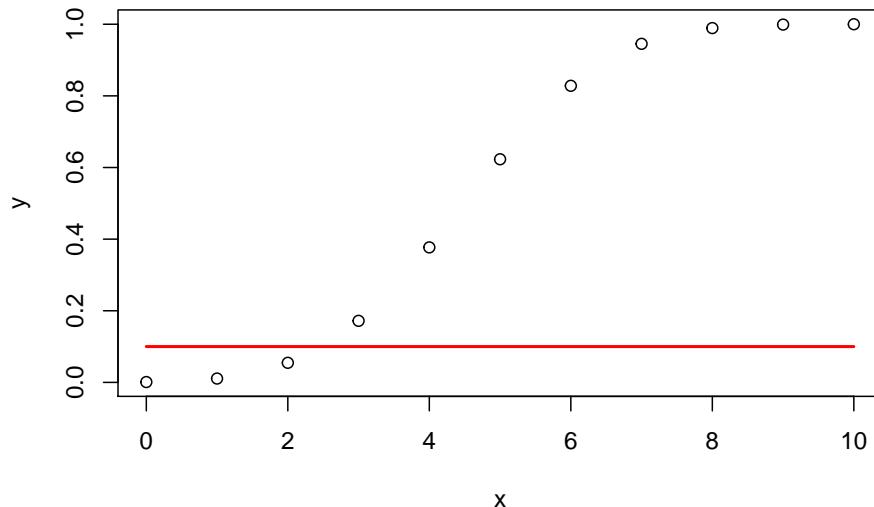
```

```

addLine <- function(x, beta0, beta1) {
  lines(x, beta0 + beta1 * x, lwd = 2, col = 2)
}

# Usage
plot(x, y)
addLine(x, beta0 = 0.1, beta1 = 0)

```



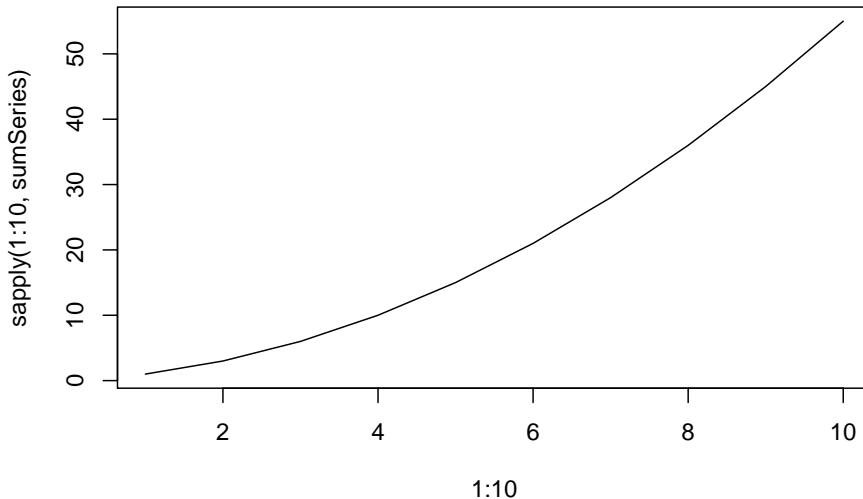
```

# The function "sapply" allows to sequentially apply a function
sapply(1:10, sqrt)
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
## [8] 2.828427 3.000000 3.162278
sqrt(1:10) # The same
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
## [8] 2.828427 3.000000 3.162278

# The advantage of "sapply" is that you can use with any function
myFun <- function(x) c(x, x^2)
sapply(1:10, myFun) # Returns a 2 x 10 matrix
## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 1 2 3 4 5 6 7 8 9 10
## [2,] 1 4 9 16 25 36 49 64 81 100

# "sapply" is useful for plotting non-vectorized functions
sumSeries <- function(n) sum(1:n)
plot(1:10, sapply(1:10, sumSeries), type = "l")

```



```

# "apply" applies iteratively a function to rows (1) or columns (2)
# of a matrix or data frame
A <- matrix(1:10, nrow = 5, ncol = 2)
A
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
apply(A, 1, sum) # Applies the function by rows
## [1] 7 9 11 13 15
apply(A, 2, sum) # By columns
## [1] 15 40

# With other functions
apply(A, 1, sqrt)
##      [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.00000 1.414214 1.732051 2.236068
## [2,] 2.44949 2.645751 2.828427 3.162278
apply(A, 2, function(x) x^2)
##      [,1] [,2]
## [1,]    1   36
## [2,]    4   49
## [3,]    9   64
## [4,]   16   81
## [5,]   25  100

```



Do the following:

- Create a function that takes as argument  $n$  and returns the value of  $\sum_{i=1}^n i^2$ .
- Create a function that takes as input the argument  $N$  and then plots the curve  $(n, \sum_{i=1}^n \sqrt{i})$  for  $n = 1, \dots, N$ . Hint: use `sapply`.

## Control structures

```

# The "for" statement allows to create loops that run along a given vector
# Prints 5 times a message (i varies in 1:5)
for (i in 1:5) {
  print(i)
}
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5

# Another example
a <- 0
for (i in 1:3) {
  a <- i + a
}
a
## [1] 6

# Nested loops are possible
A <- matrix(0, nrow = 2, ncol = 3)
for (i in 1:2) {
  for (j in 1:3) {
    A[i, j] <- i + j
  }
}

# The "if" statement allows to create conditional structures of the forms:
# if (condition) {
#   # Something
# } else {
#   # Something else
# }
# These structures are thought to be inside functions

# Is the number positive?
isPositive <- function(x) {
  if (x > 0) {
    print("Positive")
  } else {
    print("Not positive")
  }
}
isPositive(1)
## [1] "Positive"
isPositive(-1)
## [1] "Not positive"

# A loop can be interrupted with the "break" statement
# Stop when x is above 100
x <- 1

```

```

for (i in 1:1000) {
  x <- (x + 0.01) * x
  print(x)
  if (x > 100) {
    break
  }
}
## [1] 1.01
## [1] 1.0302
## [1] 1.071614
## [1] 1.159073
## [1] 1.35504
## [1] 1.849685
## [1] 3.439832
## [1] 11.86684
## [1] 140.9406

```



Do the following:

- Compute  $\mathbf{C}_{n \times k}$  in  $\mathbf{C}_{n \times k} = \mathbf{A}_{n \times m} \mathbf{B}_{m \times k}$  from  $\mathbf{A}$  and  $\mathbf{B}$ . Use that  $c_{i,j} = \sum_{l=1}^m a_{i,l} b_{l,j}$ . Test the implementation with simple examples.
- Create a function that samples a  $\mathcal{N}(0, 1)$  and returns the first sampled point that is larger than 4.
- Create a function that simulates  $N$  samples from the distribution of  $\max(X_1, \dots, X_n)$  where  $X_1, \dots, X_n$  are iid  $\mathcal{U}(0, 1)$ .

## Appendix D

# 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 E

# Least squares and maximum likelihood estimation

Least squares had a prominent role in linear models. Why is so? After all, it is a purely *geometrical argument* for fitting a plane to a cloud of points. It is not motivated as a *statistical procedure* for estimating the unknown parameters  $\beta$ ... at least apparently.

However, **least squares estimation is equivalent to maximum likelihood estimation** under the assumptions of the model seen in Section 2.3 (Normality is especially important here due to the squares present in the exponential of the Normal pdf). So maximum likelihood estimation, the most well-known statistical estimation method, is behind least squares if the assumptions of the model hold.

First, recall that given the sample  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$ , due to the assumptions introduced in Section 2.3, we have that:

$$Y_i | (X_{i1} = x_{i1}, \dots, X_{ip} = x_{ip}) \sim \mathcal{N}(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}, \sigma^2), \quad i = 1, \dots, n$$

with  $Y_1, \dots, Y_n$  being independent conditionally on the sample of predictors. Equivalently stated in a compact matrix way:

$$\mathbf{Y} | \mathbf{X} \sim \mathcal{N}_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}).$$

From these two equations we can obtain the log-likelihood function of  $Y_1, \dots, Y_n$  conditionally<sup>1</sup> on  $\mathbf{X}_1, \dots, \mathbf{X}_n$  as

$$\ell(\boldsymbol{\beta}) = \log \phi(\mathbf{Y}; \mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}) = \sum_{i=1}^n \log \phi(Y_i; (\mathbf{X}\boldsymbol{\beta})_i, \sigma). \quad (\text{E.1})$$

Now we are ready to show the next result.

**Theorem E.1.** *Under assumptions i–iv in Section 2.3, the maximum likelihood estimate of  $\boldsymbol{\beta}$  is the least squares estimate (2.7):*

$$\hat{\boldsymbol{\beta}}_{\text{ML}} = \arg \max_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \ell(\boldsymbol{\beta}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{Y}.$$

---

<sup>1</sup>We assume that the randomness is on the response only.

*Proof.* Expanding the first equality at (E.1) gives (recall that  $|\sigma^2 \mathbf{I}|^{1/2} = \sigma^n$ )

$$\ell(\boldsymbol{\beta}) = -\log((2\pi)^{n/2} \sigma^n) - \frac{1}{2\sigma^2} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})' (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

In order to differentiate with respect to  $\boldsymbol{\beta}$ , we use that, for two vector-valued functions  $f$  and  $g$ :

$$\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A} \text{ and } \frac{\partial f(\mathbf{x})' g(\mathbf{x})}{\partial \mathbf{x}} = f(\mathbf{x})' \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} + g(\mathbf{x})' \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}.$$

Then, differentiating with respect to  $\boldsymbol{\beta}$  and equating to zero gives

$$\frac{1}{\sigma^2} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})' \mathbf{X} = \frac{1}{\sigma^2} (\mathbf{Y}' \mathbf{X} - \boldsymbol{\beta}' \mathbf{X}' \mathbf{X}) = 0.$$

This means that optimizing  $\ell$  does not require knowledge on  $\sigma^2$ ! Nicely, solving the above equation yields

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}' \mathbf{X})^{-1} \mathbf{X} \mathbf{Y}.$$

□



Maximum likelihood estimation is **asymptotically optimal** when estimating unknown parameters of a model. This is a very appealing property which means that, when the sample size  $n$  is large, it is **guaranteed to perform better than any other estimation method**, where *better* is understood in terms of the mean squared error.

## Appendix F

# Dealing with missing data

Missing data, codified as `NA` in R, can be problematic in predictive modeling. By default, most of the regression models in R work with the complete cases of the data, this is, **they exclude the cases in which there is at least one `NA`**. This may be problematic in certain cases. Let's see an example.

```
# The airquality dataset contains NA's
data("airquality")
head(airquality)
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5    1
## 2    36     118  8.0   72     5    2
## 3    12     149 12.6   74     5    3
## 4    18     313 11.5   62     5    4
## 5    NA      NA 14.3   56     5    5
## 6    28      NA 14.9   66     5    6
summary(airquality)
##      Ozone          Solar.R          Wind          Temp
##  Min.   : 1.00   Min.   : 7.0   Min.   : 1.700   Min.   :56.00
##  1st Qu.:18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00
##  Median :31.50   Median :205.0   Median : 9.700   Median :79.00
##  Mean   :42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88
##  3rd Qu.:63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00
##  Max.   :168.00  Max.   :334.0   Max.   :20.700   Max.   :97.00
##  NA's   :37      NA's   :7
##      Month          Day
##  Min.   :5.000   Min.   : 1.0
##  1st Qu.:6.000   1st Qu.: 8.0
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
##
# Let's add more NA's for the sake of illustration
set.seed(123456)
airquality$Solar.R[runif(nrow(airquality)) < 0.7] <- NA
airquality$Day[runif(nrow(airquality)) < 0.1] <- NA
# See what are the fully-observed cases
```

```

comp <- complete.cases(airquality)
mean(comp) # Only 15% of cases are fully observed
## [1] 0.1568627

# Complete cases
head(airquality[comp, ])
##   Ozone Solar.R Wind Temp Month Day
## 1     41     190  7.4   67     5    1
## 2     36     118  8.0   72     5    2
## 9      8     19 20.1   61     5    9
## 13    11     290  9.2   66     5   13
## 14    14     274 10.9   68     5   14
## 15    18      65 13.2   58     5   15

# Linear model on all the variables
summary(lm(Ozone ~ ., data = airquality)) # 129 not included
##
## Call:
## lm(formula = Ozone ~ ., data = airquality)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -23.790 -10.910 -2.249 10.960 33.246
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -71.50880  31.49703 -2.270 0.035704 *  
## Solar.R     -0.01112   0.03376 -0.329 0.745596    
## Wind        -0.61129   0.96113 -0.636 0.532769    
## Temp         1.82870   0.42224  4.331 0.000403 *** 
## Month       -2.86513   2.22222 -1.289 0.213614    
## Day         -0.28710   0.41700 -0.688 0.499926    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.12 on 18 degrees of freedom
##   (129 observations deleted due to missingness)
## Multiple R-squared:  0.5957, Adjusted R-squared:  0.4833 
## F-statistic: 5.303 on 5 and 18 DF,  p-value: 0.00362

# Caution! Even if the problematic variable is excluded, only the complete
# observations are employed
summary(lm(Ozone ~ . - Solar.R, data = airquality))
##
## Call:
## lm(formula = Ozone ~ . - Solar.R, data = airquality)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -24.43 -11.56 -1.67 11.19 33.11
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    

```

```

## (Intercept) -72.2501 30.6707 -2.356 0.029386 *
## Wind -0.6236 0.9376 -0.665 0.514001
## Temp 1.7980 0.4021 4.472 0.000261 ***
## Month -2.6533 2.0767 -1.278 0.216762
## Day -0.2944 0.4065 -0.724 0.477851
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.74 on 19 degrees of freedom
## (129 observations deleted due to missingness)
## Multiple R-squared: 0.5932, Adjusted R-squared: 0.5076
## F-statistic: 6.927 on 4 and 19 DF, p-value: 0.001291

# Notice the difference with
summary(lm(Ozone ~ ., data = subset(airquality, select = -Solar.R)))
##
## Call:
## lm(formula = Ozone ~ ., data = subset(airquality, select = -Solar.R))
##
## Residuals:
##     Min      1Q      Median      3Q      Max
## -42.677 -12.609  -3.125  11.993  98.805
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -74.4456   25.8567  -2.879  0.00488 ** 
## Wind        -3.1064    0.7028  -4.420 2.51e-05 ***
## Temp         2.1666    0.2876   7.534 2.25e-11 ***
## Month        -3.6493   1.6343  -2.233  0.02778 *  
## Day          0.3162    0.2549   1.241  0.21768    
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.12 on 100 degrees of freedom
## (48 observations deleted due to missingness)
## Multiple R-squared: 0.5963, Adjusted R-squared: 0.5801
## F-statistic: 36.92 on 4 and 100 DF, p-value: < 2.2e-16

# Model selection can be problematic with missing data, since the number
# of complete cases changes with the addition or removing of predictors
mod <- lm(Ozone ~ ., data = airquality)

# stepAIC drops an error
modAIC <- stepAIC(mod)
## Start: AIC=138.55
## Ozone ~ Solar.R + Wind + Temp + Month + Day
##
##             Df Sum of Sq    RSS    AIC
## - Solar.R  1      28.2 4708.1 136.70
## - Wind     1     105.2 4785.0 137.09
## - Day      1     123.2 4803.1 137.18
## <none>          4679.9 138.55
## - Month    1     432.2 5112.1 138.67

```

```

## - Temp      1  4876.6 9556.5 153.69
## Error in stepAIC(mod): number of rows in use has changed: remove missing values?

# Also, this will be problematic (the number of complete cases changes
# with the predictors considered!)
modBIC <- stepAIC(mod, k = log(nrow(airquality)))
## Start:  AIC=156.73
## Ozone ~ Solar.R + Wind + Temp + Month + Day
##
##          Df  Sum of Sq    RSS    AIC
## - Solar.R  1      28.2 4708.1 151.85
## - Wind     1     105.2 4785.0 152.24
## - Day      1     123.2 4803.1 152.33
## - Month    1     432.2 5112.1 153.82
## <none>          4679.9 156.73
## - Temp     1  4876.6 9556.5 168.84
## Error in stepAIC(mod, k = log(nrow(airquality))): number of rows in use has changed: remove missing values

# Comparison of AICs or BICs is spurious: the scale of the likelihood changes
# with the sample size (the likelihood decreases with n), which increases
# AIC / BIC with n. Hence using BIC / AIC is not adequate for model selection
# with missing data.
AIC(lm(Ozone ~ ., data = airquality))
## [1] 208.6604
AIC(lm(Ozone ~ ., data = subset(airquality, select = -Solar.R)))
## [1] 955.0681

# Considers only complete cases including Solar.R
AIC(lm(Ozone ~ . - Solar.R, data = airquality))
## [1] 206.8047

```

We have seen the problems that missing data may cause in regression models. There are many techniques designed to handle missing data, depending on the missing data mechanism (whether it is completely at random or there is some pattern in the missing process) and the approach to impute the data (parametric, nonparametric, Bayesian, etc). We do not give an exhaustive view of the topic here, but we outline **three concrete approaches to handle missing data in practice**:

1. **Use complete cases.** This is the simplest solution and can be achieved by restricting the analysis to the set of fully-observed observations. The advantage of this solution is that it can be implemented very easily by using the `complete.cases` or `na.omit` f, and therefore the precision of the estimates will be much lower. In addition, it may lead to a biased representation of the original data (if the missing process is associated with the values of the predictors).
2. **Remove predictors with many missing data.** This is another simple solution in case most of the missing data is concentrated in one predictor.
3. **Use imputation for the missing values.** The idea is to replace the missing observations on the response or the predictors with suitable values:
  - When the response is missing, we can use a **predictive model to predict the missing response**, then create a new fully-observed dataset containing the predictions instead of the missing values, and finally re-estimate the predictive model in this expanded dataset. This approach is attractive if most of the missing data is in the response (see next point).
  - When different predictors and the response are missing, we can use a direct imputation for them. The simplest approach is to **replace the missing data with the sample mean** of the observed cases (in the case of quantitative variables). This and **more sophisticated imputation methods**, based on predictive models, are available within the `mice` package (van Buuren and

Groothuis-Oudshoorn, 2011). This approach is interesting if the data contains many NAs scattered in different predictors (hence a complete cases analysis will be inefficient).

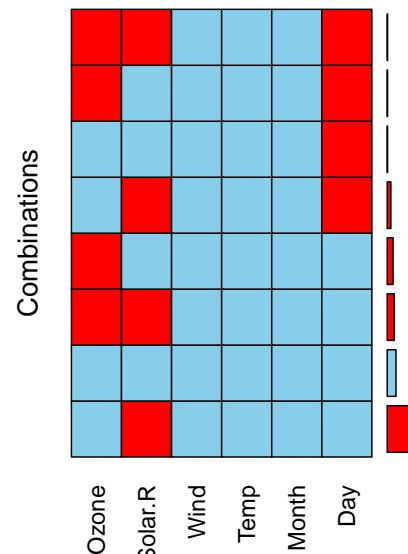
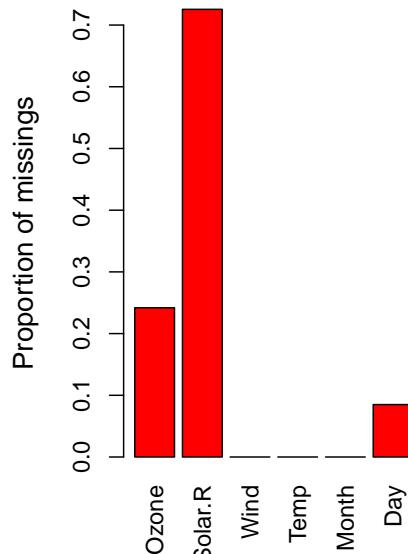
Let's put in practise these three approaches in the previous example.

```
# The complete cases approach is the default in R
summary(lm(Ozone ~ ., data = airquality))
##
## Call:
## lm(formula = Ozone ~ ., data = airquality)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.790 -10.910 -2.249 10.960 33.246
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -71.50880  31.49703 -2.270 0.035704 *
## Solar.R     -0.01112   0.03376 -0.329 0.745596
## Wind        -0.61129   0.96113 -0.636 0.532769
## Temp         1.82870   0.42224  4.331 0.000403 ***
## Month        -2.86513   2.22222 -1.289 0.213614
## Day          -0.28710   0.41700 -0.688 0.499926
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.12 on 18 degrees of freedom
## (129 observations deleted due to missingness)
## Multiple R-squared:  0.5957, Adjusted R-squared:  0.4833
## F-statistic: 5.303 on 5 and 18 DF,  p-value: 0.00362

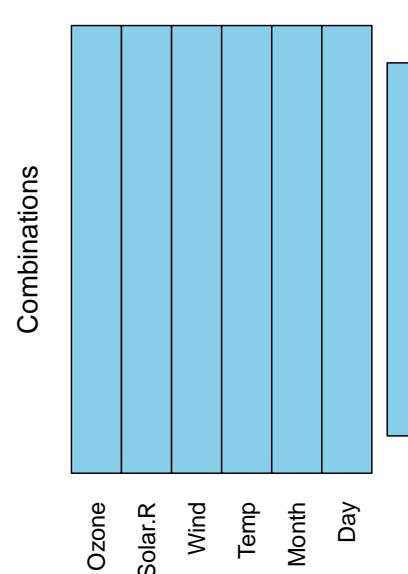
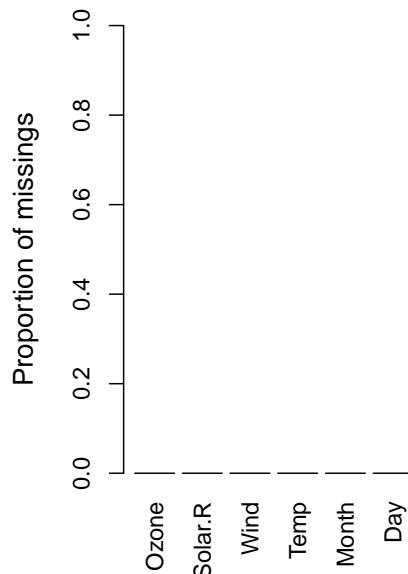
# However, since the complete cases that R is going to consider depends on
# which predictors are included, it is safer to exclude NA's explicitly
# before the fitting of a model
airqualityNoNA <- na.exclude(airquality)
summary(airqualityNoNA)
##      Ozone          Solar.R          Wind          Temp
## Min.   : 8.00   Min.   :13.0   Min.   : 6.300   Min.   :58.00
## 1st Qu.:13.00   1st Qu.:61.5   1st Qu.: 9.575   1st Qu.:65.50
## Median :17.00   Median :178.5   Median :11.500   Median :71.50
## Mean   :28.21   Mean   :161.8   Mean   :11.887   Mean   :72.54
## 3rd Qu.:36.25   3rd Qu.:255.0   3rd Qu.:14.300   3rd Qu.:79.50
## Max.   :96.00   Max.   :334.0   Max.   :20.700   Max.   :97.00
##
##      Month          Day
## Min.   :5.00   Min.   : 1.00
## 1st Qu.:5.00   1st Qu.:11.25
## Median :6.00   Median :16.50
## Mean   :6.75   Mean   :15.79
## 3rd Qu.:9.00   3rd Qu.:20.25
## Max.   :9.00   Max.   :29.00

# The package VIM has a function to visualize where the missing data is
# present. It gives the percentage of NA's for each variable and most
# important combinations of NA's
library(VIM)
```

```
aggr(airquality)
```



```
aggr(airqualityNoNA)
```



```
# Stepwise regression without NA's - no problem
modBIC1 <- stepAIC(lm(Ozone ~ ., data = airqualityNoNA),
                     k = log(nrow(airqualityNoNA)), trace = 0)
summary(modBIC1)
##
## Call:
## lm(formula = Ozone ~ Temp, data = airqualityNoNA)
##
## Residuals:
##      Min      1Q  Median      3Q      Max
## -27.645 -13.180  -0.136   8.926  37.666
##
## Coefficients:
```

```

##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -90.1846    24.6414  -3.660  0.00138 **
## Temp         1.6321     0.3367   4.847 7.63e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.95 on 22 degrees of freedom
## Multiple R-squared:  0.5164, Adjusted R-squared:  0.4944
## F-statistic: 23.49 on 1 and 22 DF,  p-value: 7.634e-05

# But we only take into account 16% of the original data
nrow(airqualityNoNA) / nrow(airquality)
## [1] 0.1568627

# Removing the predictor with many NA's, as we did before
# We also exclude NA's from other predictors
airqualityNoSolar.R <- na.exclude(subset(airquality, select = -Solar.R))
modBIC2 <- stepAIC(lm(Ozone ~ ., data = airqualityNoSolar.R),
                     k = log(nrow(airqualityNoSolar.R)), trace = 0)
summary(modBIC2)
##
## Call:
## lm(formula = Ozone ~ Wind + Temp + Month, data = airqualityNoSolar.R)
##
## Residuals:
##    Min     1Q     Median     3Q     Max
## -43.973 -14.170  -3.107  10.638 102.297
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -67.7279    25.3507  -2.672  0.0088 **
## Wind        -3.0439     0.7028  -4.331 3.51e-05 ***
## Temp         2.1323     0.2870   7.429 3.59e-11 ***
## Month        -3.6167    1.6384  -2.207   0.0295 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.17 on 101 degrees of freedom
## Multiple R-squared:  0.5901, Adjusted R-squared:  0.5779
## F-statistic: 48.46 on 3 and 101 DF,  p-value: < 2.2e-16
# In this example the approach works well because most of the NA's are
# associated to the variable Solar.R

# Imput data using the sample mean
library(mice)
airqualityMean <- complete(mice(data = airquality,
                                 m = 1, method = "mean"))
##
##   iter imp variable
## 1   1   Ozone  Solar.R  Day
## 2   1   Ozone  Solar.R  Day
## 3   1   Ozone  Solar.R  Day
## 4   1   Ozone  Solar.R  Day

```

```

## 5 1 Ozone Solar.R Day
head(airqualityMean)
##      Ozone Solar.R Wind Temp Month      Day
## 1 41.00000 190.0000 7.4   67      5 1.00000
## 2 36.00000 118.0000 8.0   72      5 2.00000
## 3 12.00000 181.7857 12.6  74      5 3.00000
## 4 18.00000 181.7857 11.5  62      5 15.62857
## 5 42.12931 181.7857 14.3  56      5 5.00000
## 6 28.00000 181.7857 14.9  66      5 15.62857

# Explanation of the syntax:
# - complete() serves to retrieve the completed dataset from the mids object
# - m = 1 specifies that we only want a reconstruction of the dataset
#   because the imputation method is deterministic (it can have randomness)
# - method = "mean" says that we want the sample mean to be used to fill
#   NA's in all the columns. This only works properly for quantitative variables.

# Impute using linear regression for the response (first column) and mean
# for the predictors (remaining five columns)
airqualityLm <- complete(mice(data = airquality, m = 1,
                                method = c("norm.predict", rep("mean", 5))))
## iter imp variable
## 1 1 Ozone Solar.R Day
## 2 1 Ozone Solar.R Day
## 3 1 Ozone Solar.R Day
## 4 1 Ozone Solar.R Day
## 5 1 Ozone Solar.R Day
head(airqualityLm)
##      Ozone Solar.R Wind Temp Month      Day
## 1 41.00000 190.0000 7.4   67      5 1.00000
## 2 36.00000 118.0000 8.0   72      5 2.00000
## 3 12.00000 181.7857 12.6  74      5 3.00000
## 4 18.00000 181.7857 11.5  62      5 15.62857
## 5 -13.30742 181.7857 14.3  56      5 5.00000
## 6 28.00000 181.7857 14.9  66      5 15.62857

# Imputed data - some extrapolation problems may happen
airqualityLm$Ozone[is.na(airquality$Ozone)]
## [1] -13.3074185 33.4091565 -15.0741600 -6.4059375 15.1332432
## [6] 43.7243937 34.5263132 0.1587007 57.7324072 62.1124555
## [11] 30.1665972 70.4711454 71.9654463 75.6442705 38.1564742
## [16] 43.4948337 57.9854244 69.7469457 63.1063096 51.9523679
## [21] 50.0467842 56.8757982 41.2870224 49.6011029 33.0063081
## [26] 67.4279593 48.9713911 54.0391717 51.9323459 50.1573546
## [31] 46.1466219 71.2904665 49.9085704 39.0166836 26.9295520
## [36] 77.0666270 26.9453064

# Notice that the imputed data is the same (except for a small truncation that
# is introduced by mice) as
predict(lm(airquality$Ozone ~ ., data = airqualityMean),
       newdata = airqualityMean[is.na(airquality$Ozone), -1])
##      5          10         25         26         27         32
## -13.3537515 33.3839709 -15.1032941 -6.4344422 15.0745276 43.7185093

```

```

##      33      34      35      36      37      39
## 34.5128480 0.1488983 57.7500138 62.1313290 30.1891953 70.4905697
##      42      43      45      46      52      53
## 72.0216949 75.6909337 38.1841717 43.5104926 57.9764970 69.7165869
##      54      55      56      57      58      59
## 63.0884481 51.9374205 50.0400961 56.8792515 41.2844048 49.6257521
##      60      61      65      72      75      83
## 33.0354266 67.4490679 48.9905942 54.0457773 51.9941898 50.1697328
##      84     102     103     107     115     119
## 46.1697595 71.3235772 49.9344726 39.0222991 26.9297719 77.0827188
##      150
## 26.9508942

# Removing the truncation with ridge = 0
complete(mice(data = airquality, m = 1,
               method = c("norm.predict", rep("mean", 5)),
               ridge = 0))[is.na(airquality$Ozone), 1]

##
##  iter imp variable
##  1   1  Ozone  Solar.R  Day
##  2   1  Ozone  Solar.R  Day
##  3   1  Ozone  Solar.R  Day
##  4   1  Ozone  Solar.R  Day
##  5   1  Ozone  Solar.R  Day
## [1] -13.3537515 33.3839709 -15.1032941 -6.4344422 15.0745276
## [6] 43.7185093 34.5128480 0.1488983 57.7500138 62.1313290
## [11] 30.1891953 70.4905697 72.0216949 75.6909337 38.1841717
## [16] 43.5104926 57.9764970 69.7165869 63.0884481 51.9374205
## [21] 50.0400961 56.8792515 41.2844048 49.6257521 33.0354266
## [26] 67.4490679 48.9905942 54.0457773 51.9941898 50.1697328
## [31] 46.1697595 71.3235772 49.9344726 39.0222991 26.9297719
## [36] 77.0827188 26.9508942

# The default mice's method (predictive mean matching) works better in this
# case (in the sense that it does not yield negative Ozone values)
# Notice that there is randomness in the imputation!
airqualityMice <- complete(mice(data = airquality, m = 1, seed = 123))
##
##  iter imp variable
##  1   1  Ozone  Solar.R  Day
##  2   1  Ozone  Solar.R  Day
##  3   1  Ozone  Solar.R  Day
##  4   1  Ozone  Solar.R  Day
##  5   1  Ozone  Solar.R  Day
head(airqualityMice)
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5    1
## 2    36     118  8.0   72     5    2
## 3    12     273 12.6   74     5    3
## 4    18      47 11.5   62     5    8
## 5    19     264 14.3   56     5    5
## 6    28     264 14.9   66     5   27

```

See the available methods for `mice()` at `?mice`. There are specific methods for different kinds of variables

(numeric, factor with two levels, and factors of more than two levels) and fairly advanced imputation methods.

## Appendix G

# Multinomial logistic regression

The logistic model seen in Section 5.2.1 can be generalized to categorical variables  $Y$  with more than two possible levels, namely  $\{1, \dots, J\}$ . Given the predictors  $X_1, \dots, X_p$ , *multinomial logistic regression* models the probability of each level  $j$  of  $Y$  by

$$p_j(\mathbf{x}) := \mathbb{P}[Y = j | X_1 = x_1, \dots, X_p = x_p] = \frac{e^{\beta_{0j} + \beta_{1j}X_1 + \dots + \beta_{pj}X_p}}{1 + \sum_{l=1}^{J-1} e^{\beta_{0l} + \beta_{1l}X_1 + \dots + \beta_{pl}X_p}} \quad (\text{G.1})$$

for  $j = 1, \dots, J-1$  and (for the last level  $J$ )

$$p_J(\mathbf{x}) := \mathbb{P}[Y = J | X_1 = x_1, \dots, X_p = x_p] = \frac{1}{1 + \sum_{l=1}^{J-1} e^{\beta_{0l} + \beta_{1l}X_1 + \dots + \beta_{pl}X_p}}. \quad (\text{G.2})$$

Note that (G.1) and (G.2) imply that  $\sum_{j=1}^J p_j(\mathbf{x}) = 1$  and that there are  $(J-1) \times (p+1)$  coefficients ( $(J-1)$  intercepts and  $(J-1) \times p$  slopes). Also, (G.2) reveals that the last level,  $J$ , is given a different treatment. This is because it is the *reference level* (it could be a different one, but is tradition to choose the last one).

The multinomial logistic model has an interesting interpretation in terms of logistic regressions. Taking the quotient between (G.1) and (G.2) gives

$$\frac{p_j(\mathbf{x})}{p_J(\mathbf{x})} = e^{\beta_{0j} + \beta_{1j}X_1 + \dots + \beta_{pj}X_p} \quad (\text{G.3})$$

for  $j = 1, \dots, J-1$ . Therefore, applying a logarithm to both sides we have:

$$\log \frac{p_j(\mathbf{x})}{p_J(\mathbf{x})} = \beta_{0j} + \beta_{1j}X_1 + \dots + \beta_{pj}X_p. \quad (\text{G.4})$$

This equation is indeed very similar to (5.7). If  $J = 2$ , it is the same up to a change in the codes for the levels: the logistic regression giving the probability of  $Y = 1$  versus  $Y = 2$ . On the LHS of (G.4) we have the logarithm of the ratio of two probabilities and on the RHS a linear combination of the predictors. If the probabilities on the LHS were complementary (if they added up to one), then we would have a log-odds and hence a logistic regression for  $Y$ . This is not the situation, but it is close: instead of odds and log-odds,

we have *ratios* and *log-ratios* of non complementary probabilities. Also, it gives a good insight on what the multinomial logistic regression is: **a set of  $J-1$  independent “logistic regressions” for the probability of  $Y = j$  versus the probability of the reference  $Y = J$ .**

Equation (G.3) gives also interpretation on the coefficients of the model since

$$p_j(\mathbf{x}) = e^{\beta_{0j} + \beta_{1j}X_1 + \dots + \beta_{pj}X_p} p_J(\mathbf{x}).$$

Therefore:

- $e^{\beta_{0j}}$ : is the ratio between  $p_j(\mathbf{0})/p_J(\mathbf{0})$ , the probabilities of  $Y = j$  and  $Y = J$  when  $X_1 = \dots = X_p = 0$ . If  $e^{\beta_{0j}} > 1$  (equivalently,  $\beta_{0j} > 0$ ), then  $Y = j$  is more likely than  $Y = J$ . If  $e^{\beta_{0j}} < 1$  ( $\beta_{0j} < 0$ ), then  $Y = j$  is less likely than  $Y = J$ .
- $e^{\beta_{lj}}, l \geq 1$ : is the **multiplicative** increment of the ratio between  $p_j(\mathbf{x})/p_J(\mathbf{x})$  for an increment of one unit in  $X_l = x_l$ , provided that the remaining variables  $X_1, \dots, X_{l-1}, X_{l+1}, \dots, X_p$  do not change. If  $e^{\beta_{lj}} > 1$  (equivalently,  $\beta_{lj} > 0$ ), then  $Y = j$  becomes more likely than  $Y = J$  for each increment in  $X_j$ . If  $e^{\beta_{lj}} < 1$  ( $\beta_{lj} < 0$ ), then  $Y = j$  becomes less likely than  $Y = J$ .

The following code illustrates how to compute a basic multinomial regression in R.

```
# Package included in R that implements multinomial regression
library(nnet)

# Data from the voting intentions in the 1988 Chilean national plebiscite
data(Chile)
summary(Chile)

##   region      population      sex          age      education
##   C :600      Min.   : 3750   F:1379      Min.   :18.00   P   :1107
##   M :100      1st Qu.: 25000   M:1321      1st Qu.:26.00   PS  : 462
##   N :322      Median :175000                    Median :36.00   S   :1120
##   S :718      Mean   :152222                    Mean   :38.55   NA's:  11
##  SA:960      3rd Qu.:250000                    3rd Qu.:49.00
##                  Max.   :250000                    Max.   :70.00
##                               NA's   :1

##      income      statusquo      vote
##      Min.   : 2500   Min.   :-1.80301   A   :187
##      1st Qu.: 7500   1st Qu.:-1.00223   N   :889
##      Median :15000   Median :-0.04558   U   :588
##      Mean   :33876   Mean   : 0.00000   Y   :868
##      3rd Qu.:35000   3rd Qu.: 0.96857   NA's:168
##      Max.   :200000  Max.   : 2.04859
##      NA's   :98      NA's   :17

# vote is a factor with levels A (abstention), N (against Pinochet),
# U (undecided), Y (for Pinochet)

# Fit of the model done by multinom: Response ~ Predictors
# It is an iterative procedure (maxit sets the maximum number of iterations)
# Read the documentation in ?multinom for more information
mod1 <- multinom(vote ~ age + education + statusquo, data = Chile,
                  maxit = 1e3)
## # weights: 24 (15 variable)
## initial value 3476.826258
## iter 10 value 2310.201176
## iter 20 value 2135.385060
## final value 2132.416452
## converged
```

```

# Each row of coefficients gives the coefficients of the logistic
# regression of a level versus the reference level (A)
summary(mod1)
## Call:
## multinom(formula = vote ~ age + education + statusquo, data = Chile,
##           maxit = 1000)
##
## Coefficients:
## (Intercept)      age educationPS educationS statusquo
## N  0.3002851 0.004829029  0.4101765 -0.1526621 -1.7583872
## U  0.8722750 0.020030032 -1.0293079 -0.6743729  0.3261418
## Y  0.5093217 0.016697208 -0.4419826 -0.6909373  1.8752190
##
## Std. Errors:
## (Intercept)      age educationPS educationS statusquo
## N  0.3315229 0.006742834  0.2659012  0.2098064  0.1292517
## U  0.3183088 0.006630914  0.2822363  0.2035971  0.1059440
## Y  0.3333254 0.006915012  0.2836015  0.2131728  0.1197440
##
## Residual Deviance: 4264.833
## AIC: 4294.833

# Set a different level as the reference (N) for easening interpretations
Chile$vote <- relevel(Chile$vote, ref = "N")
mod2 <- multinom(vote ~ age + education + statusquo, data = Chile,
                  maxit = 1e3)
## # weights: 24 (15 variable)
## initial value 3476.826258
## iter 10 value 2393.713801
## iter 20 value 2134.438912
## final value 2132.416452
## converged
summary(mod2)
## Call:
## multinom(formula = vote ~ age + education + statusquo, data = Chile,
##           maxit = 1000)
##
## Coefficients:
## (Intercept)      age educationPS educationS statusquo
## A -0.3002035 -0.00482911 -0.4101274  0.1525608  1.758307
## U  0.5720544  0.01519931 -1.4394862 -0.5217093  2.084491
## Y  0.2091397  0.01186576 -0.8521205 -0.5382716  3.633550
##
## Std. Errors:
## (Intercept)      age educationPS educationS statusquo
## A  0.3315153 0.006742654  0.2658887  0.2098012  0.1292494
## U  0.2448452 0.004819103  0.2116375  0.1505854  0.1091445
## Y  0.2850655 0.005700894  0.2370881  0.1789293  0.1316567
##
## Residual Deviance: 4264.833
## AIC: 4294.833
exp(coef(mod2))
## (Intercept)      age educationPS educationS statusquo

```

```

## A 0.7406675 0.9951825 0.6635657 1.1648133 5.802607
## U 1.7719034 1.0153154 0.2370495 0.5935052 8.040502
## Y 1.2326171 1.0119364 0.4265095 0.5837564 37.846937

# Some highlights:
# - intercepts do not have too much interpretation (correspond to age = 0).
# A possible solution is to center age by its mean (so age = 0 would
# represent the mean of the ages)
# - both age and statusquo increase the probability of voting Y, A or U
# with respect to voting N -> conservativeness increases with ages
# - both age and statusquo increase more the probability of voting Y and U
# than A -> elderly and status quo supporters are more decided to participate
# - a PS level of education increases the probability of voting N. Same for
# a S level of education, but more prone to A

# Prediction of votes - three profile of voters
newdata <- data.frame(age = c(23, 40, 50),
                      education = c("PS", "S", "P"),
                      statusquo = c(-1, 0, 2))

# Probabilities of belonging to each class
predict(mod2, newdata = newdata, type = "probs")
##           N          A          U          Y
## 1 0.856057623 0.064885869 0.06343390 0.01562261
## 2 0.208361489 0.148185871 0.40245842 0.24099422
## 3 0.000288924 0.005659661 0.07076828 0.92328313

# Predicted class
predict(mod2, newdata = newdata, type = "class")
## [1] N U Y
## Levels: N A U Y

```



Multinomial logistic regression will suffer from numerical instabilities and its iterative algorithm might even fail to converge if the levels of the categorical variable are very separated (e.g., two data clouds clearly separated corresponding to a different level of the categorical variable).

## Appendix H

# Course evaluation

Evaluation is done by means of **two group projects**, to be done in groups of 3 (preferable) or 2 students, plus a presentation of one randomly-selected report. The final grade (in the scale 0-10) is:

```
min(0.70 * reports_grade + 0.30 * individual_presentation_grade, 10)
```

The `individual_presentation_grade` is in the scale 0-10. The `reports_grade` (in the scale 0-12) is the weighted grade of the two reports about the following topics:

1. Linear models I, II, and III (weight: 60%; **deadline: 2017/12/31**).
2. Generalized linear models **or**<sup>1</sup> Nonparametric regression (weight: 40%; **deadline: 2018/01/25**).

Deadlines are subjected to minor changes due to the course development.

### Groups

It is up to you to form the groups based on your grade expectations, affinity, complementary skills, etc. Take into account that all the students in a group will be graded evenly for the `reports_grade`.

Communicate the group compositions by filling this Google Sheet (you need to log in with your UC3M account). An example: if students A, B, and C form the group number 4 in the second project, add a 4 to the the rows of A, B, and C in the `Project II` column. Keep the same numbers in both columns if the composition of the group does not change. Under extraordinary circumstances to be communicated in advance to the professor, it is possible to switch group members from project to project. Be advised that this might have undesirable consequences if you do so: you might have to present two different randomly-chosen group reports.

### Report structure

The report must analyze a real dataset of your choice using the statistical methodology that we have seen in the lessons and labs. The purpose is to demonstrate that you understand and know how to apply and interpret the studied statistical techniques in a real-case scenario. Simulation studies (*i.e.*, analysis of artificially generated data) are also allowed if they are meant to illustrate interesting features or insights of the methods. The data analyzed does not have to be necessarily ‘Big Data’ and you may use the same dataset for different reports (if that makes sense).

Compulsory format guidelines:

- Structure: title, authors, abstract, first section, second section and so on. Do not use a cover.

---

<sup>1</sup>This is a `l`, not an `xor`.

- Font size: 11 points.
- Font type: any sensible choice (**Comic Sans** is not).
- Margins: any sensible choice. **Rmd** defaults works quite well.
- Spacing: single space, single column.
- Length: **10 pages at most**. This includes tables, graphics, code, equations, appendices, etc. Everything must fit in 10 pages. Want extra space? You can buy it at -1.00 points per extra page! ;)
- Style: be concise, specific, and insightful. Make a clever use of the space.
- Code: do not include *all* the code in the report. Instead of, just add the parts you want to highlight (or that you believe are particularly interesting) and describe what the code is doing, but provide it for reproducibility. A great way of doing this is to create an **R Markdown** document and omit the lengthy code chunks, providing the **.pdf** and **.Rmd** outputs.

Mandatory report structure:

1. **Abstract.** Provide a concise summary of the project. It must not exceed 250 words.
2. **Introduction.** State what is the problem to be studied. Provide some context, the question(s) that you want to address, a motivation of its importance, references, etc. Must not be *very* extensive.
3. **Methods.** Describe in detail the methods you are going to use, showing that you know what you are writing about. For example, you can include: background, useful interpretations, connections with other methods, alternative expositions, discussions, remarks, etc. If you wish, you can describe generally the method and then focus in exploring in more detail one particular feature that you use for the analysis. Optionally, go beyond what was taught in the lessons by considering extensions, improvements or better (in certain sense) implementations.
4. **Statistical analysis.** Apply the statistical methods to the real data. Justify their adequacy to the data studied. Obtain analyses, in the form of summaries and plots. Provide a discussion about the outputs, interpret them and obtain insights about the data. These last points tend to be skipped and are very important!
5. **Conclusions.** Summary of what was addressed in the project and of the most important conclusions. Takeaway messages. The conclusions are not required to be spectacular, but *fair and honest* in terms of what you have discovered.
6. **References.** Refer to the sources of information that you have employed (for the data, for information on the data, for the methodology, for the statistical analyses, etc).

## Report grading

The report grading will be performed according to the following breakdown:

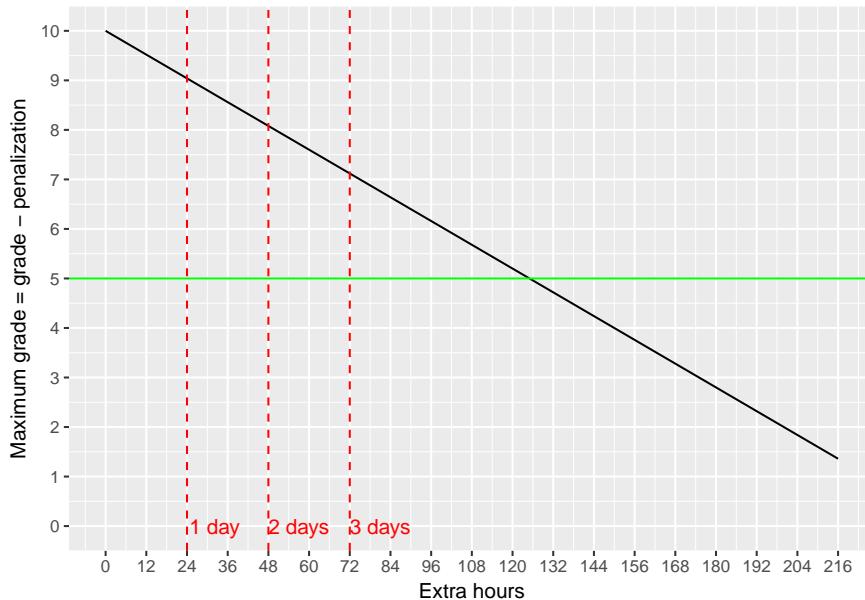
- **Method description** (4.5 points). In general, graded according to the evidence you show about the knowledge of the methods. For example, method descriptions in your own words that are accurate, detailed, and present your own insights are positively received.
- **Statistical analysis** (3.5 points). Graded according to the adequacy of the method to the data, the quantity and quality of the analysis provided, and, very importantly, the interpretation and the insights obtained from the statistical analyses.
- **Presentation** of the report (2 points). This involves the readability, the conciseness, the correct usage of English, and the overall presentation quality (**R Markdown** is preferred!). Code documentation (comment what the code is doing!), reproducibility, and clarity are also evaluated.
- **Excellence** (2 bonus points). Awarded for creativity of the analysis, originality of the problem studied and/or of the data acquisition process, use of advanced statistical tools, use of points briefly covered in lessons/labs, use of extensions of the methods, description of advanced insights of the methods, completeness of the report, use of advanced presentation tools, etc. Only awarded if the sum of regular points is above 7.5.

## Report hand in

Upload **one** report per group, in the form of a **.pdf**, to the Turnitin task associated to that report in AulaGlobal. Upload only a **.pdf**, not other file, so this can be checked for originality. **Also**, send **one** email per group to edgarcia@est-econ.uc3m.es with the subject “PM - Report X - Group Y” (replace X and Y as convenience) and write the group members in the body of the email. Attach the report (in **.pdf**) and all relevant scripts (e.g., **.Rmd** and **.R**) in a single **.zip** file. Be sure to ensure easy reproducibility of analysis (e.g., include the data you employed or the pertinent **.RData**).

**Deadlines:** the deadlines finish at **23:59** and the list of days can be seen above. Recall that the last deadline, the 25th of January at 23:59, is quite close to the presentation day, the 30th of January.

Need extra time? Buy it at expenses of a penalization in your grade:  $-0.04$  points per extra hour! ;) But only until the **25th of January**, reports received after that date will not be graded. The maximum grade according to the hours past the deadline is given in the graph below.



## Project presentations

The **30th of January, from 09:00 to 21:00 in room 0.B.08**, each group must present one of the four projects. The project to be presented is randomly selected. On the 25th of January, a list with the selected reports and presentation schedule will be made public, so you know at which time you have to present. Restrictions, if any, must be communicated and justified in advance. The presentation will determine the **individual\_presentation\_grade**.

The evaluation will be carried out as follows:

- Each group must present one report. Strict presentation policy: **all students must present for 5 minutes**. This is a 15-minutes or 10-minutes limit for the whole presentation, depending on the group size. Tailor your presentation to that time limit and break it evenly among the group members.
- All group members must be present in the presentation. Failure to be at the presentation results in **individual\_presentation\_grade < 0**.
- The **presentation must follow tightly the report contents**. You just need to explain properly what you did in the group project without adding new material. Take this into account when preparing the reports.
- In the presentation, you may want to highlight your main (personal or group) contributions and show your understanding of the methods.

- **Questions may be asked during and after the presentation of each student.** Questions may be about the theory, implementation, or other matters related with the report. You are assumed to know and be able to defend all these topics. Together with the presentation, the accurateness, conciseness, and detail in the answers will determine the **grade**.

### **Academic fraud**

Evidences of academic fraud will have serious consequences, such as a zero grade for the whole group and the reporting of the fraud detection to the pertinent academic authorities. Academic fraud includes (but is not limited to) plagiarism, use of sources without proper credit, project outsourcing, and the use of external tutoring not mentioned explicitly.

# Bibliography

- Allaire, J. J., Cheng, J., Xie, Y., McPherson, J., Chang, W., Allen, J., Wickham, H., Atkins, A., Hyndman, R., and Arslan, R. (2017). *rmarkdown: Dynamic Documents for R*. R package version 1.6.
- Ashenfelter, O., Ashmore, D., and Lalonde, R. (1995). Bordeaux wine vintage quality and the weather. *CHANCE*, 8(4):7–14.
- Dalal, S. R., Fowlkes, E. B., and Hoadley, B. (1989). Risk analysis of the space shuttle: Pre-challenger prediction of failure. *J. Am. Stat. Assoc.*, 84(408):945–957.
- Durbán, M. (2017). *Modelización Estadística*. Lecture notes.
- Fan, J. and Gijbels, I. (1996). *Local Polynomial Modelling and its Applications*, volume 66 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London.
- Furnival, G. M. and Wilson, R. W. (1974). Regressions by leaps and bounds. *Technometrics*, 16(4):499–511.
- Harrison, D. and Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air. *J. Environ. Econ. Manag.*, 5(1):81–102.
- James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 103 of *Springer Texts in Statistics*. Springer, New York. With applications in R.
- Kuhn, M. and Johnson, K. (2013). *Applied predictive modeling*. Springer, New York.
- Loader, C. (1999). *Local regression and likelihood*. Statistics and Computing. Springer-Verlag, New York.
- McCullagh, P. and Nelder, J. A. (1983). *Generalized linear models*. Monographs on Statistics and Applied Probability. Chapman & Hall, London.
- Miller, A. (1992). Algorithm AS 274: least squares routines to supplement those of Gentleman. *J. Roy. Statist. Soc. Ser. C*, 41(2):458–478.
- Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117.
- Nadaraya, E. (1964). On estimating regression. *Teor. Veroyatnost. i Primenen*, 9(1):157–159.
- Parzen, E. (1962). On estimation of a probability density function and mode. *Ann. Math. Statist.*, 33(3):1065–1076.
- Peña, D. (2002). *Ánálisis de datos multivariantes*. McGraw-Hill, Madrid.
- 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.
- R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27(3):832–837.

- van Buuren, S. and Groothuis-Oudshoorn, K. (2011). *mice: multivariate imputation by chained equations in R*. *Journal of Statistical Software*, 45(3):1–67.
- Wand, M. P. and Jones, M. C. (1995). *Kernel Smoothing*, volume 60 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, Ltd., London.
- Wasserman, L. (2004). *All of Statistics*. Springer Texts in Statistics. Springer-Verlag, New York. A concise course in statistical inference.
- Wasserman, L. (2006). *All of Nonparametric Statistics*. Springer Texts in Statistics. Springer, New York.
- Watson, G. S. (1964). Smooth regression analysis. *Sankhyā Ser. A*, 26:359–372.
- Wood, S. N. (2006). *Generalized Additive Models: An Introduction with R*. Texts in Statistical Science Series. Chapman & Hall/CRC, Boca Raton.
- Xie, Y. (2016a). *bookdown: Authoring Books and Technical Documents with R Markdown*. The R Series. Chapman & Hall/CRC, Boca Raton.
- Xie, Y. (2016b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.14.