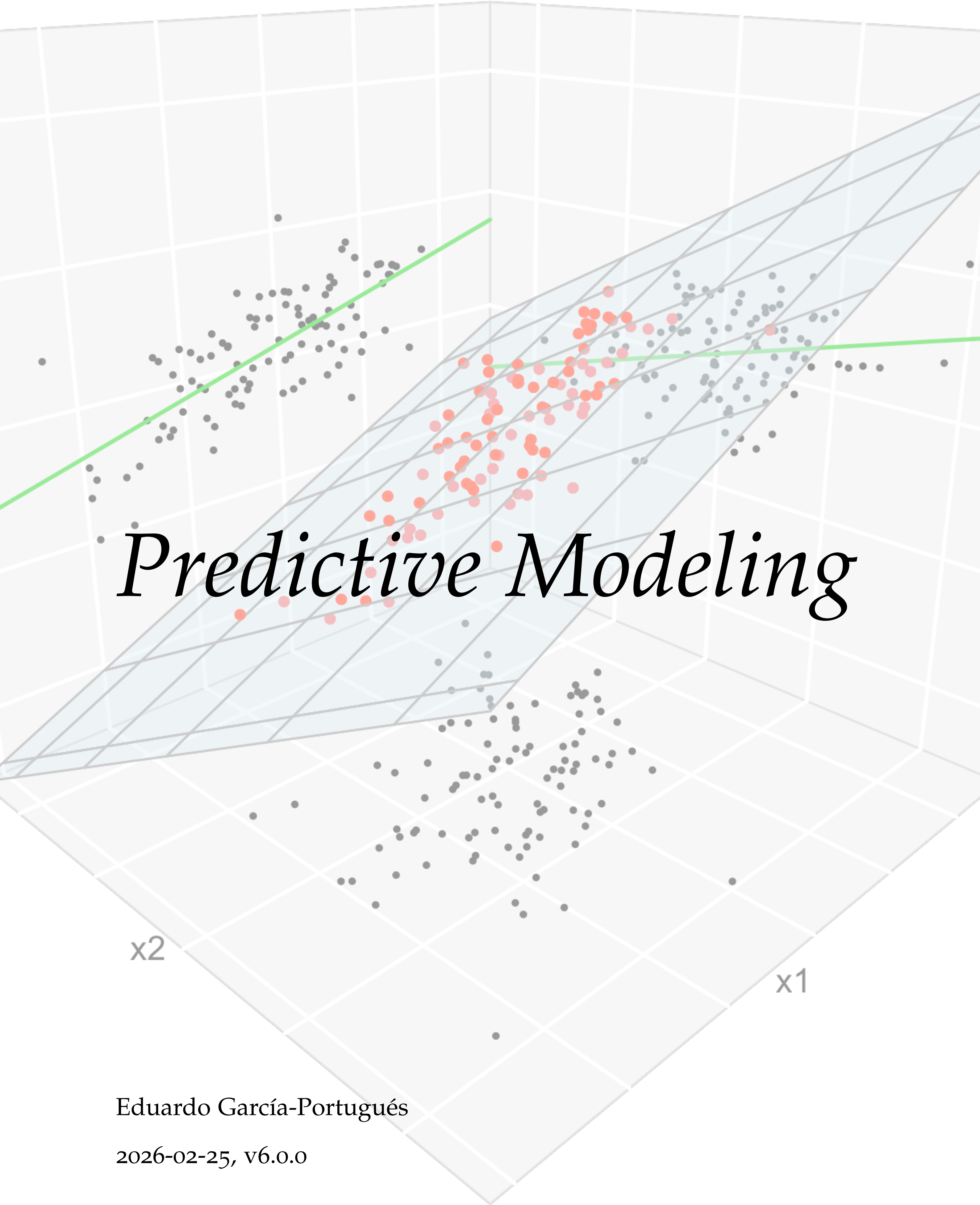
# *Predictive Modeling*



Eduardo García-Portugués

2026-02-25, v6.0.0

ISBN 978-84-09-29679-8

AVAILABLE AT

EGARPOR.GITHUB.IO/PM-UC3M

*February 2026*

# *Contents*

# *Welcome*

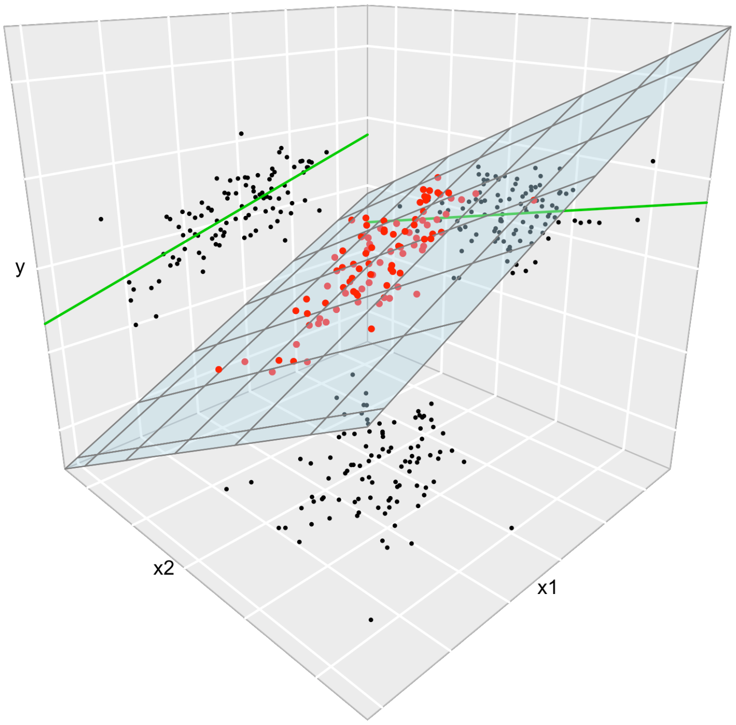


Figure 1: Illustration of how the regression plane (blue) of $Y$ on $X_1$ and $X_2$ does not extend the simple linear regressions (green lines) of $Y$ on $X_1$ and of $Y$ on $X_2$.

Welcome to *Predictive Modeling*! This book covers the statistical foundations and computational practice of regression-based predictive modeling. The unifying thread is the regression model $Y = m(X_1, \ldots, X_p) + \varepsilon$, which relates a response $Y$ to predictors $X_1, \ldots, X_p$. How do we estimate $m$? How to quantify the uncertainty in its estimation? How to evaluate predictive performance in practice?

The book builds from linear models into their modern extensions, adaptations, and generalizations. Another title could have been "linear models and friends", understanding "friends" generously as methods that extend linear models or use them as building blocks. This progression moves from the rigid structure of vanilla linear models to the flexibility of nonparametric smoothing, passing through high-dimensional, big data, and non-normal settings.

The book strikes a balance between statistical theory and hands-on modeling practice. Each chapter develops the key ideas alongside fully reproducible R code on simulated and real data. Three main case studies anchor the core chapters: predicting Bordeaux wine quality from meteorological variables, modeling Boston housing values from structural and neighborhood characteristics, and assessing O-ring failure probability in the Challenger disaster. The exposition is complemented by circa 20 Shiny applications, allowing the reader to visualize key concepts and develop intuition for how methods behave in practice.

A collection of more than 60 exercises is included to help the reader test their understanding and apply the methods. These exercises vary in difficulty and type, mainly focusing on data analysis tasks and programming exercises. The reader should close each chapter with a clear understanding of *why* and *how* the methods work, together with the practical ability to deploy them.

## *Scope of the book*

The central focus of the book is **regression modeling for prediction** through the regression model $Y = m(X_1, \ldots, X_p) + \varepsilon$. The main arc runs from *linear models* and their practical and modern extensions (model selection, dimension reduction, shrinkage, and big data), to *generalized linear models* for different response types, and finally to *nonparametric regression* via local polynomial fitting and local likelihood. Recurring themes are bias–variance tradeoffs, regularization and tuning, model assessment, and the practical consequences of modeling assumptions.

The book does not intend to be comprehensive on the predictive modeling landscape. Important methods, such as tree-based models, neural networks, and support vector machines, lie beyond the scope of this text. Nevertheless, the concepts emphasized throughout, such as regularization through shrinkage, model selection, and diagnostic checks, provide a solid foundation for engaging with the wider predictive modeling landscape.

The book begins with Chapter 1, Introduction, which frames predictive modeling through the regression model $Y = m(X_1, \ldots, X_p) + \varepsilon$ and clarifies what it means to learn a regression function from data. It introduces the key tradeoffs that recur throughout, such as prediction accuracy versus interpretability and bias versus variance, and illustrates overfitting. The chapter also establishes the notation and basic probabilistic background used in the rest of the book.

Chapter 2, Linear models I, develops the multiple linear model $Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$ as the foundational parametric approach to regression. Beginning with least squares and its geometric interpretation, the chapter treats model assumptions and their necessity, sampling distributions and inference for coefficients, confidence and prediction intervals, ANOVA decomposition, and goodness-of-fit measures. The prediction of Bordeaux wine quality from weather data provides a running case study through the chapter.

Chapter 3, Linear models II, focuses on linear modeling in more realistic scenarios, where many predictors are available, hence many competing model specifications are possible, transformations of predictors are often needed, and assumptions need to be checked. It covers model selection through information criteria and cross-validation, notes on model selection consistency, use of qualitative predictors and interactions, nonlinear extensions including polynomials, and the usual toolkit for model diagnostics. The chapter closes with dimension reduction techniques within linear models: principal components regression, including a review of principal component analysis, and partial least squares. The pricing of housing in Boston is used as the recurrent case study.

Chapter 4, Linear models III, extends linear regression to non-classical settings in which ordinary least squares is infeasible, unstable, or impractical. Ridge regression and the lasso are introduced as shrinkage methods for high-dimensional problems in which $p \gg n$, with particular attention to tuning-parameter selection, analytical form of ridge regression, and variable selection through lasso. The chapter also treats linear models where the parameter vector $\boldsymbol{\beta}$ is subject to linear constraints. A different extension is the multivariate linear model where the response is a vector $\mathbf{Y} = (Y_1, \ldots, Y_q)$: estimation, inference, and shrinkage through ridge and lasso are developed. Mathematical and computational considerations for big data close the chapter.

Chapter 5, Generalized linear models, broadens the regression framework to non-normal responses through the exponential family and link functions. Logistic regression for binary data is developed in depth: estimation by maximum likelihood, interpretation via odds ratios, inference, deviance, model selection, diagnostics, shrinkage through ridge and lasso, and big data considerations. Formulation and estimation are extended to generic generalized linear models, with specifics provided for Poisson regression and binomial regression for count data. The Challenger disaster provides the central case study for the chapter.

Finally, Chapter 6, Nonparametric regression, moves beyond parametric structure by estimating $m$ with minimal assumptions, letting the data determine the shape of the regression function. The chapter develops the core tools of smoothing starting from nonparametric density estimation (histograms and kernel density estimators) and then moving to kernel regression, including the Nadaraya–Watson estimator and local polynomial regression, with careful attention to bandwidth selection and its bias–variance implications. Extensions to mixed multivariate predictors, prediction and confidence intervals, and local likelihood methods for binary responses complete the chapter.

## *Software & code*

The book contains a substantial amount of code, with snippets that are intended to be self-contained within the chapter in which they appear. This helps illustrate how the methods and theory translate to practice. The **software** employed throughout this book is *the* statistical language R and the RStudio IDE (Integrated Development Environment). Basic knowledge of both is assumed.[1]

The **Shiny interactive apps** in the book can be downloaded and run locally, which also allows inspection of their source code. Check out this GitHub repository for the sources.

Several packages that are not included within R by default are used throughout the book. All of the packages can be installed with the following commands:

```
# Installation of required packages
packages <- c("MASS", "car", "readxl", "rgl", "rmarkdown", "nortest",
              "latex2exp", "pca3d", "ISLR", "pls", "corrplot", "glmnet",
              "mvtnorm", "biglm", "leaps", "lme4", "viridis", "ffbase",
              "ks", "KernSmooth", "nor1mix", "np", "locfit",
              "manipulate", "mice", "VIM", "nnet")
install.packages(packages)
```

The book makes explicit mention of the package to which a function belongs by using the operator `::`, except when the use of the functions of a package is very repetitive and that package is loaded. You can load all the packages by running:

```
# Load packages
lapply(packages, library, character.only = TRUE)
```

The book's R code snippets are collected in the following scripts:

- Chapter 1: `01-intro.R`.
- Chapter 2: `02-lm-i.R`.
- Chapter 3: `03-lm-ii.R`.
- Chapter 4: `04-lm-iii.R`.
- Chapter 5: `05-glm.R`.
- Chapter 6: `06-npreg.R`.
- Appendix: `07-appendix.R`.

To download them, use your browser's **Save link as...** option.

## *Datasets*

The following is a handy list of all the relevant datasets used in the course together with brief descriptions. The list is sorted according to the order of appearance of the datasets in the notes.

- `wine.csv`. 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`. Contains a single `data.frame`, named `leastSquares`, with 50 observations of the variables `x`, `yLin`, `yQua`, and `yExp`. These are generated as $X \sim \mathcal{N}(0,1)$, $Y_{\mathrm{lin}} = -0.5 + 1.5X + \varepsilon$, $Y_{\mathrm{qua}} = -0.5 + 1.5X^2 + \varepsilon$, and $Y_{\mathrm{exp}} = -0.5 + 1.5 \cdot 2^X + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, 0.5^2)$. The purpose of the dataset is to illustrate the least squares fitting.

[1] 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.

- least-squares-3D.RData. Contains a single data.frame, named leastSquares3D, with 50 observations of the variables x1, x2, x3, yLin, yQua, and yExp. These are generated as $X_1, X_2 \sim \mathcal{N}(0,1)$, $X_3 = X_1 + \mathcal{N}(0, 0.05^2)$, $Y_{\mathrm{lin}} = -0.5 + 0.5X_1 + 0.5X_2 + \varepsilon$, $Y_{\mathrm{qua}} = -0.5 + X_1^2 + 0.5X_2 + \varepsilon$, and $Y_{\mathrm{exp}} = -0.5 + 0.5e^{X_2} + X_3 + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0,1)$. The purpose of the dataset is to illustrate the least squares fitting with several predictors.

- assumptions.RData. 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 moreAssumptions.RData dataset has the same structure.

- assumptions3D.RData. 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. 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 ?MASS::Boston.

- cpus.txt and gpus.txt. 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. Contains 19 performance metrics for the 20 football teams in La Liga 2015/2016.

- challenger.txt. 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. Contains data for 90 country parcels in which the Biomass, pH of the terrain (categorical variable), and number of Species were measured.

- heart.txt. Contains data for 226 patients suspected of having a future heart attack. The variables are CK (level of creatinine kinase), and ha and ok (number of patients that suffered a heart attack and did not suffer it, respectively).

- Chile.txt. 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). Retrieved from `data(Chile, package = "carData")`.

To download them, use your browser's **Save link as...** option.

## Citation

You may use the following BIBTEX entry when citing this book:

```
@book{PredictiveModeling2026,
    title       = {Predictive Modeling},
    author      = {Garc\'ia-Portugu\'es, E.},
    year        = {2026},
    note        = {Version 6.0.0. ISBN 978-84-09-29679-8},
    url         = {https://egarpor.github.io/PM-UC3M/}
}
```

You may also want to use the following template:

> García-Portugués, E. (2026). *Predictive Modeling*. Version 6.0.0. ISBN 978-84-09-29679-8. Available at `https://egarpor.github.io/PM-UC3M/`.

A previous version of the book at the now-discontinued bookdown.org hosting service was known as *Notes for Predictive Modeling*.

## License

## *Contributions*

Contributions, reporting of typos, and feedback on the book are very welcome. Send an email to `edgarcia@est-econ.uc3m.es` and I will gladly add your name to the list of contributors.

## *Credits*

Several excellent references were used in preparing this book. The following list presents the books that have been consulted:

- Chacón and Duong (2018) (Section 6.1.4)
- DasGupta (2008) (Section 3.5.2)
- Durbán (2017) (Section 5.2.2)
- Fan and Gijbels (1996) (Sections 6.2, 6.2.3, and 6.2.4)
- Hastie et al. (2009) (Section 4.1)
- James et al. (2013) (Sections 2.2 – 2.7, 3.1, 3.5, and 3.6.3, 4.1)
- Kuhn and Johnson (2013) (Section 1.1)
- Li and Racine (2007) (Section 6.3)
- Loader (1999) (Section 6.5)
- McCullagh and Nelder (1983) (Sections 5.2 – 5.6)
- Peña (2002) (Sections 2.2 – 2.7, 3.5, and 5.2.1)
- Seber and Lee (2003) (Section 4.2)
- Seber (1984) (Section 4.3)
- Wand and Jones (1995) (Sections 6.1.2, 6.1.3, and 6.2.4)
- Wasserman (2004) (Sections 6.5)
- Wasserman (2006) (Sections 6.2.4)
- Wood (2006) (Sections 5.2.2 and 5.7)

This book was made possible thanks to the excellent open-source software Xie (2016), Xie (2020), Allaire et al. (2020), Xie and Allaire (2020), and R Core Team (2020). In addition, some layout improvements build on the outstanding work of Úcar (2018). The icons used in the book were designed by madebyoliver, freepik, and roundicons from Flaticon.

Last but not least, the book has benefited from contributions from the following people (in alphabetical order):

- Apezteguía García, Ainara (fixed a typo)
- Botz, Katherine (performed a thorough proofreading of the course materials, fixing a large number of typos)
- Caballero Cárdenas, Jorge (fixed five typos)
- Carrera Maestro, Antonio (fixed two bugs and three typos)
- Castillo Estévez, Marcos José (fixed two typos)
- Cerdán Pedraza, Luis (performed an outstanding proofreading of the course materials fixing more than fifty typos and style issues)
- Chettouh, Frederik (fixed a typo and two bugs)
- Demir, Gulnur (fixed two typos)
- Escalante Ariza, Andrés (fixed a typo)

- Fernández, José Ángel (fixed several typos)
- García Ramírez, Celia (fixed two bugs)
- González Berzal, Trinidad (fixed a typo)
- González González, David (fixed two typos)
- Marín Abril, Antonio (fixed two bugs)
- Modet Álamo, Andrés (performed an excellent review of the course materials detecting and fixing more than thirty typos and four bugs)
- Palmero Muñoz, Santiago (fixed a typo and a bug)
- Petraccaro, Federico (fixed three typos)
- Ramírez Díaz, Enrique (fixed a typo)
- Razgovorov, Pavel (fixed a typo)
- Rodríguez Beltrán, Cristina (fixed a typo and two bugs)
- Rodríguez Ramírez, Manuel (fixed two typos)
- Romero González, Celia (fixed a typo)
- Royo Ruiz, Carlota (fixed a bug)
- Stincone, Leonardo (fixed a typo and a bug)

# 1
# *Introduction*

## 1.1 *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 $X_1, \ldots, X_p$. The focus is on learning what is the probabilistic model that relates $Y$ with $X_1, \ldots, X_p$, and use that acquired knowledge for predicting $Y$ given an observation of $X_1, \ldots, X_p$. Some concrete examples of this are:

- Predicting the wine quality ($Y$) from a set of environmental variables ($X_1, \ldots, X_p$).
- Predicting the number of sales ($Y$) from a set of marketing investments ($X_1, \ldots, X_p$).
- Modeling the average house value in a given suburb ($Y$) from a set of community-related features ($X_1, \ldots, X_p$).
- Predicting the probability of failure ($Y$) of a rocket launcher from the ambient temperature ($X_1$).
- Predicting students academic performance ($Y$) according to education resources and learning methodologies ($X_1, \ldots, X_p$).
- Predicting the fuel consumption of a car ($Y$) from a set of driving variables ($X_1, \ldots, X_p$).

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

$$Y = m(X_1, \ldots, X_p) + \varepsilon, \tag{1.1}$$

where $m$ is the *regression function* and $\varepsilon$ is a random error with zero mean that accounts for the uncertainty of knowing $Y$ if $X_1, \ldots, X_p$ are known. The function $m : \mathbb{R}^p \to \mathbb{R}$ is unknown in practice and its estimation is the objective of predictive modeling: $m$ **encodes the relation**[1] **between** $Y$ **and** $X_1, \ldots, X_p$. In other words, $m$ captures the *trend* of the relation between $Y$ and $X_1, \ldots, X_p$, and $\varepsilon$ represents the *stochasticity* of that relation. Knowing $m$ allows to predict $Y$. This

[1] The relation is encoded in *average* by means of the *conditional expectation*.

[2] That can be regarded as "structures for $m$".

course is devoted to *statistical models*[2] that allow us to come up with an estimate of $m$, denoted by $\hat{m}$, that can be used to predict $Y$.

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 $m : \mathbb{R} \to \mathbb{R}$ ($p = 1$) with $m(x) = a + bx + cx^2$. Suppose the following data consists of 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[3]):

```r
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")
```

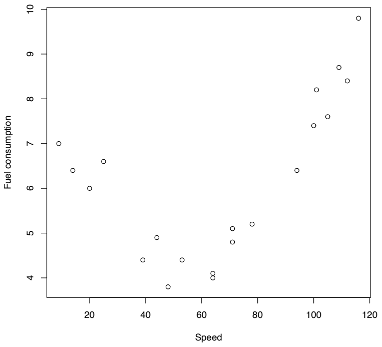


Figure 1.1: Scatterplot of fuel consumption vs. speed.

From this data, we can estimate $m$ by means of a polynomial model[4]:

```r
# 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
```

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 estimated fuel consumption at speed 90 km/h is `8.512421 - 0.153291 * 90 + 0.001408 * 90^2 = 6.1210`.

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

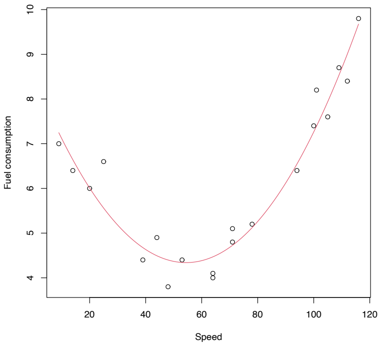


Figure 1.2: Fitted quadratic model.

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

1. The **prediction accuracy versus interpretability** trade-off. *Prediction accuracy* is key in any predictive model: of course, the better the model is able to predict $Y$, the more useful it will be. However, some models 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 exactly which variables are most influential in $Y$, to be able to interpret the parameters of the model, and to

[3] This is an alternative useful view of $\varepsilon$: the aggregation of the effects that we cannot account for predicting $Y$.

[4] 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.

translate the prediction process to non-experts. In essence, interpretability allows to **explain precisely** how and why the model behaves when predicting $Y$ from $X_1, \ldots, X_p$. Most of the models covered in this text clearly **favor interpretability**[5] and hence they may make a sacrifice in terms of their prediction accuracy when compared with more convoluted models.

2. **Model correctness versus model usefulness**. Correctness and usefulness are two different concepts in modeling. The first refers to the model being *statistically* correct, this is, it translates to stating that the **assumptions** on which the model relating $Y$ with $X_1, \ldots, X_p$ is built are satisfied. The second refers to the model being **useful for explaining or predicting** $Y$ from $X_1, \ldots, X_p$. Both concepts are certainly 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 may be completely spurious.[6]

3. **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 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, up to the point in which they end up not estimating the true relation between $Y$ and $X_1, \ldots, X_p$, $m$, but merely *interpolating* the observed data. This well-known phenomenon is called **overfitting** and it can be avoided by splitting the dataset in two datasets:[7] the *training* dataset, used for estimating the model; and the *testing* dataset, used for evaluating the fitted model predictive performance. On the other hand, excessive simplicity (**underfitting**) is also problematic, since the true relation between $Y$ and $X_1, \ldots, X_p$ may be overly simplified. Therefore, a trade-off in the degree of flexibility has to be attained for having a good model. This is often referred to as the **bias-variance trade-off** (low flexibility increases the bias of the fitted model, high flexibility increases the variance). An illustration of this transversal problem in predictive modeling is given in Figure 1.1.

Illustration of overfitting 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 (predic-

[5] Not only that but they are *neatly* interpretable.

[6] Particularly, it usually happens that the inference based on erroneous assumptions *underestimates* variability, as the assumptions tend to ensure that the information of the sample is maximal for estimating the model at hand. Thus, inference based on erroneous assumptions results in a false sense of confidence: a larger error is made in reality than the one the model theory states.

[7] Three datasets if we are fitting *hyperparameters* or *tuning parameters* in our model: the *training* dataset for estimating the model parameters; the *validation* dataset for estimating the hyper-parameters; and the *testing* dataset for evaluating the final performance of the fitted model.

tion), 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 available here.

## 1.2 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 modeling we are concerned about the prediction or explanation of a **response** $Y$ from a set of **predictors** $X_1,\ldots,X_p$. Both $Y$ and $X_1,\ldots,X_p$ are random variables, but we use them in a different way: our interest lies in predicting or explaining $Y$ *from* $X_1,\ldots,X_p$. Another name for $Y$ is *dependent variable* and $X_1,\ldots,X_p$ are sometimes referred to 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) $f(x)=F'(x)$,[8] 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 defined as[9]

$$\begin{aligned}\mathbb{E}[g(X)] &:= \int g(x)\,\mathrm{d}F(x)\\ &:= \begin{cases}\displaystyle\int g(x)f(x)\,\mathrm{d}x, & \text{if } X \text{ is continuous,}\\ \displaystyle\sum_{\{x\in\mathbb{R}:\mathbb{P}[X=x]>0\}} g(x)\mathbb{P}[X=x], & \text{if } X \text{ is discrete.}\end{cases}\end{aligned}$$

The sign ":=" emphasizes that the Left Hand Side (LHS) of the equality is defined for the first time as the Right Hand Side (RHS). Unless otherwise stated, the integration limits of any integral are $\mathbb{R}$ or $\mathbb{R}^p$. The *variance* is defined as $\mathbb{V}\mathrm{ar}[X]:=\mathbb{E}[(X-\mathbb{E}[X])^2]=\mathbb{E}[X^2]-\mathbb{E}[X]^2$.

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 $\mathbf{A}^\top$ to the transpose of $\mathbf{A}$. Boldfaced capitals will be used simultaneously for denoting matrices and also *random vectors* $\mathbf{X}=(X_1,\ldots,X_p)$, which are collections of random variables $X_1,\ldots,X_p$. The (joint) cdf of $\mathbf{X}$ is[10]

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

[8] Respectively, $F(x)=\int_{-\infty}^x f(t)\,\mathrm{d}t$.

[9] The precise mathematical meaning of "$\mathrm{d}F_X(x)$" is the Riemann–Stieltjes integral.

[10] Understood as the probability that $(X_1\leq x_1)$ *and* ... *and* $(X_p\leq x_p)$.

and, if $\mathbf{X}$ is continuous, its (joint) pdf is $f := \frac{\partial^p}{\partial x_1 \cdots \partial x_p} F$.

The *marginals* of $F$ and $f$ are the cdf and pdf of $X_j$, $j = 1, \ldots, p$, respectively. They are defined as:

$$\begin{aligned} F_{X_j}(x_j) &:= \mathbb{P}[X_j \leq x_j] = F(\infty, \ldots, \infty, x_j, \infty, \ldots, \infty), \\ f_{X_j}(x_j) &:= \frac{\partial}{\partial x_j} F_{X_j}(x_j) = \int_{\mathbb{R}^{p-1}} f(\mathbf{x}) \,\mathrm{d}\mathbf{x}_{-j}, \end{aligned}$$

where $\mathbf{x}_{-j} := (x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_p)$. 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 $X_1 \mid (X_2, \ldots, X_p)$ are defined, respectively, as

$$\begin{aligned} F_{X_1 \mid \mathbf{X}_{-1} = \mathbf{x}_{-1}}(x_1) &:= \mathbb{P}[X_1 \leq x_1 \mid \mathbf{X}_{-1} = \mathbf{x}_{-1}], \\ f_{X_1 \mid \mathbf{X}_{-1} = \mathbf{x}_{-1}}(x_1) &:= \frac{f(\mathbf{x})}{f_{\mathbf{X}_{-1}}(\mathbf{x}_{-1})}. \end{aligned}$$

The *conditional expectation* of $Y \mid X$ is the following random variable[11]

$$\mathbb{E}[Y \mid X] := \int y \,\mathrm{d}F_{Y|X}(y \mid X).$$

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

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

and the *correlation* between them is defined as

$$\mathrm{Cor}[X_1, X_2] := \frac{\mathrm{Cov}[X_1, X_2]}{\sqrt{\mathbb{V}\mathrm{ar}[X_1]\mathbb{V}\mathrm{ar}[X_2]}}.$$

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

$$\begin{aligned} \mathbb{V}\mathrm{ar}[\mathbf{X}] :=& \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top] \\ =& \mathbb{E}[\mathbf{X}\mathbf{X}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^\top \\ =& \begin{pmatrix} \mathbb{V}\mathrm{ar}[X_1] & \mathrm{Cov}[X_1, X_2] & \cdots & \mathrm{Cov}[X_1, X_p] \\ \mathrm{Cov}[X_2, X_1] & \mathbb{V}\mathrm{ar}[X_2] & \cdots & \mathrm{Cov}[X_2, X_p] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}[X_p, X_1] & \mathrm{Cov}[X_p, X_2] & \cdots & \mathbb{V}\mathrm{ar}[X_p] \end{pmatrix}, \end{aligned}$$

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

$$\mathbb{E}[\mathbf{A}\mathbf{X} + \mathbf{b}] = \mathbf{A}\mathbb{E}[\mathbf{X}] + \mathbf{b}, \quad \text{for a } q \times p \text{ matrix } \mathbf{A} \text{ and } \mathbf{b} \in \mathbb{R}^q. \tag{1.2}$$

It follows from (1.2) that

$$\mathbb{V}\mathrm{ar}[\mathbf{A}\mathbf{X} + \mathbf{b}] = \mathbf{A}\mathbb{V}\mathrm{ar}[\mathbf{X}]\mathbf{A}^\top, \quad \text{for a } q \times p \text{ matrix } \mathbf{A} \text{ and } \mathbf{b} \in \mathbb{R}^q. \tag{1.3}$$

[11] Recall that the $X$-part of $\mathbb{E}[Y \mid X]$ is *random*. However, $\mathbb{E}[Y \mid X = x]$ is *deterministic*.

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 the usual *normal* distribution. Its (joint) pdf is given by

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

The $p$-dimensional normal has a nice linear property that stems from (1.2) and (1.3):

$$\mathbf{A}\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) + \mathbf{b} \overset{d}{=} \mathcal{N}_q(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top). \tag{1.4}$$

Notice that when $p = 1$, and $\boldsymbol{\mu} = \mu$ and $\boldsymbol{\Sigma} = \sigma^2$, then the pdf of the usual normal $\mathcal{N}(\mu, \sigma^2)$ is recovered:[12]

$$\phi(x; \mu, \sigma^2) := \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

When $p = 2$, the pdf is expressed in terms of $\boldsymbol{\mu} = (\mu_1, \mu_2)^\top$ 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 $-1 < \rho < 1$:

$$\begin{aligned} \phi(x_1, x_2; \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho) &:= \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \qquad (1.5) \\ \times \exp&\left\{ -\frac{1}{2(1-\rho^2)} \left[ \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \right] \right\}. \end{aligned}$$

The surface defined by (1.5) can be regarded as a 3-dimensional bell. In addition, it serves to provide concrete examples of the functions introduced above:

- Joint pdf:

  $$f(x_1, x_2) = \phi(x_1, x_2; \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho).$$

- Marginal pdfs:

  $$f_{X_1}(x_1) = \int \phi(x_1, t_2; \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho)\,\mathrm{d}t_2 = \phi(x_1; \mu_1, \sigma_1^2)$$

  and $f_{X_2}(x_2) = \phi(x_2; \mu_2, \sigma_2^2)$. Hence $X_1 \sim \mathcal{N}\left(\mu_1, \sigma_1^2\right)$ and $X_2 \sim \mathcal{N}\left(\mu_2, \sigma_2^2\right)$.

- Conditional pdfs:

  $$\begin{aligned} f_{X_1|X_2=x_2}(x_1) =& \frac{f(x_1, x_2)}{f_{X_2}(x_2)} = \phi\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2), (1-\rho^2)\sigma_1^2\right), \\ f_{X_2|X_1=x_1}(x_2) =& \phi\left(x_2; \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1), (1-\rho^2)\sigma_2^2\right). \end{aligned}$$

  Hence

  $$\begin{aligned} X_1 \mid X_2 = x_2 &\sim \mathcal{N}\left(\mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2), (1-\rho^2)\sigma_1^2\right), \\ X_2 \mid X_1 = x_1 &\sim \mathcal{N}\left(\mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1-\mu_1), (1-\rho^2)\sigma_2^2\right). \end{aligned}$$

[12] If $\mu = 0$ and $\sigma = 1$ (*standard normal*), then the pdf and cdf are simply denoted by $\phi$ and $\Phi$, without extra parameters.

- Conditional expectations:

$$\mathbb{E}[X_1 \mid X_2 = x_2] = \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2),$$
$$\mathbb{E}[X_2 \mid X_1 = x_1] = \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1).$$

- Joint cdf:

$$\int_{-\infty}^{x_2}\int_{-\infty}^{x_1} \phi(t_1, t_2; \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho)\,\mathrm{d}t_1\,\mathrm{d}t_2.$$

- Marginal cdfs: $\int_{-\infty}^{x_1} \phi(t; \mu_1, \sigma_1^2)\,\mathrm{d}t =: \Phi(x_1; \mu_1, \sigma_1^2)$ and analogously $\Phi(x_2; \mu_2, \sigma_2^2)$.

- Conditional cdfs:

$$\int_{-\infty}^{x_1} \phi\left(t; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1-\rho^2)\sigma_1^2\right)\,\mathrm{d}t = \Phi\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1-\rho^2)\sigma_1^2\right)$$

  and analogously $\Phi\left(x_2; \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1), (1-\rho^2)\sigma_2^2\right)$.

Figure 1.3 graphically summarizes the concepts of joint, marginal, and conditional distributions within the context of a 2-dimensional normal.

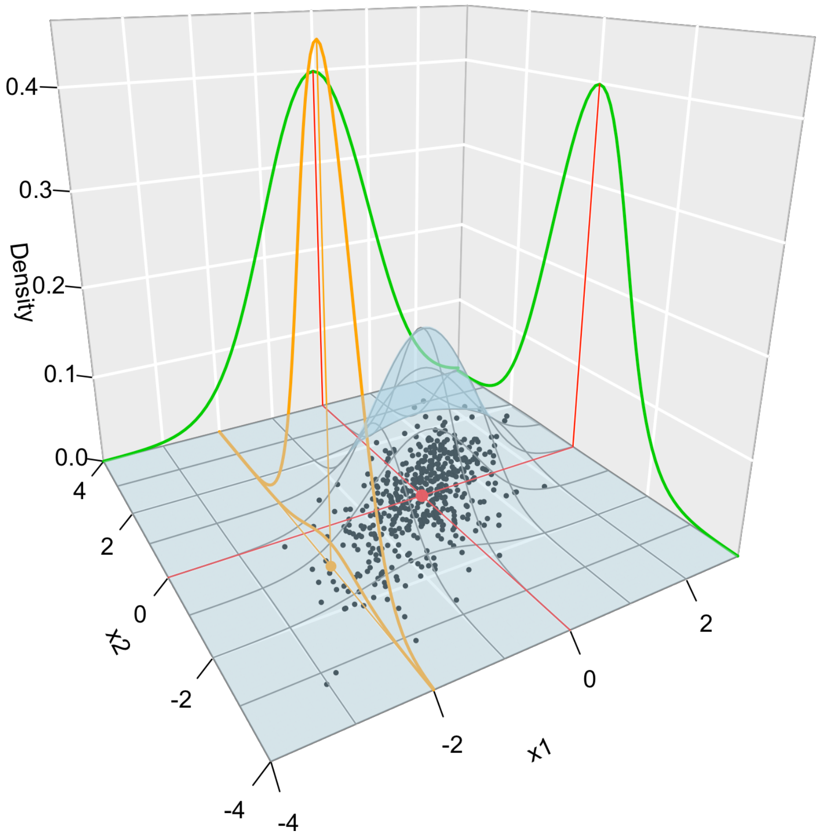


Figure 1.3: Visualization of the joint pdf (in blue), marginal pdfs (green), conditional pdf of $X_2 \mid X_1 = x_1$ (orange), expectation (red point), and conditional expectation $\mathbb{E}[X_2 \mid X_1 = x_1]$ (orange point) of a 2-dimensional normal. The conditioning point of $X_1$ is $x_1 = -2$. 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 $f(x_1, x_2)$ (lower orange curve) with $x_1 = -2$ 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.

Finally, 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 $X_j$. Thus we will deal with samples of the form $\{(X_{i1},\ldots,X_{ip},Y_i)\}_{i=1}^n$.

# 2
# *Linear models I: multiple linear model*

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

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

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 $(X_1, Y_1), \ldots, (X_n, Y_n)$ such that its scatterplot is the one in Figure 2.1.

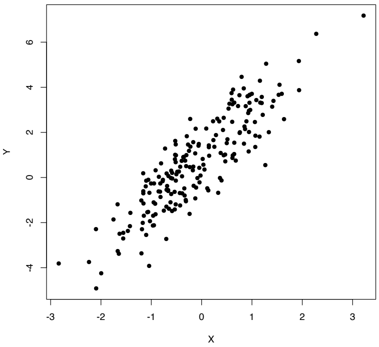


Figure 2.1: Scatterplot of a sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ showing a linear pattern.

## 2.1 *Case study:* The Bordeaux equation

> 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. You can see a short review of the story at the Financial Times[1] and at the video in Figure 2.2.

Figure 2.2: ABC interview to Orley Ashenfelter, broadcasted in 1992. Video also available here.

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 weather conditions. Because Bordeaux wines taste better when they are older,[2] 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 will change substantially when the aged wine is in the market. Therefore, being able to *predict the*

[1] "How computers routed the experts", Financial Times, 31/08/2007.

[2] Young wines are astringent, when the wines age they lose their astringency.

*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?*

The `wine.csv` file contains 27 red Bordeaux vintages. The data is the same data[3] originally employed by Ashenfelter et al. (1995), except for the inclusion of the variable `Year`, the exclusion of `NA`s and the reference price used for the wine. 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 a series of 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 in 1983 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. A portion of 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

[3] Source: `http://www.liquidasset.com/winedata.html`.

on the price. Before doing so, we need to introduce the required statistical machinery.

## 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 extend it to the multiple linear model.

### 2.2.1 *Simple linear model*

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

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{2.1}$$

holds between $X$ and $Y$. In (2.1), $\beta_0$ and $\beta_1$ are known as the *intercept* and *slope*, respectively. The random variable $\varepsilon$ has mean zero and is 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

$$\mathbb{E}[Y \mid X = x] = \beta_0 + \beta_1 x, \tag{2.2}$$

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

The 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 clear **interpretation of the coefficients**:

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

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

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

In other words, we look for the estimators $(\hat\beta_0, \hat\beta_1)$ such that

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

The motivation for minimizing the RSS is geometrical, as shown by Figure 2.2.1. We aim 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 available here.

It can be seen that *the minimizers of the RSS*[4] are

$$\hat\beta_0 = \bar Y - \hat\beta_1 \bar X, \quad \hat\beta_1 = \frac{s_{xy}}{s_x^2}, \tag{2.3}$$

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.[5]
- $s_{xy} = \frac{1}{n}\sum_{i=1}^n (X_i - \bar X)(Y_i - \bar Y)$ is the sample covariance. It measures the degree of linear association between $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$. 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 to maximum likelihood estimation under the normal distribution (see Appendix B).

Let's see how to obtain automatically the minimizers of the error in Figure 2.2.1 by the `lm` (**l**inear **m**odel) 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.

```
# Call lm
lm(yLin ~ x, data = leastSquares)
##
## Call:
## lm(formula = yLin ~ x, data = leastSquares)
##
```

[4] They are unique and always exist (except if $s_x = 0$, when all the data points are the same). They can be obtained by solving $\frac{\partial}{\partial\beta_0}\mathrm{RSS}(\beta_0,\beta_1) = 0$ and $\frac{\partial}{\partial\beta_1}\mathrm{RSS}(\beta_0,\beta_1) = 0$.

[5] The sample standard deviation is $s_x = \sqrt{s_x^2}$.

```
## 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

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

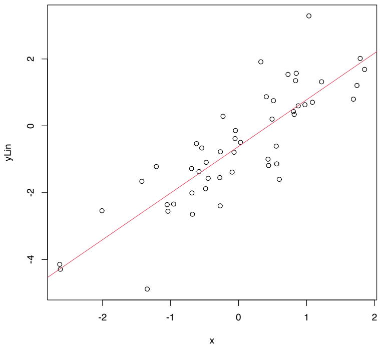


Figure 2.3: Linear fit for the data employed in Figure 2.2.1 minimizing the RSS.

```
# Access coefficients with $coefficients
mod$coefficients
## (Intercept)           x
##  -0.6153744   1.3950973

# Compute the minimized RSS
sum((yLin - mod$coefficients[1] - mod$coefficients[2] * x)^2)
## [1] 41.02914
sum(mod$residuals^2)
## [1] 41.02914

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

**Exercise 2.1.** Check that you cannot 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. They are *random variables*, since they are computed from the random sample $(X_1, Y_1), \ldots, (X_n, Y_n)$.

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.

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

# Graphical -- pairwise relations with linear and "smooth" regressions
car::scatterplotMatrix(wine, col = 1, regLine = list(col = 2),
                       smooth = list(col.smooth = 4, col.spread = 4))
```

As we can see, Year and FrancePop are very dependent, and Year and Age are *perfectly* dependent. This is so because Age = 1983 - 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:

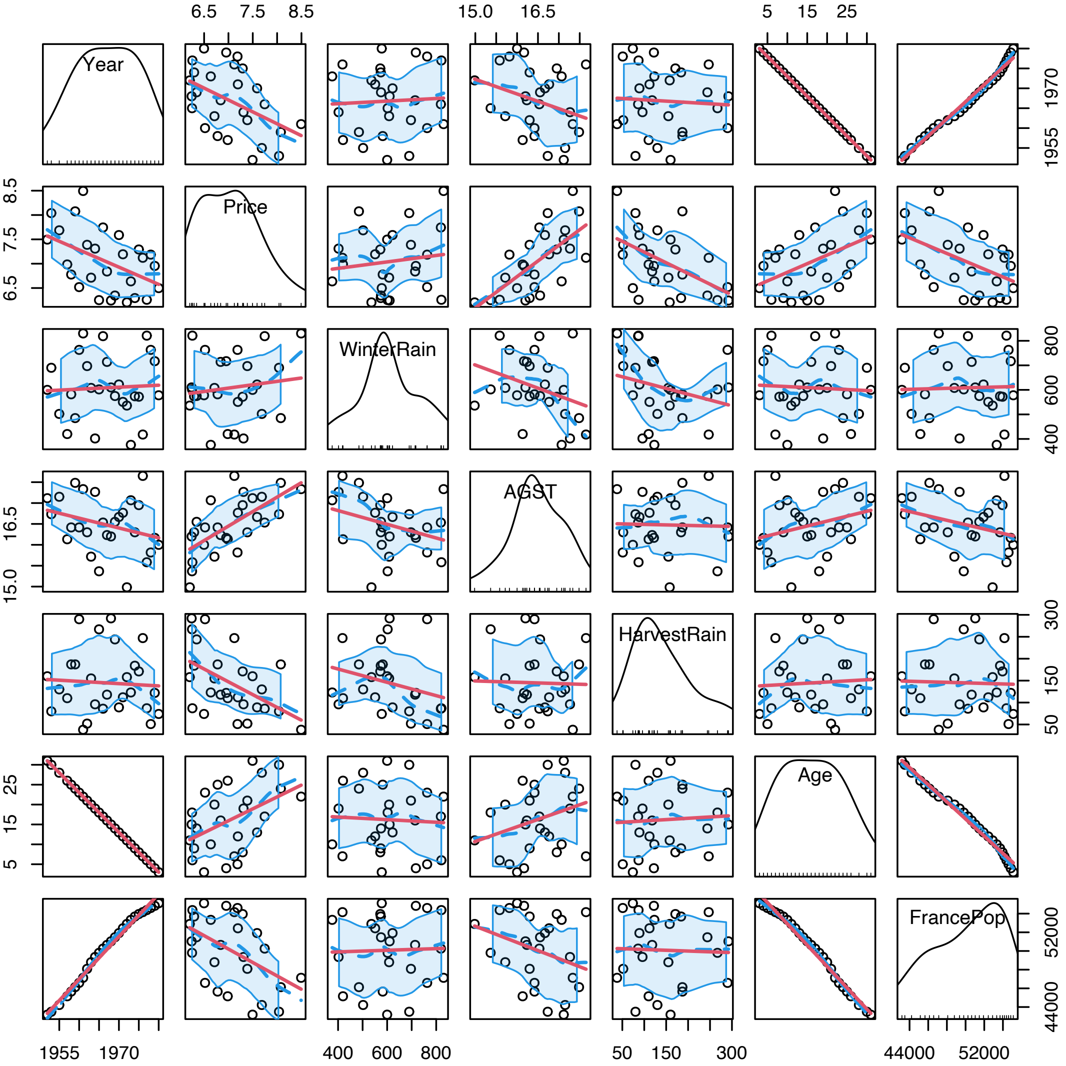


Figure 2.4: Scatterplot matrix for the `wine` dataset. The diagonal plots show density estimators of the pdf of each variable (see Section 6.1.2). The $(i, j)$-th scatterplot shows the data of $X_i$ vs. $X_j$, where the red line is the regression line of $X_i$ (response) on $X_j$ (predictor) and the blue curve represents a smoother that estimates nonparametrically the regression function of $X_i$ on $X_j$ (see Section 6.2). The dashed blue curves are the confidence intervals associated to the nonparametric smoother.

```
# 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 in Section 2.7.1, 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"  "aliased"       "sigma"         "df"
##  [8] "r.squared"     "adj.r.squared" "fstatistic"    "cov.unscaled"

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

**Exercise 2.2.** Complete the analysis by computing the linear models `Price ~ FrancePop`, `Price ~ Age`, `Price ~ WinterRain`, and `Price ~ HarvestRain`. Name them as `modFrancePop`, `modAge`, `modWinterRain`, and `modHarvestRain`. Obtain their $R^2$'s 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 with a more advanced model.

### 2.2.3 Multiple linear model

The multiple linear model extends the simple linear model by describing the relation between several random variables $X_1, \ldots, X_p$ [6] and $Y$. Therefore, as before, the multiple linear model is *constructed by assuming* that the linear relation

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon \tag{2.4}$$

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

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \tag{2.5}$$

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

The LHS of (2.5) is the conditional expectation of $Y$ given $X_1, \ldots,$ $X_p$. 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 $X_1, \ldots, X_p$. Hence the neat **interpretation of the coefficients**:

- $\beta_0$: is the mean of $Y$ when $X_1 = \cdots = X_p = 0$.
- $\beta_j$, $1 \leq j \leq p$: is the increment in the mean of $Y$ for an increment of one unit in $X_j = x_j$, provided that the rest of predictors $X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_p$ *remain constant*.

Figure 2.2.3 illustrates the geometrical interpretation of a multiple linear model: a hyperplane in $\mathbb{R}^{p+1}$. If $p = 1$, the hyperplane is actually a line, the regression line for simple linear regression. If $p = 2$, then the regression plane can be visualized in a 3-dimensional plot.

The estimation of $\beta_0, \beta_1, \ldots, \beta_p$ 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 on the hyperplane. Before doing so, we need to introduce some helpful matrix notation:

- A sample of $(X_1, \ldots, X_p, Y)$ is denoted as $\{(X_{i1}, \ldots, X_{ip}, Y_i)\}_{i=1}^n$, where $X_{ij}$ is the $i$-th observation of the $j$-th predictor $X_j$. We denote with $\mathbf{X}_i := (X_{i1}, \ldots, X_{ip})$ to the $i$-th observation of $(X_1, \ldots, X_p)$, so the sample simplifies to $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$.

- The *design matrix* contains all the information of the predictors plus a column of ones

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

[6] Note that now $X_1$ represents the first predictor and not the first element of a sample of $X$.

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

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

Thanks to the matrix notation, we can turn the sample version[7] of the multiple linear model, namely

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

into something as compact as

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

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 available here.

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

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

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

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

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*[8]:

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

Luckily, thanks to the matrix form of (2.6), it is possible[9] to compute a closed-form expression for the least squares estimates:

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

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 $\mathbf{X}$.

[7] Recall that (2.4) and (2.5) were referring to the relation of the random variable (or population) $Y$ with the random variables $X_1, \ldots, X_p$. Those are *population versions* of the linear model and clearly generate the *sample versions* when they are replicated for each observation $(\mathbf{X}_i, Y_i)$, $i = 1, \ldots, n$, of the random vector $(\mathbf{X}, Y)$.

[8] It can be seen that they are unique and that they always exist, provided that $\mathrm{rank}(\mathbf{X}^o p\mathbf{X}) = p + 1$.

[9] It follows from $\frac{\partial(\mathbf{A}\mathbf{x})}{\partial \mathbf{x}} = \mathbf{A}$ and $\frac{\partial(f(\mathbf{x})^\top g(\mathbf{x}))}{\partial \mathbf{x}} = f(\mathbf{x})^\top \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} + g(\mathbf{x})^\top \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ for two vector-valued functions $f, g : \mathbb{R}^p \to \mathbb{R}^m$, where $\frac{\partial(f(\mathbf{x})^\top g(\mathbf{x}))}{\partial \mathbf{x}}$ is the gradient *row* vector of $f^\top g$, and $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ and $\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}$ are the Jacobian *matrices* of $f$ and $g$, respectively.

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

```
load(file = "least-squares-3D.RData")
```

Let's compute the coefficients of the regression of `yLin` on 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 added and then the regression is done on the sum.[10] Instead, this syntax is designed to resemble the mathematical form of 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

# Matrix X
X <- cbind(1, leastSquares3D$x1, leastSquares3D$x2)

# Vector Y
Y <- leastSquares3D$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]
## [1,] -0.5702694
## [2,]  0.4832624
## [3,]  0.3214894
```

**Exercise 2.3.** Compute $\hat{\beta}$ for the regressions `yLin ~ x1 + x2`, `yQua ~ x1 + x2`, and `yExp ~ x2 + x3` using:

a. Equation (2.7) and
b. 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* $\hat{Y}_1, \ldots, \hat{Y}_n$, where

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

  They are the vertical projections of $Y_1, \ldots, Y_n$ onto the fitted plane (see Figure 2.2.3). In a matrix form, inputting (2.6)

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

  where $\mathbf{H} := \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ 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 2.2.3).

[10] If you wanted to do so, you will need to use the function `I()` for indicating that + is not including predictors in the model, but is acting as the algebraic sum operator.

- The *residuals* (or estimated errors) $\hat\varepsilon_1, \ldots, \hat\varepsilon_n$, where

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

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

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

```
# Fitted values
mod$fitted.values

# Residuals
mod$residuals
```

We conclude with an important insight on the relation of multiple and simple linear regressions that is illustrated in Figure 2.5. The data used in that figure 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)
```

Consider the multiple linear model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon_1$ and its associated simple linear models $Y = \alpha_0 + \alpha_1 X_1 + \varepsilon_2$ and $Y = \gamma_0 + \gamma_1 X_2 + \varepsilon_3$, where $\varepsilon_1, \varepsilon_2, \varepsilon_3$ are random errors. Assume that we have a sample $\{(X_{i1}, X_{i2}, Y_i)\}_{i=1}^n$. Then, in general, $\hat\alpha_0 \neq \hat\beta_0 \neq \hat\gamma_0$, $\hat\alpha_1 \neq \hat\beta_1$, and $\hat\gamma_1 \neq \hat\beta_2$. Even if $\alpha_0 = \beta_0 = \gamma_0$, $\alpha_1 = \beta_1$, and $\gamma_1 = \beta_2$. That is, in general, **the inclusion of a new predictor changes the coefficient estimates of the rest of predictors**.

**Exercise 2.4.** With the above `data`, check how the fitted coefficients change for `y ~ x1`, `y ~ x2`, and `y ~ x1 + x2`.

### 2.2.4 *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:
```

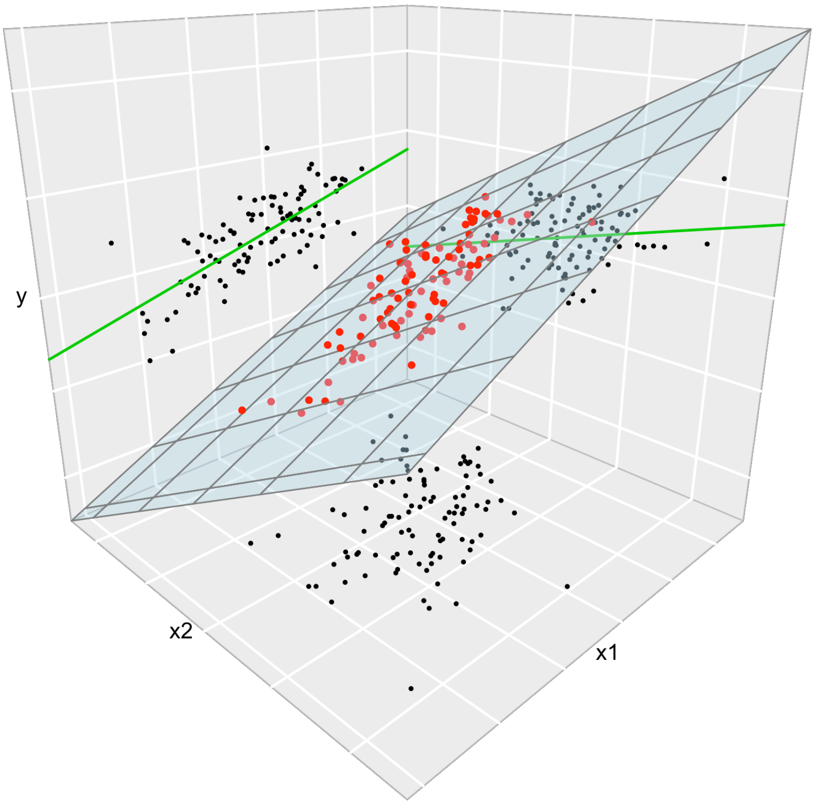


Figure 2.5: The regression plane (blue) of $Y$ on $X_1$ and $X_2$ and its relation with the simple linear regressions (green lines) of $Y$ on $X_1$ and of $Y$ on $X_2$. The red points represent the sample for $(X_1, X_2, Y)$ and the black points the sample projections for $(X_1, X_2)$ (bottom), $(X_1, Y)$ (left), and $(X_2, Y)$ (right). As it can be seen, the regression plane does not extend the simple linear regressions.

```
## (Intercept)   WinterRain         AGST  HarvestRain          Age    FrancePop
##  -2.343e+00    1.153e-03    6.144e-01   -3.837e-03    1.377e-02   -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 `Price` $= -2.343 + 0.013 \times$ `Age` $+0.614 \times$ `AGST` $-0.000 \times$ `FrancePop` $-0.003 \times$ `HarvestRain` $+0.001 \times$ `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 and depends on the dependence among the predictors.

## 2.3 *Assumptions of the model*

A natural[11] question to ask is: "Why do we need assumptions?" The answer is that we need *probabilistic assumptions* to ground **statistical inference** about the model parameters. Or, in other words, to quantify the variability of the estimator $\hat{\boldsymbol{\beta}}$ and to infer properties about the unknown population coefficients $\boldsymbol{\beta}$ from the sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$.

The assumptions of the multiple linear model are:

i. **Linearity**: $\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$.
ii. **Homoscedasticity**: $\mathbb{V}\mathrm{ar}[\varepsilon \mid X_1 = x_1, \ldots, X_p = x_p] = \sigma^2$.
iii. **Normality**: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.
iv. **Independence of the errors**: $\varepsilon_1, \ldots, \varepsilon_n$ 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)

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

Recall that, except assumption iv, the rest are expressed in terms of the random variables, not in terms of the sample. Thus they are population versions, rather than sample versions. It is however trivial to express (2.8) in terms of assumptions about the sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$:

$$Y_i \mid (X_{i1} = x_{i1}, \ldots, X_{ip} = x_{ip}) \sim \mathcal{N}(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}, \sigma^2), \quad (2.9)$$

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

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

Figures 2.8 and 2.9 represent situations where the assumptions of the model for $p = 1$ are respected and violated, respectively.

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 available here.

[11] After all, we already have a neat way of estimating $\boldsymbol{\beta}$ from the data... Isn't it all is needed?

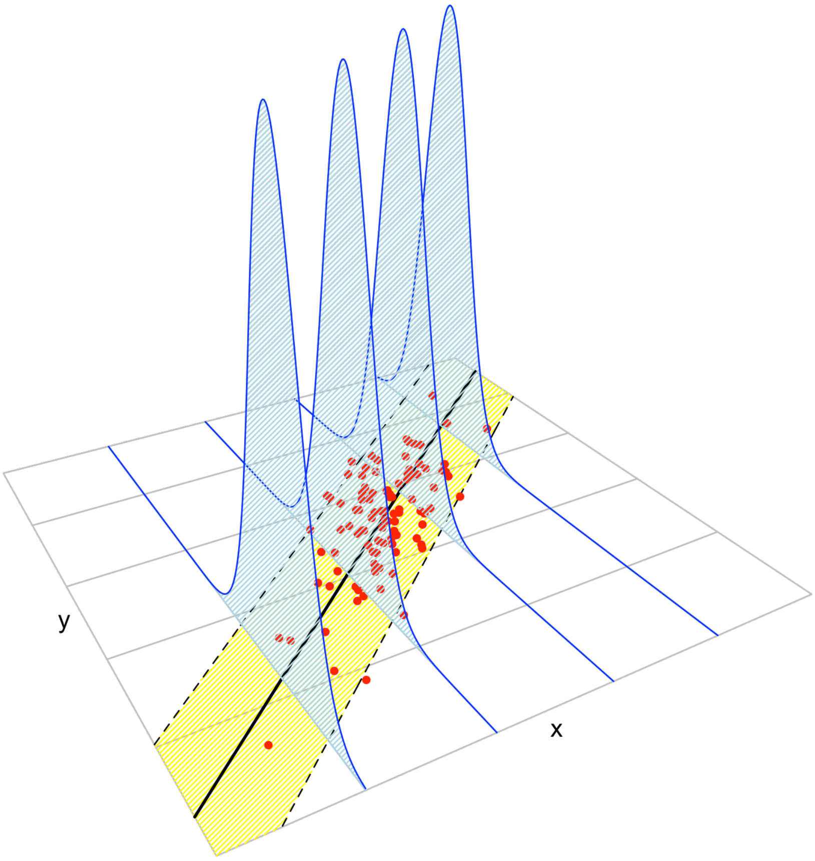


Figure 2.6: The key concepts of the simple linear model. The red points represent a sample with population regression line $y = \beta_0 + \beta_1 x$ given by the black line. The yellow band denotes where 95% of the data is, according to the model. The blue densities represent the conditional density of $Y$ given $X = x$, whose means lie in the regression line.

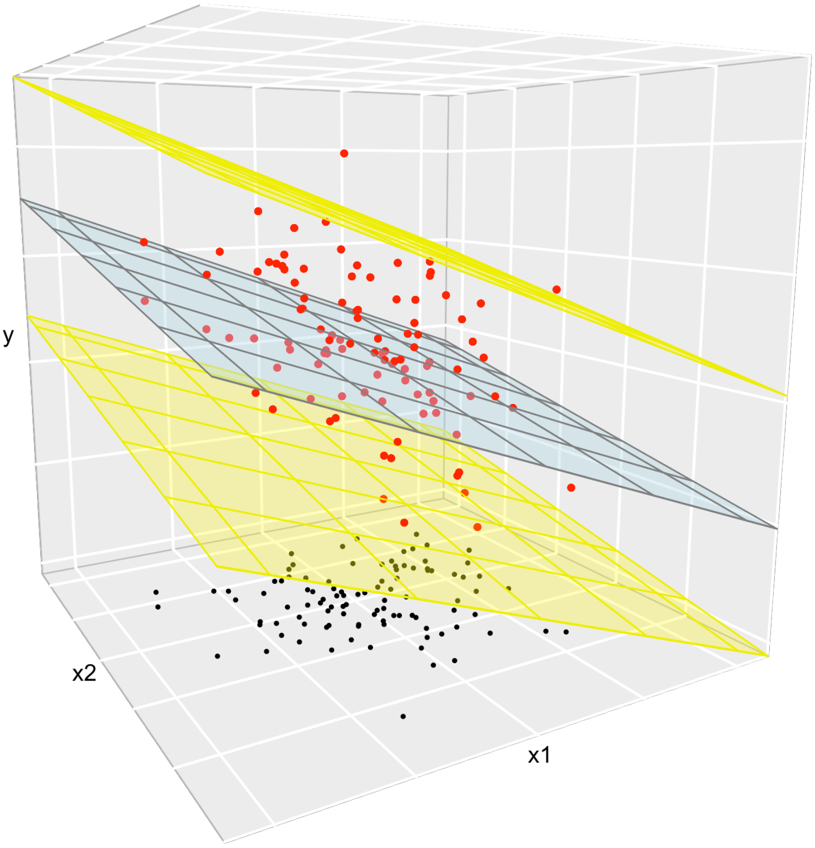


Figure 2.7: The key concepts of the multiple linear model when $p = 2$. The red points represent a sample with population regression plane $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ given by the blue plane. The black points represent the associated observations of the predictors. The space between the yellow planes denotes where 95% of the data is, according to the model.

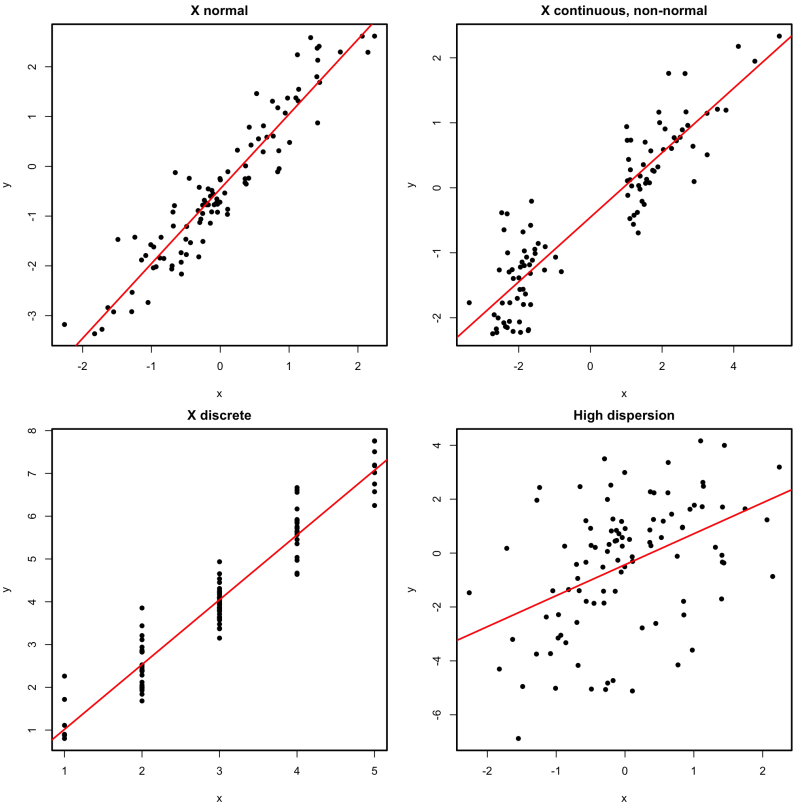


Figure 2.8: Perfectly valid simple linear models (all the assumptions are verified).

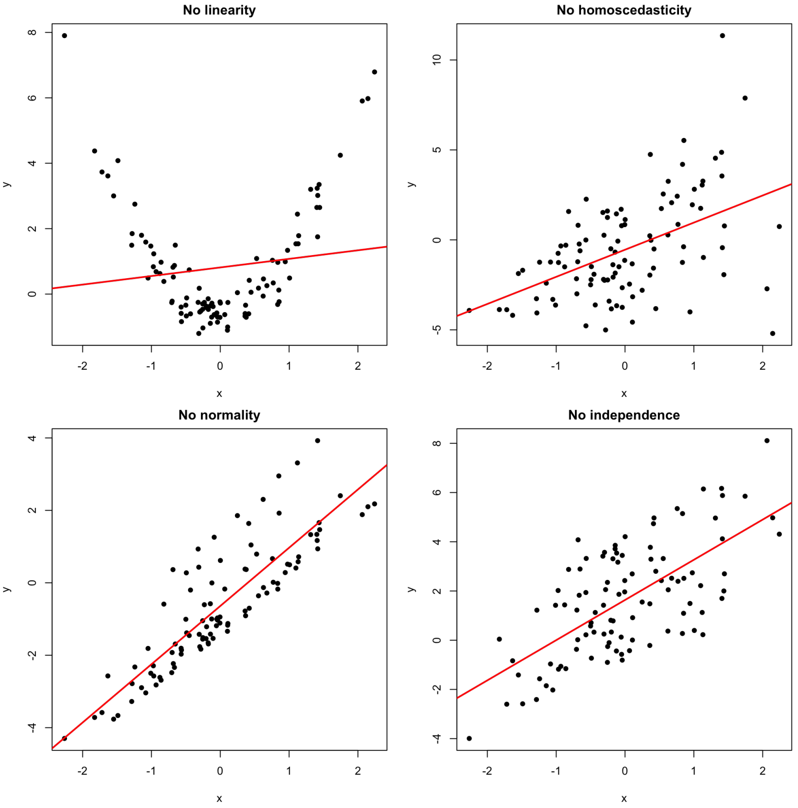


Figure 2.9: Problematic simple linear models (a single assumption does not hold).

**Exercise 2.5.** The dataset `assumptions.RData` contains the variables `x1`, ..., `x9` and `y1`, ..., `y9`. For each regression `y1 ~ x1`, ..., `y9 ~ x9`:

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

## 2.4 *Inference for model parameters*

The assumptions introduced in the previous section allow us to specify what is the distribution of the *random vector* $\hat{\boldsymbol{\beta}}$. The distribution is derived conditionally on the predictors' sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$. In other words, we assume that the randomness of $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ comes only from the error terms and not from the predictors.[12] To denote this, we employ lowercase for the predictors' sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

### 2.4.1 *Distributions of the fitted coefficients*

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

$$\hat{\boldsymbol{\beta}} \sim \mathcal{N}_{p+1}\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}\right). \tag{2.11}$$

This result can be obtained from the form of $\hat{\boldsymbol{\beta}}$ given in (2.7), the sample version of the model assumptions given in (2.10), and the linear transformation property of a normal given in (1.4). Equation (2.11) implies that the marginal distribution of $\hat{\beta}_j$ is

$$\hat{\beta}_j \sim \mathcal{N}\left(\beta_j, \mathrm{SE}(\hat{\beta}_j)^2\right), \tag{2.12}$$

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

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

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

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

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

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

with

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

Some insights on (2.13) and (2.14), illustrated interactively in Figure 2.4.1, are the following:

- **Bias**. Both estimates are unbiased. That means that their expectations are the true coefficients for any sample size $n$.

- **Variance**. The variances $\mathrm{SE}(\hat{\beta}_0)^2$ and $\mathrm{SE}(\hat{\beta}_1)^2$ have interesting interpretations in terms of their components:

[12] This is for theoretical and modeling convenience. With this assumption, we just model the randomness of $Y$ given the predictors. If the randomness of $Y$ *and* the randomness of $X_1, \ldots, X_n$ was to be modeled, we will require from a significantly more complex model.

- *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 predictors' sample $x_1, \ldots, x_n$ is fixed and new responses $Y_1, \ldots, Y_n$ are generated each time from a linear model $Y = \beta_0 + \beta_1 X + \varepsilon$. Application available here.

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

- **Bias**. All the estimates are unbiased for any sample size $n$.
- **Variance**. It depends on:
  - *Sample size $n$.* Hidden inside $\mathbf{X}^\top\mathbf{X}$. As $n$ grows, the precision of the estimators increases.
  - *Error variance $\sigma^2$.* The larger $\sigma^2$ is, the less precise $\hat{\boldsymbol{\beta}}$ is.
  - *Predictor sparsity $(\mathbf{X}^\top\mathbf{X})^{-1}$.* The more "disperse"[13] the predictors are, the more precise $\hat{\boldsymbol{\beta}}$ is.

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

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

Note the $n-p-1$ in the denominator. The factor $n-p-1$ represents the *degrees of freedom*: the number of data points minus the number of *already*[14] fitted parameters ($p$ slopes plus 1 intercept) with the data. For the interpretation of $\hat{\sigma}^2$, it is key to realize that *the mean of the residuals $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$ 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 $\varepsilon$.[15] It can be seen that $\hat{\sigma}^2$ is unbiased as an estimator of $\sigma^2$.

If we use the estimate $\hat{\sigma}^2$ instead of $\sigma^2$, we get more useful[16] distributions than (2.12):

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

[13] Undestood as small $|(\mathbf{X}^\top\mathbf{X})^{-1}|$.

[14] Prior to undertaking the estimation of $\sigma$ we have used the sample to estimate $\hat{\boldsymbol{\beta}}$. The situation is thus analogous to the discussion between the *sample variance* $s_x^2 = \frac{1}{n}\sum_{i=1}^n (X_i - \bar{X})^2$ and the *sample quasi-variance* $\hat{s}_x^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i - \bar{X})^2$ that are computed from a sample $X_1, \ldots, X_n$. When estimating $\mathbb{V}\mathrm{ar}[X]$, both estimate previously $\mathbb{E}[X]$ through $\bar{X}$. The fact that $\hat{s}_x^2$ accounts for that prior estimation through the degrees of freedom $n-1$ makes that estimator unbiased for $\mathbb{V}\mathrm{ar}[X]$ ($s_x^2$ is not).

[15] Recall that the sample variance of $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$ is $\frac{1}{n}\sum_{i=1}^n (\hat{\varepsilon}_i - \bar{\hat{\varepsilon}})^2$.

[16] In the sense of practically realistic.

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 $\beta_j$, $j = 0, \ldots, p$. We will employ them for building confidence intervals and hypothesis tests in what follows.

### 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 $\beta_j$, $j = 0, \ldots, p$:

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

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.

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 available here.

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.

### 2.4.3 *Testing on the coefficients*

The distributions in (2.16) allow also to conduct a formal *hypothesis test* on the coefficients $\beta_j$, $j = 0, \ldots, p$. For example the test for *significance*[17] is especially important, that is, the test of the hypotheses

$$H_0 : \beta_j = 0$$

for $j = 0, \ldots, p$. 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

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

which is distributed as a $t_{n-p-1}$ *under the (veracity of) the null hypothesis*.[18]

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 *two-sided* (we will focus mostly on these alternatives), such as

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

[17] Shortcut for *significantly different from zero*.

[18] This is denoted as $\frac{\hat{\beta}_j - 0}{\hat{\mathrm{SE}}(\hat{\beta}_j)} \overset{H_0}{\sim} t_{n-p-1}$.

or *one-sided*, such as

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

The test based on the $t$-statistic is referred to as the *$t$-test*. It rejects $H_0 : \beta_j = 0$ (against $H_1 : \beta_j \neq 0$) at significance level $\alpha$ for large *absolute* values of the $t$-statistic, precisely for those above the $\alpha/2$-upper quantile of the $t_{n-p-1}$ distribution. That is, it rejects $H_0$ at level $\alpha$ if $\frac{|\hat\beta_j|}{\hat{\mathrm{SE}}(\hat\beta_j)} > t_{n-p-1;\alpha/2}$.[19] For the one-sided tests, it rejects $H_0$ against $H_1 : \beta_j < 0$ or $H_1 : \beta_j > 0$ if $\frac{\hat\beta_j}{\hat{\mathrm{SE}}(\hat\beta_j)} < -t_{n-p-1;\alpha}$ or $\frac{\hat\beta_j}{\hat{\mathrm{SE}}(\hat\beta_j)} > t_{n-p-1;\alpha}$, respectively.

Remember the following insights about hypothesis testing.

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

In a 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 one-sided test $H_0 : \beta_j = 0$ vs. $H_1 : \beta_j < 0$ (respectively, $H_1 : \beta_j > 0$) 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**.

[19] In R, $t_{n-p-1;\alpha/2}$ can be computed as `qt(p = 1 - alpha / 2, df = n - p - 1)` or `qt(p = alpha / 2, df = n - p - 1, lower.tail = FALSE)`.

### 2.4.4 *Case study application*

Let's analyze the multiple linear model we have considered for the `wine` dataset, 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 `summary` output 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 $\hat{\mathrm{SE}}(\hat{\beta}_j)$.
- `t value`: $t$-statistic $\frac{\hat{\beta}_j}{\hat{\mathrm{SE}}(\hat{\beta}_j)}$.
- `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 asterisks, the more evidence supporting that $H_0$ does not hold.[20]

Note that **a high proportion of predictors are not significant** in `modWine1`: `FrancePop` and `Age` are not significant (and the intercept is not significant also). This is an indication of an **excess of predictors** adding little information to the response. One explanation is the almost perfect correlation between `FrancePop` and `Age`

[20] For example, `'**'` indicates that the $p$-value lies within 0.001 and 0.01.

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.[21] A data-driven justification of the removal of this variable is that it is the least significant in `modWine1`.

Then, the model without `FrancePop`[22] 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 `car::compareCoefs`:

```
car::compareCoefs(modWine1, modWine2)
## Calls:
## 1: lm(formula = Price ~ ., data = wine)
## 2: lm(formula = Price ~ . - FrancePop, data = wine)
##
##               Model 1   Model 2
## (Intercept)     -2.34     -3.65
## SE               7.70      1.69
##
## WinterRain   0.001153  0.001167
## SE           0.000499  0.000482
##
## AGST           0.6144    0.6164
## SE             0.0980    0.0952
##
## HarvestRain -0.003837 -0.003861
## SE           0.000837  0.000808
##
## Age           0.01377   0.02385
```

[21] This is a *context-guided* decision, not *data-driven*.

[22] Notice the use of `-` for *excluding* a particular predictor.

```
## SE            0.05821   0.00717
##
## FrancePop   -2.21e-05
## SE           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 model `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) 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 2.5. Therefore, the precise statement to make is:

> **In model** `Price ~ Age + WinterRain`, with $\alpha = 0.05$, the coefficient of `WinterRain` is non-significant.

Note that this **does not** mean that the coefficient will be always non-significant: in `Price ~ Age + AGST + HarvestRain + WinterRain` it is.

**Exercise 2.6.** Compute and interpret the CIs for the coefficients, at levels $\alpha = 0.10, 0.05, 0.01$, for the following regressions:

a. `Price ~ WinterRain + HarvestRain + AGST` (`wine`).
b. `AGST ~ Year + FrancePop` (`wine`).

**Exercise 2.7.** For the `assumptions` dataset, do the following:

a. Regression `y7 ~ x7`. Check that:

   - The intercept is not significant for the regression at any reasonable level $\alpha$.
   - The slope is significant for any $\alpha \geq 10^{-7}$.

b. 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?

In certain applications, it is useful to *center* the predictors $X_1, \ldots, X_p$ prior to fit the model, in such a way that the slope coefficients $(\beta_1, \ldots, \beta_p)$ measure the effects of deviations of the predictors from their means. Theoretically, this amounts to considering the linear model

$$Y = \beta_0 + \beta_1(X_1 - \mathbb{E}[X_1]) + \cdots + \beta_p(X_p - \mathbb{E}[X_p]) + \varepsilon.$$

In the sample case, we proceed by replacing $X_{ij}$ by $X_{ij} - \bar{X}_j$, which can be easily done by the `scale` function (see below). If, in addition, the response is also centered, then $\beta_0 = 0$ and $\hat{\beta}_0 = 0$. This centering of the data has no influence on the significance of the predictors (but has influence on the significance of $\hat{\beta}_0$), as it is just a linear transformation of them.

```
# By default, scale centers (subtracts the mean) and scales (divides by the
# standard deviation) the columns of a matrix
wineCen <- data.frame(scale(wine, center = TRUE, scale = FALSE))

# Regression with centered response and predictors
modWine3Cen <- lm(Price ~ Age + WinterRain, data = wineCen)

# Summary
summary(modWine3Cen)
##
## Call:
## lm(formula = Price ~ Age + WinterRain, data = wineCen)
##
## 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) 5.964e-16  1.110e-01   0.000   1.0000
## Age         3.606e-02  1.374e-02   2.625   0.0149 *
## WinterRain  7.813e-04  8.780e-04   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
```

## 2.5 *Prediction*

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

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

There are similarities and differences in the prediction of the conditional mean $\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$ and conditional response $Y \mid \mathbf{X} = \mathbf{x}$, which we highlight next:

- *Similarities*. The estimate is the same,[23] $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$. The CIs for both quantities are centered in $\hat{y}$.
- *Differences*. $\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$ is deterministic and $Y \mid \mathbf{X} = \mathbf{x}$ is a random variable. The prediction of the latter is noisier, because it has to take into account the randomness of $Y$. Therefore, the variance is larger for the prediction of $Y \mid \mathbf{X} = \mathbf{x}$ than for the prediction of $\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$. This has a direct consequence on the length of the prediction intervals, which are longer for $Y \mid \mathbf{X} = \mathbf{x}$ than for $\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$.

The inspection of the CIs for the conditional mean and conditional response in the simple linear model offers great insight into the previous similarities and differences, and also on what components affect precisely the quality of the prediction:

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

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

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

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

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 available here.

Notice the dependence of both CIs on $x$, $n$, and $\hat{\sigma}^2$, each of them with a clear effect on the resulting length of the interval. Note also the high similarity between (2.18) and (2.19) (both intervals are centered at $\hat{y}$ and have a similar variance) and its revealing unique difference: the extra $\hat{\sigma}^2$ in (2.19),[24] consequence of the "extra randomness" of the conditional response with respect to the conditional mean. Figure 2.5 helps to visualize these concepts and the difference between CIs interactively.

### 2.5.1 Case study application

The prediction and the 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 $\mathbf{x} = (x_1, \ldots, x_p)$ where we want to predict $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$. The prediction is $\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$ and the CIs returned are either (2.18) or (2.19).

[23] Because the prediction of a new observation from the random variable $\mathcal{N}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \sigma^2)$ is simply its mean, $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$, which is also the most likely value in a normal.

[24] A consequence of this extra $\hat{\sigma}^2$ is that the length of (2.19) cannot be reduced arbitrarily if the sample size $n$ grows.

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

```
# 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's 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
## 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
```

**Exercise 2.8.** For the `wine` dataset, do the following:

a. Regress `WinterRain` on `HarvestRain` and `AGST`. Name the fitted model `modExercise`.
b. Compute the estimate for the conditional mean of `WinterRain` for `HarvestRain` $=$ 123.0 and `AGST` $=$ 16.15. What is the CI at $\alpha = 0.01$?
c. Compute the estimate for the conditional response for `HarvestRain` $=$ 125.0 and `AGST` $=$ 15. What is the CI at $\alpha = 0.10$?
d. 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 in prediction. In this section we will see how the variance of $Y$ is decomposed into two parts, each 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 $\bar{Y} = \bar{\hat{Y}}$.[25] The ANOVA decomposition considers the following measures of variation related with the response:

- $\text{SST} := \sum_{i=1}^n \left(Y_i - \bar{Y}\right)^2$, the **Total Sum of Squares**. This is the *total variation* of $Y_1, \ldots, Y_n$, since $\text{SST} = ns_y^2$, where $s_y^2$ is the sample variance of $Y_1, \ldots, Y_n$.
- $\text{SSR} := \sum_{i=1}^n \left(\hat{Y}_i - \bar{Y}\right)^2$, 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* $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_p X_{ip}$. Also, $\text{SSR} = ns_{\hat{y}}^2$, where $s_{\hat{y}}^2$ is the sample variance of $\hat{Y}_1, \ldots, \hat{Y}_n$.

[25] This is an important result that can be checked using the matrix notation introduced in Section 2.2.3.

- SSE $:= \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2$, the **Sum of Squared Errors.**[26] Is the variation around the conditional mean. Recall that SSE $= \sum_{i=1}^n \hat{\varepsilon}_i^2 = (n-p-1)\hat{\sigma}^2$, where $\hat{\sigma}^2$ is the rescaled sample variance of $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$.

The ANOVA decomposition states that:

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

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 Figure 2.10. Figure 2.6 dynamically shows how the ANOVA decomposition places more weight on SSR or SSE according to $\hat{\sigma}^2$ (which is obviously driven by the value of $\sigma^2$).

The ANOVA table summarizes the decomposition of the variance:

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

$$H_0 : \beta_1 = \cdots = \beta_p = 0 \quad \text{vs.} \quad H_1 : \beta_j \neq 0 \text{ for any } j \geq 1,$$

that is, the hypothesis of *no linear dependence of $Y$ on $X_1, \ldots, X_p$*.[27] This is the so-called *F-test* and, if $H_0$ is rejected, allows one to conclude that **at least one $\beta_j$ is significantly different from zero.**[28] It happens that

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

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.[29] The $F$-test rejects at significance level $\alpha$ for large values of the $F$-statistic, precisely for those above the $\alpha$-upper quantile of the $F_{p,n-p-1}$ distribution, denoted by $F_{p,n-p-1;\alpha}$.[30] That is, $H_0$ is rejected if $F > F_{p,n-p-1;\alpha}$.

Illustration of the ANOVA decomposition and its dependence on $\sigma^2$ and $\hat{\sigma}^2$. Larger (respectively, smaller) $\hat{\sigma}^2$ results in more weight placed on the SSE (SSR) term. Application available here.

[26] Recall that SSE and RSS (of the least squares estimator $\hat{\boldsymbol{\beta}}$) are two names for the *same quantity* (that appears in different contexts): SSE $= \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2 = \sum_{i=1}^n \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} - \cdots - \hat{\beta}_p X_{ip}\right)^2 = \text{RSS}(\hat{\boldsymbol{\beta}})$.

[27] Geometrically: the plane is completely flat, it does not have any inclination in the $Y$ direction.

[28] And therefore, there is a statistical meaningful (i.e., not constant) linear trend to model.

[29] Little variation is explained by the regression model since $\hat{\boldsymbol{\beta}} \approx \mathbf{0}$.

[30] In R, `qf(p = 1 - alpha, df1 = n - p - 1, df2 = p)` or `qf(p = alpha, df1 = n - p - 1, df2 = p, lower.tail = FALSE)`.

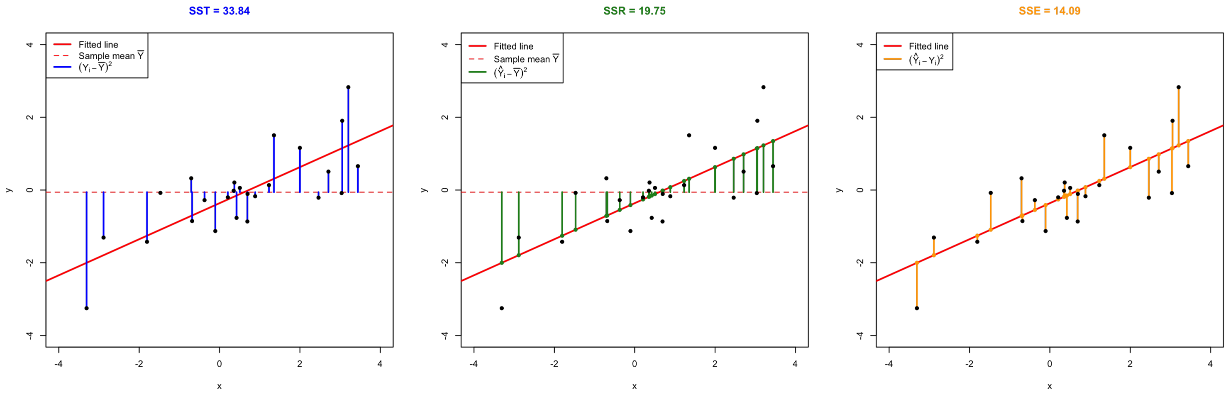


Figure 2.10: Visualization of the ANOVA decomposition. SST measures the variation of $Y_1,\ldots,Y_n$ with respect to $\bar{Y}$. SSR measures the variation of $\hat{Y}_1,\ldots,\hat{Y}_n$ with respect to $\bar{\hat{Y}} = \bar{Y}$. SSE collects the variation between $Y_1,\ldots,Y_n$ and $\hat{Y}_1,\ldots,\hat{Y}_n$, that is, the variation of the residuals.

The "ANOVA table" is a broad concept in statistics, with different variants. Here we are only covering the basic ANOVA table from the relation SST $=$ SSR + SSE. However, further sophistication is 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 takes a model as an input and returns the following *sequential* ANOVA table:[31]

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

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

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

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

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

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

In order to *exactly*[32] compute the simplified ANOVA table seen before, we can rely on the following ad-hoc function. The function takes as input a fitted `lm`:

[31] More complex – included here just for clarification of the `anova`'s output.

[32] Note that, if `mod <- lm(resp ~ preds, data)` represents a model with response `resp` and predictors `preds`, and `mod0`, is the intercept-only model `mod0 <- lm(resp ~ 1, data)` that does not contain predictors, `anova(mod0, mod)` gives a *similar*, output to the seen ANOVA table. Precisely, the first row of the outputted table stands for the SST and the second row for the SSE row (so we call it the SST–SSE table). The SSR row is not present. The seen ANOVA table (which contains SSR and SSE) and the SST–SSE table encode the same information due to the ANOVA decomposition. So it is a matter of taste and tradition to employ one or

```
# 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 not 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
```

**Exercise 2.9.** 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.

**Exercise 2.10.** 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 to the ANOVA decomposition. It is defined as

$$R^2 := \frac{\text{SSR}}{\text{SST}} = \frac{\text{SSR}}{\text{SSR}+\text{SSE}} = \frac{\text{SSR}}{\text{SSR}+(n-p-1)\hat\sigma^2}. \tag{2.21}$$

The $R^2$ measures the **proportion of variation** of the response variable $Y$ that is **explained** by the predictors $X_1, \ldots, X_p$ through the regression. Intuitively, $R^2$ measures the *tightness of the data cloud around the regression plane*. Check in Figure 2.6 how the value of $\sigma^2$[33] affects the $R^2$.

The sample correlation coefficient is intimately related with the $R^2$. For example, if $p = 1$, then it can be seen (exercise below) that $R^2 = r^2_{xy}$. More importantly, $R^2 = r^2_{y\hat y}$ for any $p$, that is, the square of the sample correlation coefficient between $Y_1, \ldots, Y_n$ and $\hat Y_1, \ldots, \hat Y_n$ is $R^2$, a fact that is not immediately evident. Let's check this fact when $p = 1$ by relying on $R^2 = r^2_{xy}$. First, by the form of $\hat\beta_0$ given in (2.3),

$$\begin{aligned}\hat Y_i &= \hat\beta_0 + \hat\beta_1 X_i\\ &= (\bar Y - \hat\beta_1 \bar X) + \hat\beta_1 X_i\\ &= \bar Y + \hat\beta_1 (X_i - \bar X).\end{aligned} \tag{2.22}$$

Then, replace (2.22) in

$$\begin{aligned}r^2_{y\hat y} &= \frac{s^2_{y\hat y}}{s^2_y s^2_{\hat y}}\\ &= \frac{\left(\sum_{i=1}^n \left(Y_i - \bar Y\right)\left(\hat Y_i - \bar Y\right)\right)^2}{\sum_{i=1}^n \left(Y_i - \bar Y\right)^2 \sum_{i=1}^n \left(\hat Y_i - \bar Y\right)^2}\\ &= \frac{\left(\sum_{i=1}^n \left(Y_i - \bar Y\right)\left(\bar Y + \hat\beta_1(X_i - \bar X) - \bar Y\right)\right)^2}{\sum_{i=1}^n \left(Y_i - \bar Y\right)^2 \sum_{i=1}^n \left(\bar Y + \hat\beta_1(X_i - \bar X) - \bar Y\right)^2}\\ &= r^2_{xy}.\end{aligned}$$

As a consequence, $r^2_{y\hat y} = r^2_{xy} = R^2$ when $p = 1$.

[33] Which is not the $\hat\sigma^2$ in (2.21), but $\hat\sigma^2$ is obviously dependent on $\sigma^2$.

**Exercise 2.11.** 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. 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, the linear models have large $R^2$'s. These counterexamples emphasize that **inference**, built on the validity of the model assumptions,[34] **will be problematic** if these assumptions are violated, no matter what is the value of $R^2$. For example, recall how biased the predictions and their associated CIs will be in $x = 0.35$ and $x = 0.65$.

```
# Simple linear model

# Create data that:
# 1) does not follow a linear model
# 2) the error is heteroscedastic
x <- seq(0.15, 1, l = 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
car::scatterplot(y ~ x, col = 1, regLine = list(col = 2), smooth = FALSE)


# Multiple linear model

# Create data that:
# 1) does not follow a linear model
# 2) the error is heteroscedastic
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!?
```

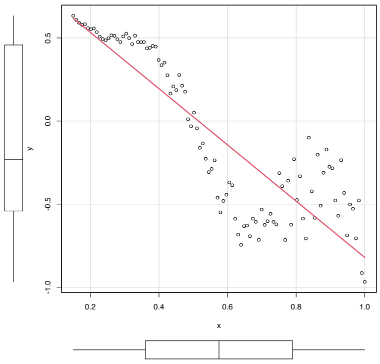


Figure 2.11: Regression line for a dataset that clearly violates the linearity and homoscedasticity assumptions. The $R^2$ is, nevertheless, as high as (approximately) 0.77.

[34] Which do not hold!

```
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
car::scatter3d(y ~ x1 + x2, fit = "linear")
rgl::rglwidget()
```

The previous counterexamples illustrate that a large $R^2$ means *nothing* in terms of inference if the assumptions of the model do not hold.

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 $X_1, \ldots, X_p$ 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 $\hat{\boldsymbol{\beta}}$.

The ANOVA decomposition gives another view on the least-squares estimates: **$\hat{\boldsymbol{\beta}}$ are the estimated coefficients that maximize the $R^2$** (among all the possible estimates we could think about). To see this, recall that

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

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

### 2.7.2 The $R^2_{Adj}$

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

$$\begin{aligned} R^2 =& \frac{\text{SSR}}{\text{SST}} = \frac{\text{SST} - \text{SSE}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}} \\ =& 1 - \frac{\hat{\sigma}^2}{\text{SST}} \times (n-p-1). \end{aligned} \tag{2.23}$$

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 (2.23) is reduced and, as a consequence, $R^2$ grows.

The *adjusted* $R^2$, $R^2_{\text{Adj}}$, is an important quantity specifically designed to overcome this $R^2$'s flaw, ubiquitous in multiple linear regression. The purpose is to have a better tool for *comparing models without systematically favoring more complex models.*[35] This alternative coefficient is defined as

$$\begin{aligned} R^2_{\text{Adj}} :=& 1 - \frac{\text{SSE}/(n-p-1)}{\text{SST}/(n-1)} = 1 - \frac{\text{SSE}}{\text{SST}} \times \frac{n-1}{n-p-1} \\ =& 1 - \frac{\hat{\sigma}^2}{\text{SST}} \times (n-1). \end{aligned} \tag{2.24}$$

The $R^2_{\text{Adj}}$ is independent of $p$, at least explicitly. If $p = 1$ then $R^2_{\text{Adj}}$ is *almost* $R^2$ (practically identical if $n$ is large). Both (2.23) and (2.24) are quite similar except for the last factor, which in the latter does not depend on $p$. Therefore, (2.24) 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 $R^2_{\text{Adj}}$ can be visualized with the following simulation study. Suppose that we generate a random dataset $\{(X_{i1}, X_{i2}, Y_i)\}_{i=1}^n$, with $n = 200$ observations of two predictors $X_1$ and $X_2$ that are distributed as a $\mathcal{N}(0,1)$, and a response $Y$ generated by the linear model

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \varepsilon_i, \tag{2.25}$$

where $\varepsilon_i \sim \mathcal{N}(0,9)$. To this data, we add 196 "garbage" predictors $X_j \sim \mathcal{N}(0,1)$ that are completely independent from $Y$. Therefore, we end up with $p = 198$ predictors where only the first two ones are relevant for explaining $Y$. We compute now the $R^2(j)$ and $R^2_{\text{Adj}}(j)$ for the models

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_j X_j + \varepsilon, \tag{2.26}$$

with $j = 1, \ldots, p$, and we plot them as the curves $(j, R^2(j))$ and $(j, R^2_{\text{Adj}}(j))$. Since $R^2$ and $R^2_{\text{Adj}}$ are **random variables**, we repeat this procedure $M = 500$ times to have an idea of the variability behind $R^2$ and $R^2_{\text{Adj}}$. Figure 2.12 contains the results of this experiment.

[35] An informal way of regarding the difference between $R^2$ and $R^2_{\text{Adj}}$ is by thinking of $R^2$ as a measure of fit that "is not aware of the dangers of overfitting". In this interpretation, $R^2_{\text{Adj}}$ is the overfitting-aware version of $R^2$.

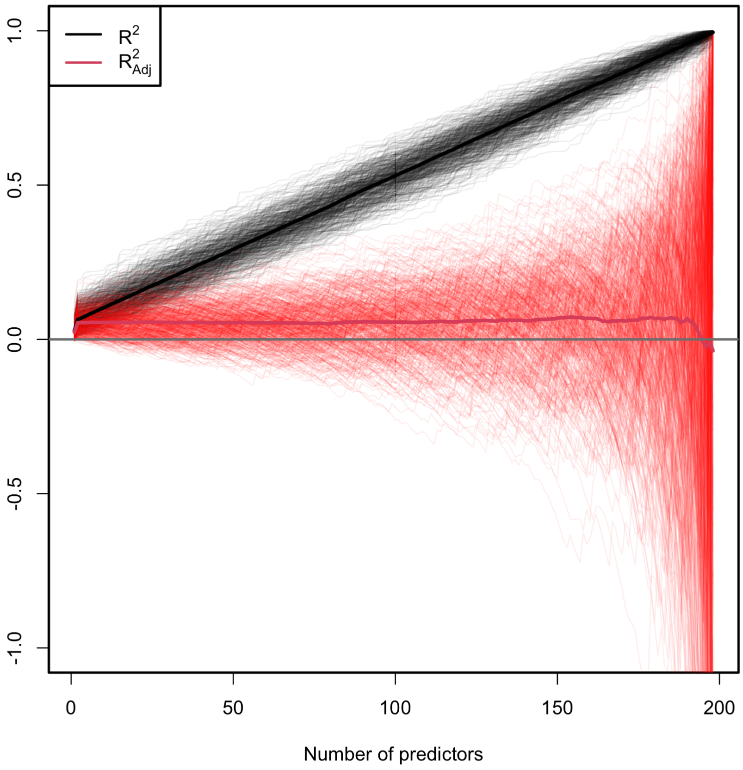


Figure 2.12: Comparison of $R^2$ and $R^2_{\text{Adj}}$ on the model (2.26) fitted with data generated by (2.25). The number of predictors $p$ ranges from 1 to 198, with only the *first two* predictors being significant. The $M = 500$ curves for each color arise from $M$ simulated datasets of sample size $n = 200$. The thicker curves are the mean of each color's curves.

As it can be seen, the $R^2$ increases linearly with the number of predictors considered, although only the first two ones were actually relevant. Thus, if we did not know about the random mechanism (2.25) that generated $Y$, we would be tempted to believe that the more adequate models would be the ones with a larger number of predictors. On the contrary, $R^2_{\text{Adj}}$ only increases in the first two variables and then exhibits a mild decaying trend, indicating that, on average, the best choice for the number of predictors is actually close to $p = 2$. However, note that $R^2_{\text{Adj}}$ has a huge variability when $p$ approaches $n - 2$, a consequence of the explosive variance of $\hat{\sigma}^2$ in that degenerate case[36]. The experiment helps to visualize that $R^2_{\text{Adj}}$ is **more adequate** than the $R^2$ for evaluating the fit of a multiple linear regression[37].

An example of a simulated dataset considered in the experiment of Figure 2.12 is:

```r
# 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))
```

[36] This indicates that $R^2_{\text{Adj}}$ can act as a model selection device *up to a certain point*, as its effectiveness becomes too variable, too erratic, when the number of predictors is very high in comparison with $n$.

[37] Coincidentally, the experiment also serves as a reminder about the randomness of $R^2$ and $R^2_{\text{Adj}}$, a fact that is sometimes overlooked.

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

**Exercise 2.12.** Implement the simulation study behind Figure 2.12 in order to replicate it.

> ⚠ The $R^2_{\text{Adj}}$ 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, $R^2_{\text{Adj}} \leq 1$ and $R^2_{\text{Adj}}$ 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 $R^2_{\text{Adj}}$.

```
# 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
```

**Exercise 2.13.** For the previous example, construct more predictors (x3, x4, ...) that are independent from y. Then check that, when the predictors are added to the model, the $R^2_{\text{Adj}}$ decreases and the $R^2$ increases.

**Beware** of the $R^2$ and $R^2_{\text{Adj}}$ for model fits with **no intercept!** If the linear model is fitted with no intercept, the `summary` function **silently** returns a `'Multiple R-squared'` and an `'Adjusted R-squared'` that do *not* correspond (see this question in the R FAQ) with the seen expressions

$$R^2 = 1 - \frac{\text{SSE}}{\sum_{i=1}^n (Y_i - \bar{Y})^2}, \quad R^2_{\text{Adj}} = 1 - \frac{\text{SSE}}{\text{SST}} \times \frac{n-1}{n-p-1}.$$

In the case with no intercept, `summary` rather returns

$$R^2_0 := 1 - \frac{\text{SSE}}{\sum_{i=1}^n Y_i^2}, \quad R^2_{0,\text{Adj}} := 1 - \frac{\text{SSE}}{\sum_{i=1}^n Y_i^2} \times \frac{n-1}{n-p-1}.$$

The reason is perhaps shocking: if the model is **fitted without intercept** and neither the response nor the predictors are centered, then the **ANOVA decomposition does *not* hold**, in the sense that

$$\text{SST} \neq \text{SSR} + \text{SSE}.$$

The fact that the ANOVA decomposition does not hold for no-intercept models has serious consequences on the theory we built. In particular, the $R^2$ can be *negative* and $R^2 \neq r^2_{y\hat{y}}$, both deriving from the fact that $\sum_{i=1}^n \hat{\varepsilon}_i \neq 0$. Therefore, the $R^2$ cannot be regarded as the "proportion of variance explained" if the model fit is performed without intercept. The $R^2_0$ and $R^2_{0,\text{Adj}}$ versions are considered because they are the ones that arise from the "no-intercept ANOVA decomposition"

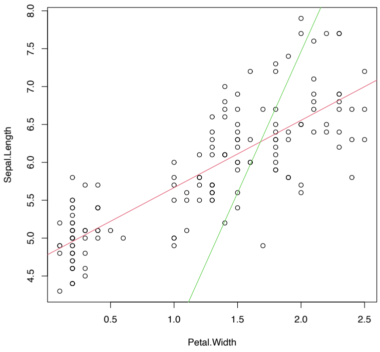
$$\text{SST}_0 = \text{SSR}_0 + \text{SSE}, \quad \text{SST}_0 := \sum_{i=1}^n Y_i^2, \quad \text{SSR}_0 := \sum_{i=1}^n \hat{Y}_i^2.$$

and therefore the $R^2_0$ is guaranteed to be a quantity in $[0,1]$. It would indeed be the proportion of variance explained if the predictors and the response were centered (i.e., if $\bar{Y} = 0$ and $\bar{X}_j = 0$, $j = 1, \ldots, p$).

The next chunk of code illustrates these concepts for the `iris` dataset.

```
# Model with intercept
mod1 <- lm(Sepal.Length ~ Petal.Width, data = iris)
mod1
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Petal.Width
##      4.7776       0.8886

# Model without intercept
mod0 <- lm(Sepal.Length ~ 0 + Petal.Width, data = iris)
mod0
```

```
##
## Call:
## lm(formula = Sepal.Length ~ 0 + Petal.Width, data = iris)
##
## Coefficients:
## Petal.Width
##       3.731

# Recall the different way of obtaining the estimators
X1 <- cbind(1, iris$Petal.Width)
X0 <- cbind(iris$Petal.Width) # No column of ones!
Y <- iris$Sepal.Length
betaHat1 <- solve(crossprod(X1)) %*% t(X1) %*% Y
betaHat0 <- solve(crossprod(X0)) %*% t(X0) %*% Y
betaHat1
##           [,1]
## [1,] 4.7776294
## [2,] 0.8885803
betaHat0
##          [,1]
## [1,] 3.731485

# Summaries: higher Rˆ2 for the model with no intercept!?
summary(mod1)
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Width, data = iris)
##
## 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
summary(mod0)
##
## Call:
## lm(formula = Sepal.Length ~ 0 + Petal.Width, data = iris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1556 -0.3917  1.0625  3.8537  5.0537
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## Petal.Width    3.732      0.150   24.87   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.609 on 149 degrees of freedom
## Multiple R-squared:  0.8058, Adjusted R-squared:  0.8045
## F-statistic: 618.4 on 1 and 149 DF,  p-value: < 2.2e-16

# Wait a moment... let's see the actual fit
plot(Sepal.Length ~ Petal.Width, data = iris)
abline(mod1, col = 2) # Obviously, much better
abline(mod0, col = 3)
```



```
# Compute the Rˆ2 manually for mod1
```

```
SSE1 <- sum(mod1$residuals^2)
SST1 <- sum((mod1$model$Sepal.Length - mean(mod1$model$Sepal.Length))^2)
1 - SSE1 / SST1
## [1] 0.6690277

# Compute the R^2 manually for mod0
SSE0 <- sum(mod0$residuals^2)
SST0 <- sum((mod0$model$Sepal.Length - mean(mod0$model$Sepal.Length))^2)
1 - SSE0 / SST0
## [1] -8.926872
# It is negative!

# Recall that the mean of the residuals is not zero!
mean(mod0$residuals)
## [1] 1.368038

# What summary really returns if there is no intercept
n <- nrow(iris)
p <- 1
R0 <- 1 - sum(mod0$residuals^2) / sum(mod0$model$Sepal.Length^2)
R0Adj <- 1 - sum(mod0$residuals^2) / sum(mod0$model$Sepal.Length^2) *
  (n - 1) / (n - p - 1)
R0
## [1] 0.8058497
R0Adj
## [1] 0.8045379

# What if we centered the data previously?
irisCen <- data.frame(scale(iris[, -5], center = TRUE, scale = FALSE))
modCen1 <- lm(Sepal.Length ~ Petal.Width, data = irisCen)
modCen0 <- lm(Sepal.Length ~ 0 + Petal.Width, data = irisCen)

# No problem, "correct" R^2
summary(modCen1)
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Width, data = irisCen)
##
## 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) -9.050e-16  3.903e-02     0.0        1
## Petal.Width  8.886e-01  5.137e-02    17.3   <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
summary(modCen0)
##
## Call:
## lm(formula = Sepal.Length ~ 0 + Petal.Width, data = irisCen)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.38822 -0.29358 -0.04393  0.26429  1.34521
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## Petal.Width   0.8886     0.0512   17.36   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4764 on 149 degrees of freedom
## Multiple R-squared:  0.669,  Adjusted R-squared:  0.6668
## F-statistic: 301.2 on 1 and 149 DF,  p-value: < 2.2e-16

# But only if we center predictor and response...
summary(lm(iris$Sepal.Length ~ 0 + irisCen$Petal.Width))
##
## Call:
## lm(formula = iris$Sepal.Length ~ 0 + irisCen$Petal.Width)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  4.455  5.550  5.799  6.108  7.189
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## irisCen$Petal.Width   0.8886     0.6322   1.406    0.162
##
## Residual standard error: 5.882 on 149 degrees of freedom
## Multiple R-squared:  0.01308,    Adjusted R-squared:  0.006461
## F-statistic: 1.975 on 1 and 149 DF,  p-value: 0.1619
```

**Exercise 2.14:** Let's play the **evil practitioner** and try to modify the $R_0^2$ returned by `summary` when the intercept is excluded. If we were really evil, we could use this knowledge to fool someone that is not aware of the difference between $R_0^2$ and $R^2$ into believing that any given model is incredibly good or bad in terms of the $R^2$!

a. For the previous example, display the $R_0^2$ as a function of a shift in mean of the predictor `Petal.Width`. What can you conclude? *Hint*: you may use `I()` for performing the shifting inside the model equation.
b. What shift on `Petal.Width` would be necessary to achieve an $R_0^2 \approx 0.95$? Is this shift unique?
c. Do the same but for a shift in the mean of the response `Sepal.Length`. What shift would be necessary to achieve an $R_0^2 \approx 0.50$? Is there a single shift?
d. Consider the multiple linear model `medv ~ nox + rm` for the `Boston` dataset. We want to tweak the $R_0^2$ to set it to any number in $[0, 1]$. Can we achieve this by only shifting `rm` *or* `medv` (only one of them)?
e. Explore systematically the $R_0^2$ for the shifting combinations of `rm` *and* `medv` and obtain a combination that delivers an $R_0^2 \approx 0.30$. *Hint*: use `filled.contour()` for visualizing the $R_0^2$ surface.

### 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
```

The model `modWine2` has the largest $R^2_{\text{Adj}}$ among the three studied. 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 through the `predict` function.

Furthermore, the **interpretation** of `modWine2` agrees with well-known facts in viticulture that make perfect sense. Precisely (answers 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 conclusions could be regarded as common folklore, keep in mind that this model

- allows us to **quantify the effect of each variable** on the wine quality and
- provides a precise way of **predicting** the quality of **future** vintages.

# 3
# Linear models II: model selection, extensions, and diagnostics

Given the response $Y$ and the predictors $X_1, \ldots, X_p$, 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* $X_1, \ldots, X_p$ for explaining $Y$. Among others, we will also see how to extend the linear model to account for non-linear relations between $Y$ and $X_1, \ldots, X_p$, how to check whether the assumptions of the model are realistic in practice, and how to incorporate dimension reduction within linear regression.

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

This case study is motivated by Harrison and Rubinfeld (1978), who proposed a *hedonic model*[1] for determining the willingness of house buyers to pay for clean air. 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 and through the dataset `MASS::Boston`.

The description of the related variables can be found in `?Boston` and Harrison and Rubinfeld (1978),[2] but we summarize here the most important ones as they appear in `Boston`. They are aggregated into five categories:

- *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` (whether there is 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 proportion of black population – low and high values of $B$ increase housing prices) and `lstat` (percent of lower status of the population).
- *Accessibility* variables: `dis` (distances to five employment centers)

[1] A hedonic model is a model that decomposes the price of an item into separate components that determine its price. For example, a hedonic model for the price of a house may decompose its price into the house characteristics, the kind of neighborhood, and the location.

[2] But be aware of the changes in units for `medv`, `black`, `lstat`, and `nox`.

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:

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

# # Alternatively
# data(Boston, package = "MASS")
```

A quick summary of the data is shown below:

```
summary(Boston)
##       crim                zn             indus            chas              nox               rm
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000   Min.   :0.3850   Min.   :3.561
##  1st Qu.: 0.08205   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000   1st Qu.:0.4490   1st Qu.:5.886
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000   Median :0.5380   Median :6.208
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917   Mean   :0.5547   Mean   :6.285
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000   3rd Qu.:0.6240   3rd Qu.:6.623
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000   Max.   :0.8710   Max.   :8.780
##       age              dis              rad              tax           ptratio          black            lstat
##  Min.   :  2.90   Min.   : 1.130   Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32   Min.   : 1.73
##  1st Qu.: 45.02   1st Qu.: 2.100   1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38   1st Qu.: 6.95
##  Median : 77.50   Median : 3.207   Median : 5.000   Median :330.0   Median :19.05   Median :391.44   Median :11.36
##  Mean   : 68.57   Mean   : 3.795   Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67   Mean   :12.65
##  3rd Qu.: 94.08   3rd Qu.: 5.188   3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23   3rd Qu.:16.95
##  Max.   :100.00   Max.   :12.127   Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90   Max.   :37.97
##       medv
##  Min.   : 5.00
##  1st Qu.:17.02
##  Median :21.20
##  Mean   :22.53
##  3rd Qu.:25.00
##  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 categories in which variables were classified.

```
car::scatterplotMatrix(~ crim + dis + medv + nox + rm, regLine = list(col = 2),
                       col = 1, smooth = list(col.smooth = 4, col.spread = 4),
                       data = Boston)
```

Note the peculiar distribution of crim, very concentrated at zero, and the asymmetry in medv, with a second mode associated with 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 (and recall that we have plotted just a subset of all the predictors).

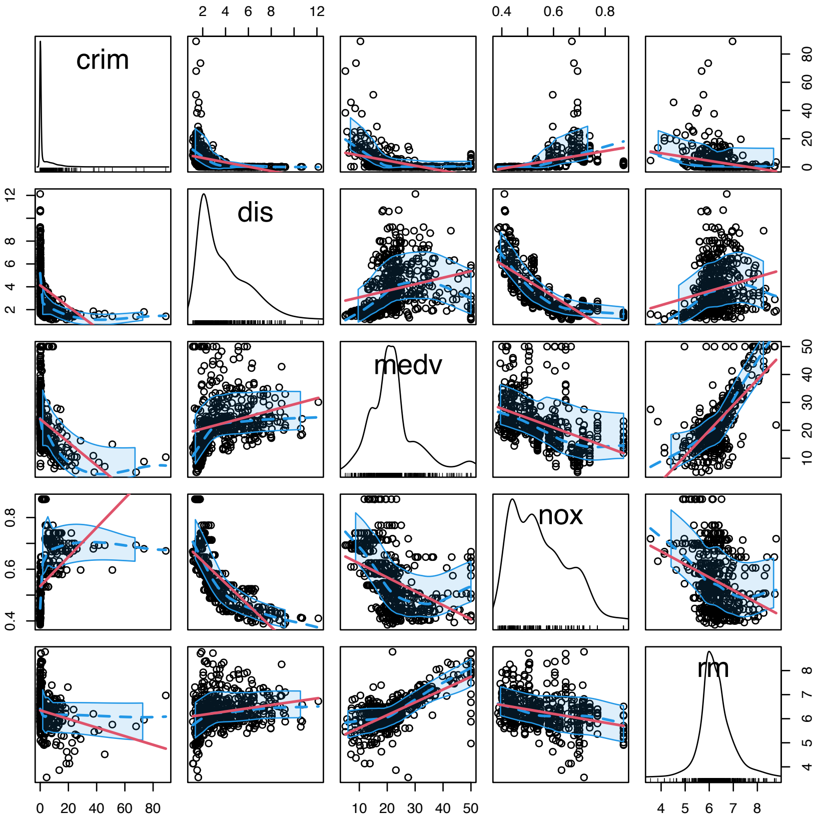


Figure 3.1: Scatterplot matrix for `crim`, `dis`, `medv`, `nox`, and `rm` from the `Boston` dataset. The diagonal panels show an estimate of the unknown pdf of each variable (see Section 6.1.2). The red and blue lines are linear and nonparametric (see Section 6.2) estimates of the regression functions for pairwise relations.

## 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 $n$:

$$p \leq n - 2.$$

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 about it, hence $\sigma^2$ cannot be estimated.[3] Therefore, we need at least $n = 3$ points, that is, $n \geq p + 2 = 3$.
- If $p = 2$, we need at least $n = 3$ points to uniquely fit a plane. But again this plane gives no information on the variation of the data about it and hence $\sigma^2$ cannot be estimated. Therefore, we need $n \geq p + 2 = 4$.

Another interpretation is the following:

[3] Applying its formula, we would obtain $\hat{\sigma}^2 = 0/0$ and so $\hat{\sigma}^2$ will not be defined.

> The fitting of a linear model with $p$ predictors involves the estimation of $p+2$ parameters $(\boldsymbol{\beta}, \sigma^2)$ from $n$ data points. The closer $p+2$ and $n$ are, the more variable the estimates $(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2)$ 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.

In the above discussion the degenerate case $p = n-1$ was excluded, as it gives a perfect and useless fit for which $\hat{\sigma}^2$ is not defined. However, $\hat{\boldsymbol{\beta}}$ is actually computable if $p = n-1$. The output of the next chunk of code clarifies the distinction between $p = n-1$ and $p = n-2$.

```r
# Data: n observations and p = n - 1 predictors
set.seed(123456)
n <- 5
p <- n - 1
df <- data.frame(y = rnorm(n), x = matrix(rnorm(n * p), nrow = n, ncol = p))

# Case p = n - 1 = 4: beta can be estimated, but sigma^2 cannot (hence, no
# inference can be performed since it requires the estimated sigma^2)
summary(lm(y ~ ., data = df))

# Case p = n - 2 = 3: both beta and sigma^2 can be estimated (hence, inference
# can be performed)
summary(lm(y ~ . - x.1, data = df))
```

The *degrees of freedom* $n-p-1$ quantify the increase in the variability of $(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2)$ when $p$ approaches $n-2$. 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 decrease, $t_{n-p-1;\alpha/2}$ increases, thus the intervals widen.
- $\hat{\sigma}^2 = \frac{1}{n-p-1}\sum_{i=1}^n \hat{\varepsilon}_i^2$ influences the $R^2$ and $R^2_{\text{Adj}}$. If no relevant variables are added to the model then $\sum_{i=1}^n \hat{\varepsilon}_i^2$ will not change substantially. However, the factor $\frac{1}{n-p-1}$ will increase as $p$ augments, inflating $\hat{\sigma}^2$ and its variance. This is exactly what happened in Figure 2.12.

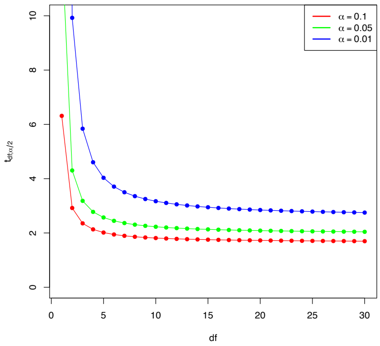


Figure 3.2: Effect of $\text{df} = n-p-1$ in $t_{\text{df};\alpha/2}$ for $\alpha = 0.10, 0.05, 0.01$.

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 sequentially compare multiple linear regression models with different predictors,[4] improving iteratively a performance measure through a ***greedy* search**.

Stepwise model selection typically uses as measure of performance 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 considered information criterion. Two common criteria are the *Bayesian Information Criterion* (BIC) and the *Akaike Information Criterion* (AIC). Both are based on **balancing the model fitness**

[4] Of course, with the same responses.

**and its complexity**:

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

where $\ell$(model) is the *log-likelihood of the model* (how well the model fits the data) and npar(model) is the number of parameters considered in the model (how complex the model is). In the case of a multiple linear regression model with $p$ predictors, npar(model) $= p + 2$. The AIC replaces the $\log(n)$ factor by a 2 in (3.1) so, compared with the BIC, it **penalizes less the more complex models.**[5] This is one of the reasons why BIC is preferred by some practitioners for performing model comparison[6]

The BIC and AIC can be computed 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|)
```

[5] Recall that $\log(n) > 2$ if $n \geq 8$.

[6] Also, because the BIC is *consistent*, see Section 3.2.2.

```
## (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 **exhaustive search** is the so-called *best subset selection* – its application to be seen in Section 4.4. The problem is that there are $2^p$ possible models to inspect![7] Fortunately, the `MASS::stepAIC` function helps us navigating this ocean of models by implementing stepwise model selection. Stepwise selection will **iteratively add predictors that decrease an information criterion and/or remove those that increase it**, depending on the mode of stepwise search that is performed.

`stepAIC` takes as input an initial model to be improved and has several variants. Let's see first how it works in its default mode using the already studied `wine` dataset.

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

We use as initial model the one featuring all the available predictors. The argument `k` of `stepAIC` stands for the factor post-multiplying npar(model) in (3.1). It defaults to 2, which gives the AIC. If we set it to `k = log(n)`, the function considers the BIC.

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

# With AIC
modAIC <- MASS::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
```

[7] Since we have to take $p$ binary decisions on whether to include or not each of the $p$ predictors.

```
## - AGST         1    3.4428 5.2486 -36.222

# The result is an lm object
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

# With BIC
modBIC <- MASS::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
```

An explanation of what `stepAIC` did for `modBIC`:

- At each step, `stepAIC` displayed information about the current value of the information criterion. For example, the BIC at the first step was `Step: AIC=-53.29` and then it improved to `Step: AIC=-56.55` in the second step.[8]
- The next model to move on was decided by exploring the information criteria of the different models resulting from adding or removing a predictor (depending on the `direction` argument, explained later). For example, in the first step the model arising from removing[9] `FrancePop` had a BIC equal to `-56.551`.

[8] The function always prints 'AIC', even if 'BIC' is employed.

[9] Note the `-` before `FrancePop`. Note also the convenient sorting of the BICs in an increasing form, so that the next best action always corresponds to the first row.

- The stepwise regression proceeded then by removing `FrancePop`, as it gave the lowest BIC. When repeating the previous exploration, it was found that removing `<none>` predictors was the best possible action.

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

The `direction` argument of `stepAIC` controls the mode of the stepwise model search:

- `"backward"`: **removes** predictors sequentially from the given model. It produces a sequence of models of *decreasing* complexity until attaining the optimal one.
- `"forward"`: **adds** predictors sequentially to the given model, using the available ones in the `data` argument of `lm`. It produces a sequence of models of *increasing* complexity until attaining the optimal one.
- `"both"` (default): a forward-backward search that, at each step, decides whether to include or exclude a predictor. Differently from the previous modes, a predictor that was excluded/included previously can be later included/excluded.

The next chunk of code clearly explains how to exploit the `direction` argument, and other options of `stepAIC`, with a modified version of the `wine` dataset. An important warning is that in order to use `direction = "forward"` or `direction = "both"`, **`scope` needs to be properly defined**. The practical advice to model selection is to run[10] several of these three search modes and retain the model with minimum BIC/AIC, being specially careful with the `scope` argument.

[10] If possible!

```
# Add an irrelevant predictor to the wine dataset
set.seed(123456)
wineNoise <- wine
n <- nrow(wineNoise)
wineNoise$noisePredictor <- rnorm(n)

# Backward selection: removes predictors sequentially from the given model

# Starting from the model with all the predictors
modAll <- lm(formula = Price ~ ., data = wineNoise)
MASS::stepAIC(modAll, direction = "backward", k = log(n))
## Start:  AIC=-50.13
## Price ~ Year + WinterRain + AGST + HarvestRain + Age + FrancePop +
##     noisePredictor
##
##
## Step:  AIC=-50.13
## Price ~ Year + WinterRain + AGST + HarvestRain + FrancePop +
##     noisePredictor
##
##                  Df Sum of Sq    RSS     AIC
## - FrancePop       1    0.0036 1.7977 -53.376
## - Year            1    0.0038 1.7979 -53.374
## - noisePredictor  1    0.0090 1.8032 -53.295
## <none>                        1.7941 -50.135
## - WinterRain      1    0.4598 2.2539 -47.271
## - HarvestRain     1    1.7666 3.5607 -34.923
## - AGST            1    3.3658 5.1599 -24.908
##
## Step:  AIC=-53.38
## Price ~ Year + WinterRain + AGST + HarvestRain + noisePredictor
##
##                  Df Sum of Sq    RSS     AIC
## - noisePredictor  1    0.0081 1.8058 -56.551
## <none>                        1.7977 -53.376
## - WinterRain      1    0.4771 2.2748 -50.317
## - Year            1    0.9162 2.7139 -45.552
## - HarvestRain     1    1.8449 3.6426 -37.606
## - AGST            1    3.4234 5.2212 -27.885
##
## 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
##
## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain,
##     data = wineNoise)
##
## Coefficients:
## (Intercept)         Year   WinterRain         AGST  HarvestRain
##   43.639042    -0.023848     0.001167     0.616392    -0.003861

# Starting from an intermediate model
modInter <- lm(formula = Price ~ noisePredictor + Year + AGST, data = wineNoise)
MASS::stepAIC(modInter, direction = "backward", k = log(n))
## Start:  AIC=-32.38
## Price ~ noisePredictor + Year + AGST
##
##                  Df Sum of Sq    RSS     AIC
## - noisePredictor  1    0.0146 5.0082 -35.601
## <none>                        4.9936 -32.384
## - Year            1    0.7522 5.7459 -31.891
## - AGST            1    3.2211 8.2147 -22.240
##
## Step:  AIC=-35.6
## Price ~ Year + AGST
##
##        Df Sum of Sq    RSS     AIC
## <none>              5.0082 -35.601
## - Year  1    0.7966 5.8049 -34.911
## - AGST  1    3.2426 8.2509 -25.417
##
## Call:
## lm(formula = Price ~ Year + AGST, data = wineNoise)
##
## Coefficients:
## (Intercept)         Year         AGST
##    41.49441     -0.02221      0.56067
# Recall that other predictors not included in modInter are not explored
# during the search (so the relevant predictor HarvestRain is not added)

# Forward selection: adds predictors sequentially from the given model

# Starting from the model with no predictors, only intercept (denoted as ~ 1)
modZero <- lm(formula = Price ~ 1, data = wineNoise)
MASS::stepAIC(modZero, direction = "forward",
              scope = list(lower = modZero, upper = modAll), k = log(n))
```

```
## Start:  AIC=-22.28
## Price ~ 1
##
##                  Df Sum of Sq     RSS     AIC
## + AGST            1    4.6655  5.8049 -34.911
## + HarvestRain     1    2.6933  7.7770 -27.014
## + FrancePop       1    2.4231  8.0472 -26.092
## + Year            1    2.2195  8.2509 -25.417
## + Age             1    2.2195  8.2509 -25.417
## <none>                        10.4703 -22.281
## + WinterRain      1    0.1905 10.2798 -19.481
## + noisePredictor  1    0.1761 10.2942 -19.443
##
## Step:  AIC=-34.91
## Price ~ AGST
##
##                  Df Sum of Sq    RSS     AIC
## + HarvestRain     1   2.50659 3.2983 -46.878
## + WinterRain      1   1.42392 4.3809 -39.214
## + FrancePop       1   0.90263 4.9022 -36.178
## + Year            1   0.79662 5.0082 -35.601
## + Age             1   0.79662 5.0082 -35.601
## <none>                        5.8049 -34.911
## + noisePredictor  1   0.05900 5.7459 -31.891
##
## Step:  AIC=-46.88
## Price ~ AGST + HarvestRain
##
##                  Df Sum of Sq    RSS     AIC
## + FrancePop       1   1.03572 2.2625 -53.759
## + Year            1   1.01159 2.2867 -53.473
## + Age             1   1.01159 2.2867 -53.473
## + WinterRain      1   0.58356 2.7147 -48.840
## <none>                        3.2983 -46.878
## + noisePredictor  1   0.06084 3.2374 -44.085
##
## Step:  AIC=-53.76
## Price ~ AGST + HarvestRain + FrancePop
##
##                  Df Sum of Sq    RSS     AIC
## + WinterRain      1   0.45456 1.8080 -56.519
## <none>                        2.2625 -53.759
## + noisePredictor  1   0.00829 2.2542 -50.562
## + Year            1   0.00085 2.2617 -50.473
## + Age             1   0.00085 2.2617 -50.473
##
## Step:  AIC=-56.52
## Price ~ AGST + HarvestRain + FrancePop + WinterRain
##
##                  Df Sum of Sq    RSS     AIC
## <none>                        1.8080 -56.519
## + noisePredictor  1 0.0100635 1.7979 -53.374
## + Year            1 0.0048039 1.8032 -53.295
## + Age             1 0.0048039 1.8032 -53.295
##
## Call:
## lm(formula = Price ~ AGST + HarvestRain + FrancePop + WinterRain,
##     data = wineNoise)
##
## Coefficients:
## (Intercept)         AGST  HarvestRain    FrancePop   WinterRain
##  -5.945e-01    6.127e-01   -3.804e-03   -5.189e-05    1.136e-03
# It is very important to set the scope argument adequately when doing forward
# search! In the scope you have to define the "minimum" (lower) and "maximum"
# (upper) models that contain the set of explorable models. If not provided,
# the maximum model will be taken as the passed starting model (in this case
# modZero) and stepAIC will not do any search
# Starting from an intermediate model
MASS::stepAIC(modInter, direction = "forward",
              scope = list(lower = modZero, upper = modAll), k = log(n))
## Start:  AIC=-32.38
## Price ~ noisePredictor + Year + AGST
## 
##                Df Sum of Sq    RSS     AIC
## + HarvestRain  1   2.71878 2.2748 -50.317
## + WinterRain   1   1.35102 3.6426 -37.606
## <none>                      4.9936 -32.384
## + FrancePop    1   0.24004 4.7536 -30.418
## 
## Step:  AIC=-50.32
## Price ~ noisePredictor + Year + AGST + HarvestRain
## 
##               Df Sum of Sq    RSS     AIC
## + WinterRain  1   0.47710 1.7977 -53.376
## <none>                     2.2748 -50.317
## + FrancePop   1   0.02094 2.2539 -47.271
## 
## Step:  AIC=-53.38
## Price ~ noisePredictor + Year + AGST + HarvestRain + WinterRain
## 
##              Df Sum of Sq    RSS     AIC
## <none>                    1.7977 -53.376
## + FrancePop  1 0.0036037 1.7941 -50.135
## 
## Call:
## lm(formula = Price ~ noisePredictor + Year + AGST + HarvestRain + 
##     WinterRain, data = wineNoise)
## 
## Coefficients:
##    (Intercept)  noisePredictor            Year            AGST     HarvestRain      WinterRain  
##      44.096639       -0.019617       -0.024126        0.620522       -0.003840        0.001211
# Recall that predictors included in modInter are not dropped during the
# search (so the irrelevant noisePredictor is kept)

# Both selection: useful if starting from an intermediate model

# Removes the problems associated with "backward" and "forward" searches done
# from intermediate models
MASS::stepAIC(modInter, direction = "both",
              scope = list(lower = modZero, upper = modAll), k = log(n))
## Start:  AIC=-32.38
## Price ~ noisePredictor + Year + AGST
## 
##                   Df Sum of Sq    RSS     AIC
## + HarvestRain     1    2.7188 2.2748 -50.317
## + WinterRain      1    1.3510 3.6426 -37.606
## - noisePredictor  1    0.0146 5.0082 -35.601
## <none>                        4.9936 -32.384
## - Year            1    0.7522 5.7459 -31.891
## + FrancePop       1    0.2400 4.7536 -30.418
## - AGST            1    3.2211 8.2147 -22.240
## 
## Step:  AIC=-50.32
## Price ~ noisePredictor + Year + AGST + HarvestRain
## 
##                   Df Sum of Sq    RSS     AIC
## - noisePredictor  1   0.01182 2.2867 -53.473
## + WinterRain      1   0.47710 1.7977 -53.376
## <none>                        2.2748 -50.317
## + FrancePop       1   0.02094 2.2539 -47.271
## - Year            1   0.96258 3.2374 -44.085
## - HarvestRain     1   2.71878 4.9936 -32.384
## - AGST            1   2.94647 5.2213 -31.180
## 
## Step:  AIC=-53.47
## Price ~ Year + AGST + HarvestRain
##
##                  Df Sum of Sq    RSS     AIC
## + WinterRain      1   0.48087 1.8058 -56.551
## <none>                        2.2867 -53.473
## + FrancePop       1   0.02497 2.2617 -50.473
## + noisePredictor  1   0.01182 2.2748 -50.317
## - Year            1   1.01159 3.2983 -46.878
## - HarvestRain     1   2.72157 5.0082 -35.601
## - AGST            1   2.96500 5.2517 -34.319
##
## Step:  AIC=-56.55
## Price ~ Year + AGST + HarvestRain + WinterRain
##
##                  Df Sum of Sq    RSS     AIC
## <none>                        1.8058 -56.551
## - WinterRain      1    0.4809 2.2867 -53.473
## + noisePredictor  1    0.0081 1.7977 -53.376
## + FrancePop       1    0.0026 1.8032 -53.295
## - Year            1    0.9089 2.7147 -48.840
## - HarvestRain     1    1.8760 3.6818 -40.612
## - AGST            1    3.4428 5.2486 -31.039
##
## Call:
## lm(formula = Price ~ Year + AGST + HarvestRain + WinterRain,
##     data = wineNoise)
##
## Coefficients:
## (Intercept)         Year         AGST  HarvestRain   WinterRain
##   43.639042    -0.023848     0.616392    -0.003861     0.001167
# It is very important as well to correctly define the scope, because "both"
# resorts to "forward" (as well as to "backward")

# Using the defaults from the full model essentially does backward selection,
# but allowing predictors that were removed to enter again at later steps
MASS::stepAIC(modAll, direction = "both", k = log(n))
## Start:  AIC=-50.13
## Price ~ Year + WinterRain + AGST + HarvestRain + Age + FrancePop +
##     noisePredictor
##
##
## Step:  AIC=-50.13
## Price ~ Year + WinterRain + AGST + HarvestRain + FrancePop +
##     noisePredictor
##
##                  Df Sum of Sq    RSS     AIC
## - FrancePop       1    0.0036 1.7977 -53.376
## - Year            1    0.0038 1.7979 -53.374
## - noisePredictor  1    0.0090 1.8032 -53.295
## <none>                        1.7941 -50.135
## - WinterRain      1    0.4598 2.2539 -47.271
## - HarvestRain     1    1.7666 3.5607 -34.923
## - AGST            1    3.3658 5.1599 -24.908
##
## Step:  AIC=-53.38
## Price ~ Year + WinterRain + AGST + HarvestRain + noisePredictor
##
##                  Df Sum of Sq    RSS     AIC
## - noisePredictor  1    0.0081 1.8058 -56.551
## <none>                        1.7977 -53.376
## - WinterRain      1    0.4771 2.2748 -50.317
## + FrancePop       1    0.0036 1.7941 -50.135
## - Year            1    0.9162 2.7139 -45.552
## - HarvestRain     1    1.8449 3.6426 -37.606
## - AGST            1    3.4234 5.2212 -27.885
##
## 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
## + noisePredictor  1    0.0081 1.7977 -53.376
## + FrancePop       1    0.0026 1.8032 -53.295
## - Year            1    0.9089 2.7147 -48.840
## - HarvestRain     1    1.8760 3.6818 -40.612
## - AGST            1    3.4428 5.2486 -31.039
##
## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain,
##     data = wineNoise)
##
## Coefficients:
## (Intercept)         Year   WinterRain         AGST  HarvestRain
##   43.639042    -0.023848     0.001167     0.616392    -0.003861

# Omit lengthy outputs
MASS::stepAIC(modAll, direction = "both", trace = 0,
              scope = list(lower = modZero, upper = modAll), k = log(n))
##
## Call:
## lm(formula = Price ~ Year + WinterRain + AGST + HarvestRain,
##     data = wineNoise)
##
## Coefficients:
## (Intercept)         Year   WinterRain         AGST  HarvestRain
##   43.639042    -0.023848     0.001167     0.616392    -0.003861
```

**Exercise 3.1.** Run the previous stepwise selections for the `Boston` dataset, with the aim of clearly understanding the different search directions. Specifically:

a. Do a `"forward"` stepwise fit starting from `medv ~ 1`.
b. Do a `"forward"` stepwise fit starting from `medv ~ crim + lstat + age`.
c. Do a `"both"` stepwise fit starting from `medv ~ crim + lstat + age`.
d. Do a `"both"` stepwise fit starting from `medv ~ .`.
e. Do a `"backward"` stepwise fit starting from `medv ~ .`.

We conclude with a couple of quirks of `stepAIC` to be aware of.

⚠ `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 D for possible alternatives to deal with missing data.

`stepAIC` and friends (`addterm` and `dropterm`) compute a **slightly different version of the BIC/AIC** than the `BIC`/`AIC` functions. Precisely, the BIC/AIC they report come from the `extractAIC` function, which **differs in an additive constant** from the output of `BIC`/`AIC`. This is not relevant for model comparison, since shifting by a common constant the BIC/AIC does not change the lower-to-higher BIC/AIC ordering of models. However, it is important to be aware of this fact in order to do *not* compare directly the output of `stepAIC` with the one of `BIC`/`AIC`. The additive constant (included in `BIC`/`AIC` but not in `extractAIC`) is $n(\log(2\pi) + 1) + \log(n)$ for the BIC and $n(\log(2\pi) + 1) + 2$ for the AIC. The discrepancy arises from simplifying the computation of the BIC/AIC for linear models and from counting $\hat{\sigma}^2$ as an estimated parameter.

The following chunk of code illustrates the relation of the AIC reported in `stepsAIC`, the output of `extractAIC`, and the BIC/AIC reported by `BIC`/`AIC`.

```
# Same BICs, different scale
n <- nobs(modBIC)
extractAIC(modBIC, k = log(n))[2]
## [1] -56.55135
BIC(modBIC)
## [1] 23.36717
# Observe that MASS::stepAIC(mod, k = log(nrow(wine))) returned as final BIC
# the one given by extractAIC(), not by BIC()! But both are equivalent, they
# just differ in a constant shift

# Same AICs, different scale
extractAIC(modAIC, k = 2)[2]
## [1] -63.03053
AIC(modAIC)
## [1] 15.59215

# The additive constant: BIC() includes it but extractAIC() does not
BIC(modBIC) - extractAIC(modBIC, k = log(n))[2]
## [1] 79.91852
n * (log(2 * pi) + 1) + log(n)
## [1] 79.91852

# Same for the AIC
AIC(modAIC) - extractAIC(modAIC)[2]
## [1] 78.62268
n * (log(2 * pi) + 1) + 2
## [1] 78.62268
```

### 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 effects. However, the non-significant coefficients are only adding noise to the prediction 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 performing a forward-backward search starting from the full model. This gives us a candidate *best model*.

```
# Best BIC and AIC models
modBIC <- MASS::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 <- MASS::stepAIC(modHouse, trace = 0, k = 2)

# Comparison: same fits
car::compareCoefs(modBIC, modAIC)
## Calls:
## 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)
##
##              Model 1  Model 2
## (Intercept)    36.34    36.34
## SE              5.07     5.07
##
## crim         -0.1084  -0.1084
## SE            0.0328   0.0328
##
## zn            0.0458   0.0458
## SE            0.0135   0.0135
##
## chas           2.719    2.719
## SE             0.854    0.854
##
## nox           -17.38   -17.38
## SE              3.54     3.54
##
## rm             3.802    3.802
## SE             0.406    0.406
##
## dis           -1.493   -1.493
## SE             0.186    0.186
```

```
##
## rad           0.2996   0.2996
## SE            0.0634   0.0634
##
## tax         -0.01178 -0.01178
## SE           0.00337  0.00337
##
## ptratio       -0.947   -0.947
## SE             0.129    0.129
##
## black        0.00929  0.00929
## SE           0.00267  0.00267
##
## lstat        -0.5226  -0.5226
## SE            0.0474   0.0474
##
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.

Using `modBIC`, we can quantify the influence of the predictor variables on the housing prices (Q1) and we can conclude that, in the final model (Q2) and with significance level $\alpha = 0.05$:

- `zn`, `chas`, `rm`, `rad`, and `black` have a **significantly positive** influence on `medv`;
- `crim`, `nox`, `dis`, `tax`, `ptratio`, and `lstat` have a **significantly negative** influence on `medv`.

**Exercise 3.2.** The functions `MASS::addterm` and `MASS::dropterm` allow adding and removing all individual predictors to a given model, and inform the BICs / AICs of the possible combinations. Check that:

a. `modBIC` cannot be improved in terms of BIC by removing predictors. Use `dropterm(modBIC, k = log(nobs(modBIC)))` for that.
b. `modBIC` cannot be improved in terms of BIC by adding predictors. Use `addterm(modBIC, scope = lm(medv ~ ., data = Boston), k = log(nobs(modBIC)))` for that.

For the second point, recall that `scope` must specify the maximal model *or formula*. However, be careful because if using the formula approach, `addterm(modBIC, scope = medv ~ ., k = log(nobs(modBIC)))` will understand that `.` refers to all the predictors *in* modBIC, not in the `Boston` dataset, and will return an error. Calling `addterm(modBIC, scope = medv ~ . + indus + age, k = log(nobs(modBIC)))` gives the required result in terms of a formula, at the expense of manually adding the remaining predictors.

### 3.2.2 *Consistency in model selection*

A caveat on the use of the BIC/AIC is that both criteria are constructed assuming that the sample size $n$ is much larger than the number of parameters in the model $(p+2)$, that is, assuming that $n \gg p+2$. Therefore, they will work reasonably well if $n \gg p+2$ but, when this is not true, they enter in "overfitting mode" and start favoring unrealistic complex models. An illustration of this phenomenon is shown in Figure 3.3, which is the BIC/AIC version of Figure 2.12 and corresponds to the simulation study of Section 2.6. The BIC/AIC curves tend to have local minima close to $p = 2$ and then increase. But, when $p+2$ gets close to $n$, they quickly drop below their values at $p = 2$.

In particular, Figure 3.3 gives the following insights:

- The steeper BIC curves are a consequence of the higher penalization that BIC introduces on model complexity with respect to AIC. Therefore, the local minima of the BIC curves are better identified than those of the flatter AIC curves.
- The BIC resists better the overfitting than the AIC. The average BIC curve, $\overline{\mathrm{BIC}}(p)$, starts giving smaller values than $\overline{\mathrm{BIC}}(2) \approx 1023.76$ when $p = 198$ ($\overline{\mathrm{BIC}}(198) \approx 788.30$). Overfitting appears

earlier in the AIC, and $\overline{\text{AIC}}(2) \approx 1010.57$ is surpassed for $p \geq 155$ ($\overline{\text{AIC}}(155) \approx 1009.57$).

- The variabilities of both AIC and BIC curves are comparable, with no significant differences between them.

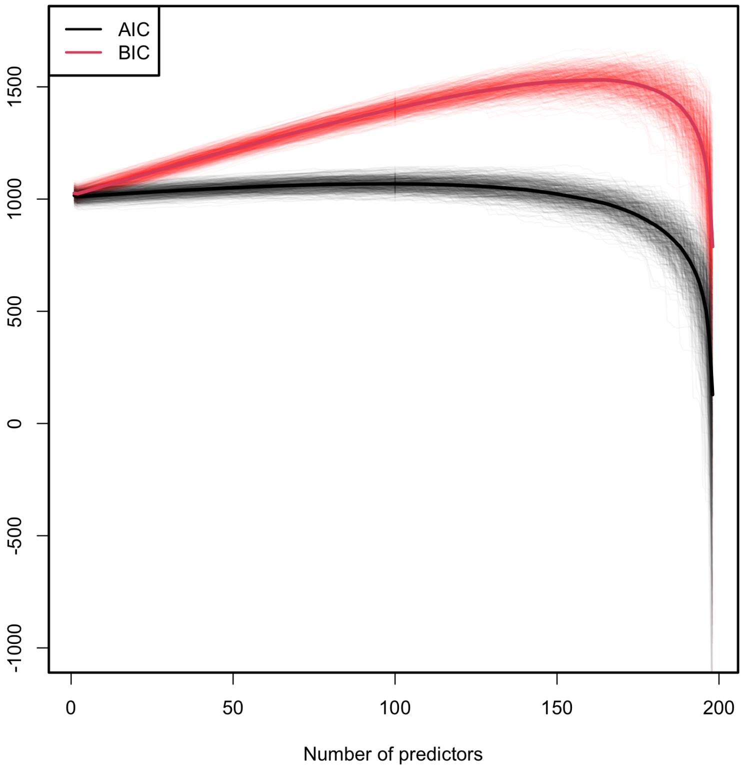


Figure 3.3: Comparison of BIC and AIC on the model (2.26) fitted with data generated by (2.25). The number of predictors $p$ ranges from 1 to 198, with only the *first two* predictors being significant. The $M = 500$ curves for each color arise from $M$ simulated datasets of sample size $n = 200$. The thicker curves are the mean of each color's curves.

Another big difference between the AIC and BIC, which is indeed behind the behaviors seen in Figure 3.3, is the ***consistency* of the BIC** in performing model selection. In simple terms, "consistency" means that, if enough data is provided, the BIC is guaranteed to identify the true data-generating model among a list of candidate models if the true model is included in the list. Mathematically, it means that, given a collection of models $M_0, M_1, \ldots, M_m$, where $M_0$ is the generating model of a sample of size $n$, then

$$\mathbb{P}\left[\arg\min_{k=0,\ldots,m} \text{BIC}(\hat{M}_k) = 0\right] \to 1 \quad \text{as} \quad n \to \infty, \tag{3.2}$$

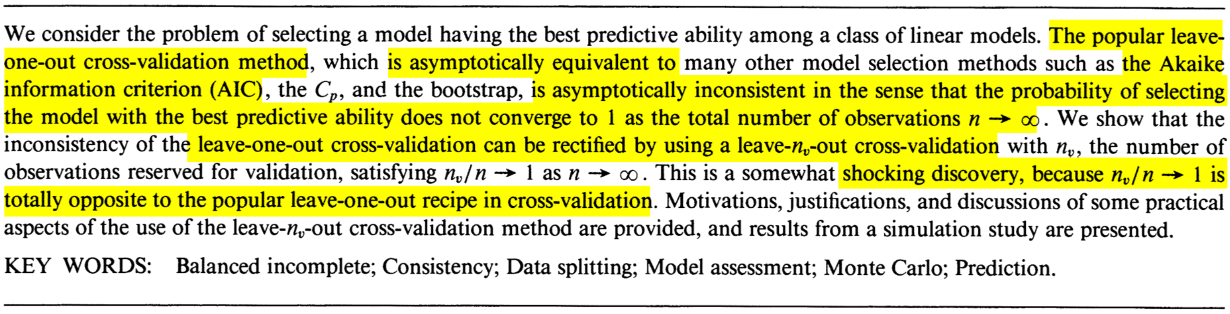
where $\hat{M}_k$ represents the $M_k$ model fitted with a sample of size $n$ generated from $M_0$.[11] Note that, despite being a nice theoretical result, its application may be unrealistic in practice, as most likely the true model is nonlinear or not present in the list of candidate models we examine.

The **AIC is inconsistent**, in the sense that (3.2) is not true if BIC is replaced by AIC. Indeed, this result can be seen as a consequence of the **asymptotic equivalence of model selection by AIC and leave-one-out cross-validation**,[12] and the inconsistency of the latter.

[11] The specifics of this result for the linear model are given in Schwarz (1978).

[12] Cross-validation and its different variants are formally addressed at the end of Section 3.6.2.

This is beautifully described in the paper by Shao (1993), whose abstract is given in Figure 3.4. The paper made a shocking discovery in terms of what is the required modification to induce consistency in model selection by cross-validation.



We consider the problem of selecting a model having the best predictive ability among a class of linear models. The popular leave-one-out cross-validation method, which is asymptotically equivalent to many other model selection methods such as the Akaike information criterion (AIC), the $C_p$, and the bootstrap, is asymptotically inconsistent in the sense that the probability of selecting the model with the best predictive ability does not converge to 1 as the total number of observations $n \to \infty$. We show that the inconsistency of the leave-one-out cross-validation can be rectified by using a leave-$n_v$-out cross-validation with $n_v$, the number of observations reserved for validation, satisfying $n_v/n \to 1$ as $n \to \infty$. This is a somewhat shocking discovery, because $n_v/n \to 1$ is totally opposite to the popular leave-one-out recipe in cross-validation. Motivations, justifications, and discussions of some practical aspects of the use of the leave-$n_v$-out cross-validation method are provided, and results from a simulation study are presented.

KEY WORDS: Balanced incomplete; Consistency; Data splitting; Model assessment; Monte Carlo; Prediction.

Figure 3.4: The abstract of Jun Shao's *Linear model selection by cross-validation* (Shao, 1993).

The Leave-One-Out Cross-Validation (LOOCV) error in a fitted linear model is defined as

$$\mathrm{LOOCV} := \frac{1}{n}\sum_{i=1}^n (Y_i - \hat{Y}_{i,-i})^2, \tag{3.3}$$

where $\hat{Y}_{i,-i}$ is the prediction of the $i$-th observation in a linear model that has been fitted without the $i$-th datum $(X_{i1}, \ldots, X_{ip}, Y_i)$. That is, $\hat{Y}_{i,-i} = \hat{\beta}_{0,-i} + \hat{\beta}_{1,-i}X_{i1} + \cdots + \hat{\beta}_{p,-i}X_{ip}$, where $\hat{\boldsymbol{\beta}}_{-i}$ are the estimated coefficients from the sample $\{(X_{j1}, \ldots, X_{jp}, Y_j)\}_{j=1, j\neq i}^n$. Model selection based on LOOCV chooses the model with minimum error (3.3) within a list of candidate models. Note that the computation of LOOCV requires, in principle, to fit $n$ separate linear models, predict, and then aggregate. However, an algebraic shortcut based on (4.25) allows us to compute (3.3) using a single linear fit.

We carry out a small simulation study in order to illustrate the consistency property (3.2) of BIC and the inconsistency of model selection by AIC and LOOCV. For that, we consider the linear model

$$\begin{aligned} Y &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \varepsilon, \\ \boldsymbol{\beta} &= (0.5, 1, 1, 0, 0, 0)^\top, \end{aligned} \tag{3.4}$$

where $X_j \sim \mathcal{N}(0,1)$ for $j = 1, \ldots, 5$ (independently) and $\varepsilon \sim \mathcal{N}(0,1)$. Only the first two predictors are relevant, the last three are "garbage" predictors. For a given sample, model selection is performed by selecting among the $2^5$ possible fitted models the ones with minimum AIC, BIC, and LOOCV. The experiment is repeated $M = 500$ times for sample sizes $n = 2^\ell$, $\ell = 3, \ldots, 12$, and the estimated probability of selecting the correct model (the one only involving $X_1$ and $X_2$) is displayed in Figure 3.5. The figure evidences empirically several interesting results:

- **LOOCV and AIC are *asymptotically* equivalent**. For large $n$, they tend to select the same model and hence their estimated probabilities of selecting the true model are almost equal. For small $n$, there are significant differences between them.

- The **BIC is consistent** in selecting the true model, and its probability of doing so quickly approaches 1, as anticipated by (3.2).
- The **AIC and LOOCV are inconsistent** in selecting the true model. Despite the sample size $n$ doubling at each step, their probability of recovering the true model gets stuck at about 0.60.
- Even for moderate $n$'s, the probability of recovering the true model by BIC quickly outperforms those of AIC/LOOCV.

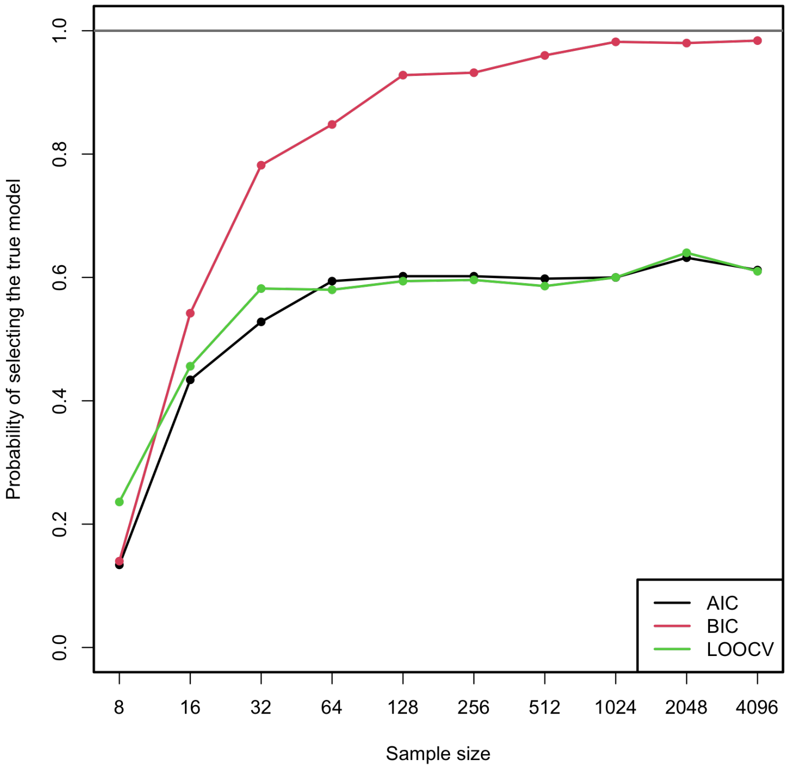


Figure 3.5: Estimation of the probability of selecting the correct model by minimizing the AIC, BIC, and LOOCV criteria, done for an exhaustive search with $p = 5$ predictors. The correct model contained two predictors.
The probability was estimated with $M = 500$ Monte Carlo runs. The horizontal axis is in logarithmic scale. The estimated proportion of true model recoveries with BIC for $n = 4096$ is 0.984.

**Exercise 3.3.** Implement the simulation study behind Figure 3.5:

1. Sample from (3.4), but take $p = 4$.
2. Fit the $2^p$ possible models.
3. Select the best model according to the AIC and BIC.
4. Repeat Steps 1–3 $M = 100$ times. Estimate by Monte Carlo the probability of selecting the true model.
5. Move $n$ in a range from 10 to 200.

Once you have a working solution, increase $(p, n, M)$ to approach the settings in Figure 3.5 (or go beyond!).

**Exercise 3.4.** Modify the Step 3 of the previous exercise to:

a. Select the best model according to the $R^2_{\text{Adj}}$ and investigate its consistency in model selection.
b. Add the LOOCV criterion in order to fully replicate Figure 3.5. *Hint*: you may want to adapt (4.25) to your needs in order to reduce computation time.

**Exercise 3.5.** Investigate what happens with the probability of selecting the true model using BIC and AIC if the exhaustive search is replaced by a stepwise selection. Precisely, do:

1. Sample from (3.4), but take $p = 10$ (pad $\boldsymbol{\beta}$ with zeros).
2. Run a forward-backward stepwise search, both for the AIC and BIC.
3. Repeat Steps 1–2 $M = 100$ times. Estimate by Monte Carlo the probability of selecting the true model.
4. Move $n$ in a range from 20 to 200.

Once you have a working solution, increase $(n, M)$ to approach the settings in Figure 3.5 (or go beyond!).

Shao (1993)'s modification on how to make consistent the model selection by cross-validation is shocking. As in LOOCV, one has to split the sample of size $n$ into two subsets: a *training set*, of size $n_t$, and a *validation set*, of size $n_v$. Of course, $n = n_t + n_v$. However, totally opposite to LOOCV, where $n_t = n - 1$ and $n_v = 1$, the modification required is to **choose $n_v$ asymptotically equal to $n$**, in the sense that $n_v/n \to 1$ as $n \to \infty$. An example would be to take $n_v = n - 20$ and $n_t = 20$. But $n_v = n/2$ and $n_t = n/2$, a typical choice, would not make the selection consistent! Shao (1993)'s result gives a great insight: the **difficulty in model selection lies in the *comparison* of models, not in their estimation**, and the sample information has to be disproportionately allocated to the comparison task to achieve a consistent model selection device.

**Exercise 3.6.** Verify Shao (1993)'s result by constructing the analogous of Figure 3.5 for the following choices of $(n_t, n_v)$:

a. $n_t = n - 1$, $n_v = 1$ (LOOCV).
b. $n_t = n/2$, $n_v = n/2$.
c. $n_t = n/4$, $n_v = (3/4)n$.
d. $n_t = 5p$, $n_v = n - 5p$.

Use first $p = 4$ predictors, $M = 100$ Monte Carlo runs, and move $n$ in a range from 20 to 200. Then increase the settings to approach those of Figure 3.5.

## 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 are also known as *categorical* variables or, in R's terminology, *factors*, and are very common in many fields, such as 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 encoded as

$$D = \begin{cases} 1, & \text{if } C = b, \\ 0, & \text{if } C = a. \end{cases}$$

$D$ is a *dummy variable*: it codifies with zeros and ones the two possible levels of the categorical variable. An example of $C$ is *smoker*, which has levels *yes* and *no*. The dummy variable associated is $D = 1$ if a person smokes and $D = 0$ if a person is non-smoker.

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, codify the rest as departures from it**.

The previous interpretation translates easily to the linear model. Assume that the dummy variable $D$ is available together with other predictors $X_1, \ldots, X_p$. Then:

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

The coefficient associated with $D$ is easily **interpretable**: $\beta_{p+1}$ is the increment in the mean of $Y$ associated with 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 with changing the level of the categorical variable from $a$ to $b$.

R does the dummification automatically, translating 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

$$D_2 = \begin{cases} 1, & \text{if } C = c, \\ 0, & \text{if } C \neq c. \end{cases}$$

Therefore, we can represent the levels of $C$ as in the following table.

| $C$ | $D_1$ | $D_2$ |
|---|---|---|
| $a$ | 0 | 0 |
| $b$ | 1 | 0 |
| $c$ | 0 | 1 |

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

In general, if we have a categorical variable $C$ with $J$ levels, then the previous process is iterated and the number of dummy variables required to encode $C$ is $J - 1$ [13]. Again, R does the dummifi-

[13] To account for the $J - 1$ possible departures from a reference level.

cation automatically if it detects that a factor variable is present in the regression model.

Let's see an example with the famous `iris` dataset.

```
# iris dataset -- factors in the last column
summary(iris)
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
##  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

# 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
## Speciessetosa     0.2488500  1.19827390
## Speciesvirginica -0.5351144 -0.06475727
# The coefficients of Speciessetosa and Speciesvirginica are
# significantly positive and negative, respectively

# 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 of the same categorical variable, say $D_2$, are significant. For example, this happens in the example above at level $\alpha = 0.01$. Then, in the considered model, the level associated with $D_1$ does not add relevant information for explaining $Y$ with respect to the reference level.

**Do not codify a categorical variable as a discrete variable**. This constitutes a major methodological failure 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
data(Boston, package = "MASS")

# 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 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 7.87 ...
##  $ chas   : int  0 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 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 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 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 dummified)

# 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 with `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 with `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 only because the form it assumes for the regression function $m$ is linear, 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 a transformed predictor $\tilde{X}$ as a replacement of the original predictor $X$. Then, for the above examples, rather than working with the sample $\{(X_i, Y_i)\}_{i=1}^n$, we consider the transformed sample $\{(\tilde{X}_i, Y_i)\}_{i=1}^n$ with:

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

An example of this simple but powerful trick is given as follows. The left panel of Figure 3.6 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 3.6, we can see that the relation of y and x^2 is now linear!

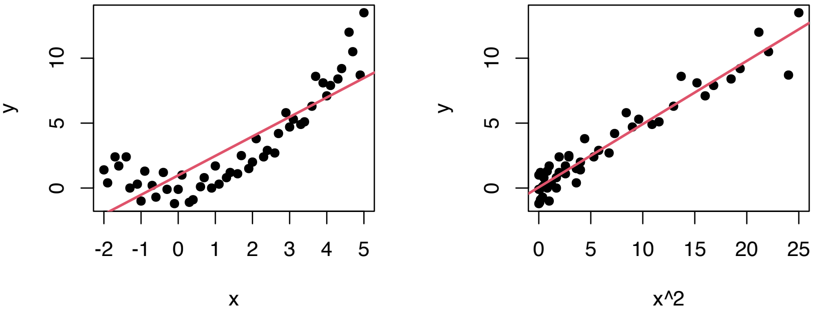


Figure 3.6: Left: quadratic pattern when plotting $Y$ against $X$. Right: linearized pattern when plotting $Y$ against $X^2$. In red, the fitted regression line.

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

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 linearize the relation between response and the predictor is a kind of art which requires good eyeballing. A first approach is to try with one of the usual transformations, displayed in Figure 3.7, depending on the pattern of the data. Figure 3.4.1 illustrates how to choose an adequate transformation for linearizing certain nonlinear data patterns.

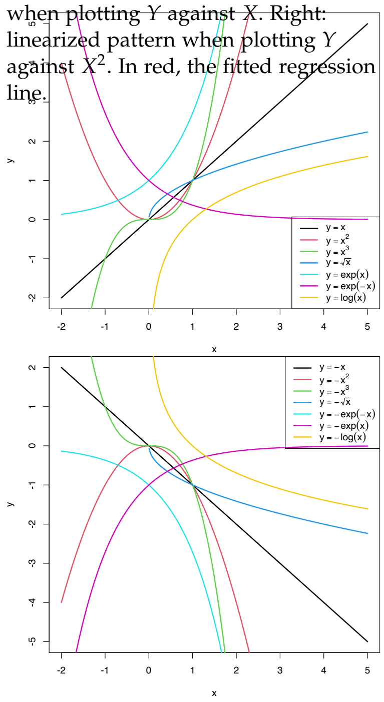


Figure 3.7: Some common nonlinear transformations and their negative counterparts. Recall the domain of definition of each transformation.

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 3.7 with the transformations to compare it with the data pattern, choose the most similar curve, and apply the corresponding function with **positive sign** (for simpler interpretation).

Let's see how to transform the predictor and perform the regressions behind Figure 3.6.

```
# 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 significant
```

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 it 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. For example, use `I((x - 1)^3 - log(3 * x))` to apply the transformation `(x - 1)^3 - log(3 * x)`.

**Exercise 3.7.** Load the dataset `assumptions.RData`. We are going to work with the regressions `y2 ~ x2`, `y3 ~ x3`, `y8 ~ x8`, and `y9 ~ x9`, in order to identify which transformation of Figure 3.7 gives the best fit. For these, do the following:

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

*Hints*:

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

The nonlinear transformations introduced for the simple linear model are readily applicable in the multiple linear model. Consequently, the **multiple linear model is able to approximate** regression functions with **nonlinear forms**, if appropriate nonlinear transformations for the predictors are used.[14]

### 3.4.2 *Polynomial transformations*

**Polynomial models** are a powerful nonlinear extension of the linear model. These are constructed by replacing each predictor $X_j$ by

[14] In particular, Taylor's theorem allows to approximate a sufficiently regular function $m$ by a polynomial, which is identifiable with a multiple linear model as seen in Section 3.4.2. This idea, applied locally, is elaborated in Section 6.2.2.

[15] The consideration of more than one predictor is conceptually straightforward, yet notationally more cumbersome.

a set of monomials $(X_j, X_j^2, \ldots, X_j^{k_j})$ constructed from $X_j$. In the simpler case with a single predictor[15] $X$, we have the $k$-th order polynomial fit:

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

With this approach, a highly flexible model is produced, as it was already shown in Figure 1.1.

The creation of polynomial models can be automated with the R's `poly` function. For the observations $(X_1, \ldots, X_n)$ of $X$, `poly` creates the matrices

$$\begin{pmatrix} X_1 & X_1^2 & \ldots & X_1^k \\ \vdots & \vdots & \ddots & \vdots \\ X_n & X_n^2 & \ldots & X_n^k \end{pmatrix} \text{ or } \begin{pmatrix} p_1(X_1) & p_2(X_1) & \ldots & p_k(X_1) \\ \vdots & \vdots & \ddots & \vdots \\ p_1(X_n) & p_2(X_n) & \ldots & p_k(X_n) \end{pmatrix}, \quad (3.5)$$

where $p_1, \ldots, p_k$ are *orthogonal polynomials*[16] of orders $1, \ldots, k$, respectively. For orthogonal polynomials, `poly` yields a data matrix in (3.5) with *uncorrelated*[17] columns. That is, such that the sample correlation coefficient between two columns is zero.

Let's see a couple of examples on using `poly`.

```
x1 <- seq(-1, 1, l = 4)
poly(x = x1, degree = 2, raw = TRUE) # (X, X^2)
##               1         2
## [1,] -1.0000000 1.0000000
## [2,] -0.3333333 0.1111111
## [3,]  0.3333333 0.1111111
## [4,]  1.0000000 1.0000000
## attr(,"degree")
## [1] 1 2
## attr(,"class")
## [1] "poly"   "matrix"
poly(x = x1, degree = 2) # By default, it employs orthogonal polynomials
##               1    2
## [1,] -0.6708204  0.5
## [2,] -0.2236068 -0.5
## [3,]  0.2236068 -0.5
## [4,]  0.6708204  0.5
## attr(,"coefs")
## attr(,"coefs")$alpha
## [1] -5.551115e-17 -5.551115e-17
##
## attr(,"coefs")$norm2
## [1] 1.0000000 4.0000000 2.2222222 0.7901235
##
## attr(,"degree")
## [1] 1 2
## attr(,"class")
## [1] "poly"   "matrix"

# Depiction of raw polynomials
x <- seq(-1, 1, l = 200)
degree <- 5
matplot(x, poly(x, degree = degree, raw = TRUE), type = "l", lty = 1,
        ylab = expression(x^k))
legend("bottomright", legend = paste("k =", 1:degree), col = 1:degree, lwd = 2)

# Depiction of orthogonal polynomials
matplot(x, poly(x, degree = degree), type = "l", lty = 1,
        ylab = expression(p[k](x)))
legend("bottomright", legend = paste("k =", 1:degree), col = 1:degree, lwd = 2)
```

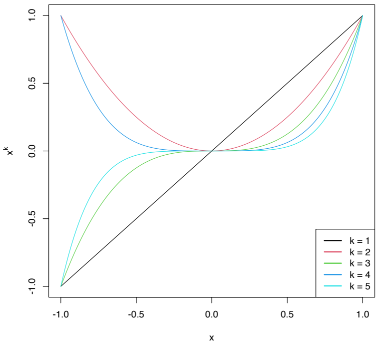


Figure 3.8: Raw polynomials $x^k$ in $(-1,1)$, up to degree $k = 5$.

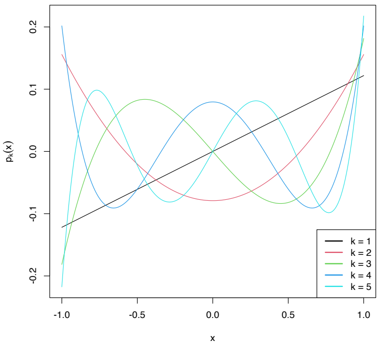


Figure 3.9: Orthogonal polynomials $p_k(x)$ in $(-1,1)$, up to degree $k = 5$.

[16] Precisely, the type of orthogonal polynomials considered are a data-driven shifting and rescaling of the *Legendre polynomials* in $(-1,1)$. The Legendre polynomials $p_j$ and $p_\ell$ in $(-1,1)$ satisfy that $\int_{-1}^{1} p_j(x) p_\ell(x)\,\mathrm{d}x = \frac{2}{2j+1}\delta_{j\ell}$, where $\delta_{j\ell}$ is the Kronecker's delta. The first five Legendre polynomials in $(-1,1)$ are, respectively: $p_1(x) = x$, $p_2(x) = \frac{1}{2}(3x^2-1)$, $p_3(x) = \frac{1}{2}(5x^3-3x)$, $p_4(x) = \frac{1}{8}(35x^4-30x^2+3)$, and $p_5(x) = \frac{1}{8}(63x^5-70x^3+15x)$. Also, $p_0(x) = 1$. The recursive formula $p_{k+1}(x) = (k+1)^{-1}\left[(2k+1)xp_k(x) - kp_{k-1}(x)\right]$ allows to obtain the $(k+1)$-th order Legendre polynomial from the previous two ones. Notice that from $k \geq 3$, $p_k(x)$ involves more monomials than just $x^k$, precisely all the monomials with a lower order and with the same parity. This may complicate the interpretation of a coefficient related to $p_k(X)$ for $k \geq 3$, as it is not directly associated to a $k$-th order effect of $X$.

[17] This is due to how the second matrix of (3.5) is computed: by means of a $QR$ decomposition associated to the first matrix. The decomposition yields a data matrix formed by polynomials that are not *exactly* the Legendre polynomials but a data-driven shifting and rescaling of them.

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
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)
```

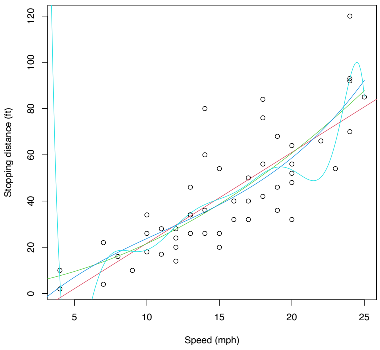


Figure 3.10: Raw and orthogonal polynomial fits of `dist ~ speed` in the `cars` dataset.

```
# 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)
```

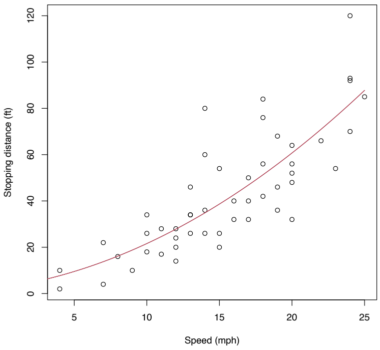


Figure 3.11: Raw and orthogonal polynomial fits of `dist ~ speed` in the `cars` dataset.

```
# 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!
car::scatterplotMatrix(mod2$model[, -1], col = 1, regLine = list(col = 2),
                       smooth = list(col.smooth = 4, col.spread = 4))
car::scatterplotMatrix(mod2Raw$model[, -1],col = 1, regLine = list(col = 2),
                       smooth = list(col.smooth = 4, col.spread = 4))
cor(mod2$model[, -1])
##   1 2
## 1 1 0
## 2 0 1
cor(mod2Raw$model[, -1])
##           1         2
## 1 1.0000000 0.9794765
## 2 0.9794765 1.0000000
```

The use of **orthogonal polynomials** instead of raw polynomials is advised for high order polynomial fits, since they avoid the numerical instabilities arising from excessive linear dependencies between the raw polynomial predictors.

### 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 by means of $X_1X_2$. 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 $X_2$.**[18] Significance testing on these coefficients can be carried out as usual.

The way of adding these interactions in `lm` is through the operators `:` and `*`. The operator `:` only adds the term $X_1X_2$, while `*` adds $X_1$, $X_2$, and $X_1X_2$ at the same time. Let's see an example in the `Boston` dataset.

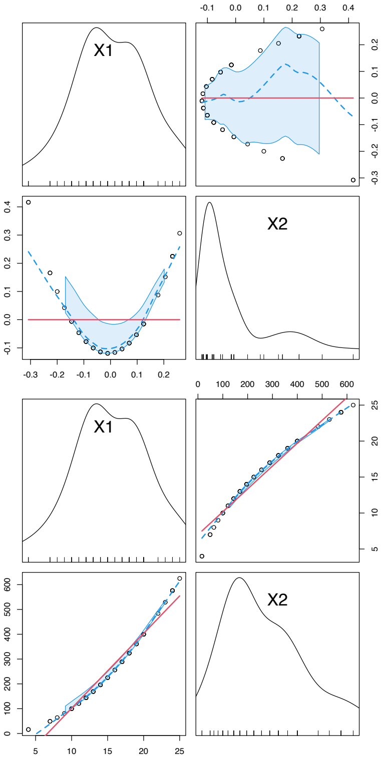


Figure 3.12: Correlations between the first and second order orthogonal polynomials associated to `speed`, and between `speed` and `speed^2`.

[18] Of course, the roles of $X_1$ and $X_2$ can be interchanged in this interpretation.

```
# 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, when age increases makes lstat affect less negatively medv.
# Note that the same interpretation does NOT hold if we switch 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
```

A fast way of accounting interactions between predictors is to use the ˆ 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)` 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)`.

**Stepwise regression** can also be performed with interaction terms. `MASS::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 <- MASS::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
MASS::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
MASS::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
```

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 $D$. Outlining these models helps understand the possible range of effects of adding a dummy variable:

1. *Predictor and no dummy variable*. The usual simple linear model:

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

2. *Predictor and dummy variable.* $D$ affects the intercept of the linear model, 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.* $D$ affects the intercept *and* the slope of the linear model, and both are different for each group:

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

4. *Predictor and interaction with dummy variable.* $D$ affects only the slope of the linear model, 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.* $D$ 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.* $D$ adds the predictor $X$ for one group and affects the intercept, which is different for each group:

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

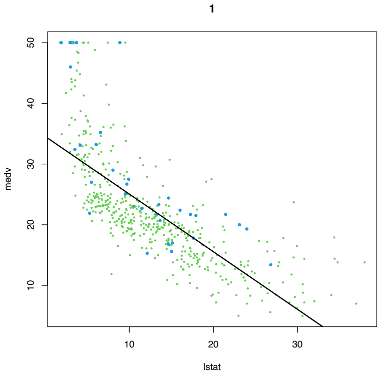
7. *Interaction of dummy and predictor.* $D$ adds the predictor $X$ for one group and the intercept is common:

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

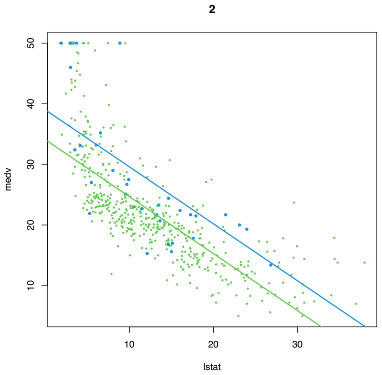
Let's see the visualization of these seven models with $Y =$ `medv`, $X =$ `lstat`, and $D =$ `chas` for the `Boston` dataset. In the following, the green color stands for data points and linear fits associated to $D = 0$, whereas blue stands for $D = 1$.

```
# Group settings
col <- Boston$chas + 3
cex <- 0.5 + 0.25 * Boston$chas

# 1. No dummy variable
```

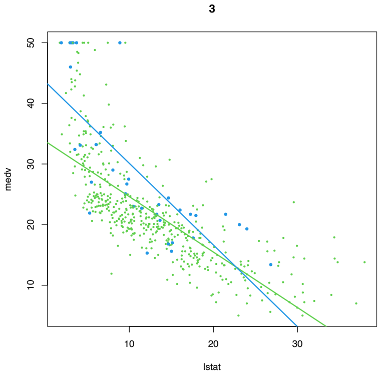
```
(mod1 <- lm(medv ~ lstat, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Coefficients:
## (Intercept)        lstat
##       34.55        -0.95
plot(medv ~ lstat, data = Boston, col = col, pch = 16, cex = cex, main = "1")
abline(coef = mod1$coefficients, lwd = 2)
```



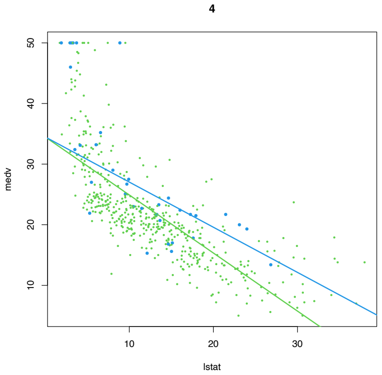
```
# 2. Dummy variable
(mod2 <- lm(medv ~ lstat + chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat + chas, data = Boston)
##
## Coefficients:
## (Intercept)        lstat         chas
##     34.0941      -0.9406       4.9200
plot(medv ~ lstat, data = Boston, col = col, pch = 16, cex = cex, 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)
```



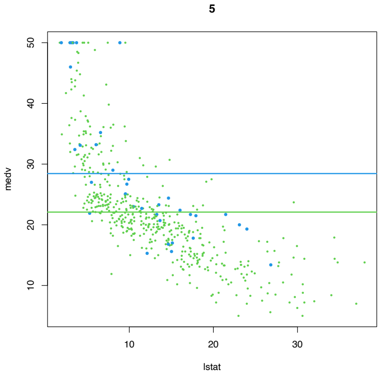
```
# 3. Dummy variable, with interaction
(mod3 <- lm(medv ~ lstat * chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat * chas, data = Boston)
##
## Coefficients:
## (Intercept)        lstat         chas   lstat:chas
##     33.7672      -0.9150       9.8251      -0.4329
plot(medv ~ lstat, data = Boston, col = col, pch = 16, cex = cex, 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)
```



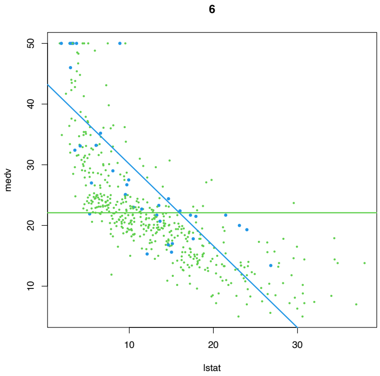
```
# 4. Dummy variable only present in interaction
(mod4 <- lm(medv ~ lstat + lstat:chas, data = Boston))
##
## Call:
## lm(formula = medv ~ lstat + lstat:chas, data = Boston)
##
## Coefficients:
## (Intercept)        lstat   lstat:chas
##     34.4893      -0.9580       0.2128
plot(medv ~ lstat, data = Boston, col = col, pch = 16, cex = cex, 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)
```



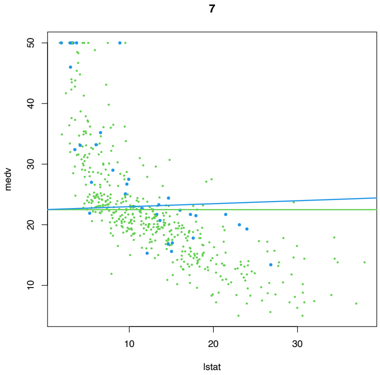
```
# 5. Dummy variable and no predictor
(mod5 <- lm(medv ~ chas, data = Boston))
##
## Call:
## lm(formula = medv ~ chas, data = Boston)
##
## Coefficients:
## (Intercept)         chas
##      22.094        6.346
plot(medv ~ lstat, data = Boston, col = col, pch = 16, cex = cex, main = "5")
abline(a = 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 = col, pch = 16, cex = cex, 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 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 = col, pch = 16, cex = cex, 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 with `chas`
```

This discussion can be extended to factors with **several levels** with more associated dummy variables. The next code chunk shows how three group-specific linear regressions are modeled together by means of two dummy variables in the `iris` dataset.

```
# Does not take into account the groups in the data
modIris <- lm(Sepal.Width ~ Petal.Width, data = iris)
modIris$coefficients
## (Intercept) Petal.Width
##   3.3084256  -0.2093598

# Adding interactions with the groups
modIrisSpecies <- lm(Sepal.Width ~ Petal.Width * Species, data = iris)
modIrisSpecies$coefficients
##                    (Intercept)                    Petal.Width             Speciesversicolor              Speciesvirginica
##                      3.2220507                      0.8371922                    -1.8491878                    -1.5272777
## Petal.Width:Speciesversicolor  Petal.Width:Speciesvirginica
##                      0.2164556                    -0.2057870

# 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)
```

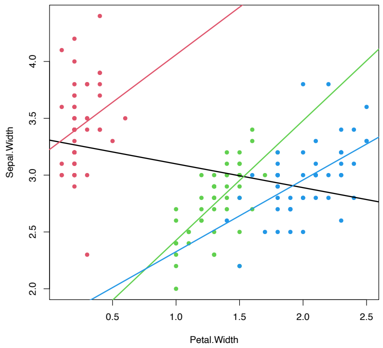


Figure 3.13: The three linear fits of `Sepal.Width ~ Petal.Width * Species` for each of the three levels in the `Species` factor (`setosa` in red, `versicolor` in green, and `virginica` in blue) in the `iris` dataset. The black line represents the linear fit for `Sepal.Width ~ Petal.Width`, that is, the linear fit without accounting for the levels in `Species`. Recall how `Sepal.Width`is *positively* correlated with `Petal.Width` *within* each group, but is *negatively* correlated in the *aggregated* data.

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.

If groups are present in the data, **always explore the intra-group correlations!**

### 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`.

**Exercise 3.8.** 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 $R^2_{\text{Adj}}$? 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 of the linear model.

We will illustrate them in the `wine` dataset, which is available in the `wine.RData` workspace:

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

Before going into specifics, recall the following general comment on performing model diagnostics.

> 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. Therefore, identifying the **root cause** of the assumptions failure is key in order to try to find a patch.

### 3.5.1 *Linearity*

Linearity between the response $Y$ and the predictors $X_1, \ldots, X_p$ 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 $X_1, \ldots, X_p$ in a significant way, then the linear fit will yield flawed conclusions, with varying degrees of severity, about the explanation of $Y$. Therefore, linearity is a **crucial assumption**.

### How to check it

The so-called *residuals vs. fitted values plot* is the scatterplot of $\{(\hat{Y}_i, \hat{\varepsilon}_i)\}_{i=1}^n$ and is a very useful tool for detecting linearity departures using a *single*[19] graphical device. Figure 3.14 illustrates how the residuals vs. fitted values plots behave for situations in which the linearity is known to be respected or violated.

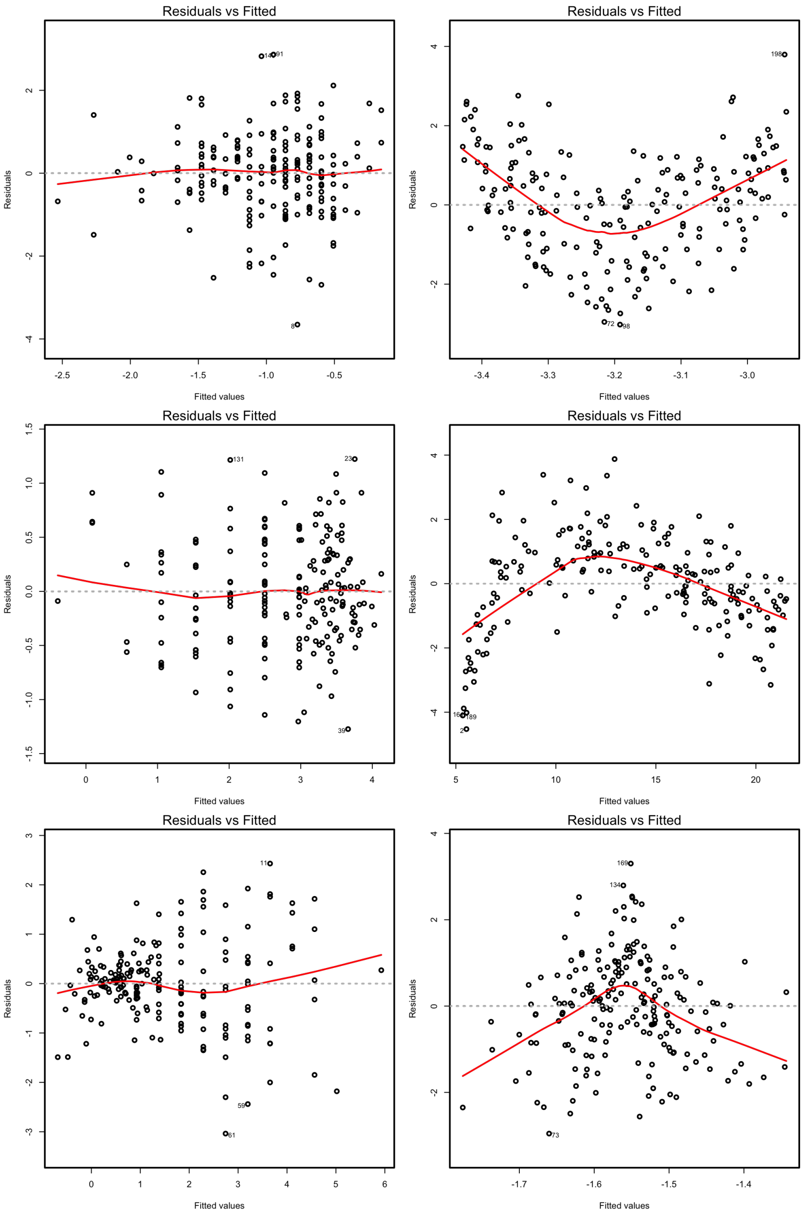


Figure 3.14: Residuals vs. fitted values plots for datasets respecting (left column) and violating (right column) the linearity assumption.

[19] If $p = 1$, then it is possible to inspect the scatterplot of the $\{(X_{i1}, Y_i)\}_{i=1}^n$ in order to determine whether linearity is plausible. But the usefulness of this graphical check quickly decays with the dimension $p$, as $p$ scatterplots need to be investigated. That is precisely the key point for relying in the residuals vs. fitted values plot.

**Under linearity**, we expect that there is **no significant trend** in the residuals $\hat{\varepsilon}_i$ with respect to $\hat{Y}_i$. This can be assessed by checking if a flexible fit of the mean of the residuals is constant. Such fit is given by the **red line** produced by:

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

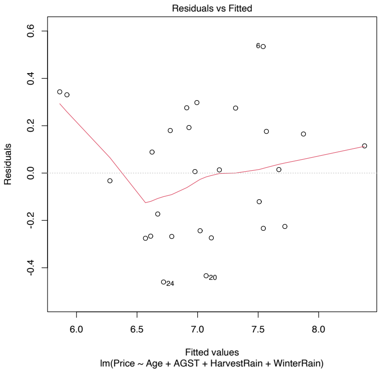


Figure 3.15: Residuals vs. fitted values for the `Price ~ Age + AGST + HarvestRain + WinterRain` model for the `wine` dataset.

If nonlinearities are observed, it is worth to plot the *regression terms* of the model. These are the $p$ scatterplots of $\{(X_{ij}, Y_i)\}_{i=1}^n$, for $j = 1, \ldots, p$, that are accompanied by the regression lines $y = \hat{\beta}_0 + \hat{\beta}_j x_j$, where $(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)^\top$ come from the *multiple* linear fit, not from individual *simple* linear regressions. The regression terms help identifying which predictors have nonlinear effects on $Y$.

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

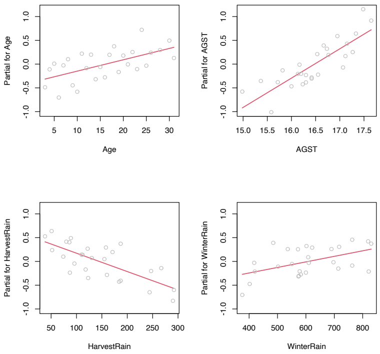


Figure 3.16: Regression terms for `Price ~ Age + AGST + HarvestRain + WinterRain` in the `wine` dataset.

### What to do if it fails

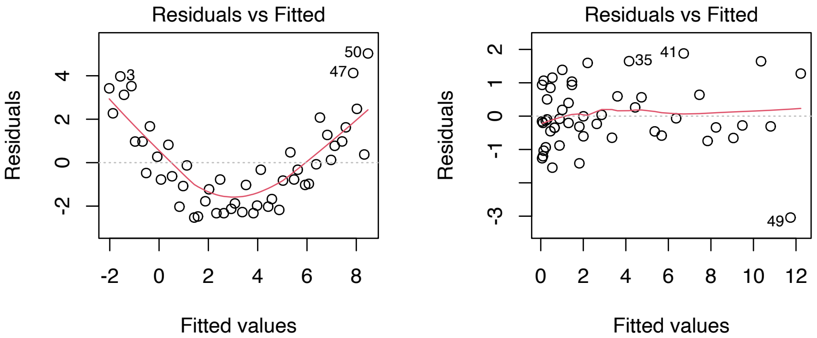
Using an adequate nonlinear transformation for the problematic predictors or adding interaction terms, as we saw in Section 3.4, might be helpful in linearizing the relation between $Y$ and $X_1, \ldots, X_p$. Alternatively, one can consider a nonlinear transformation $f$ *for the response* $Y$, at the expenses of:

1. Modeling $W := f(Y)$ rather than $Y$, thus slightly changing the setting of the original modeling problem.
2. Alternatively, untransforming the linear prediction $\hat{W}$ as $\hat{Y}_f := f^{-1}(\hat{W})$ and performing **biased predictions**[20,21] for $Y$.

Transformations of $Y$ are explored in Section 3.5.7. Chapter 5 is devoted to replacing the untransformed approach with a more principled one.[22]

Let's see the transformation of predictors in the example that motivated Section 3.4.

```r
par(mfrow = c(1, 2))
plot(lm(y ~ x, data = nonLinear), 1) # Nonlinear
plot(lm(y ~ I(x^2), data = nonLinear), 1) # Linear
```



[20] Assume that $f(Y) = \beta_0 + \beta_1 X + \varepsilon$, with $f$ nonlinear and invertible. Clearly, $W = f(Y)$ satisfies $W = \beta_0 + \beta_1 X + \varepsilon$, and we can estimate $\mathbb{E}[W \mid X = x]$ with $\hat{w} = \hat{\beta}_0 + \hat{\beta}_1 x$ unbiasedly, i.e., $\mathbb{E}[\hat{w} \mid X_1, \ldots, X_n] = \mathbb{E}[W \mid X = x]$. Define $\hat{y}_f := f^{-1}(\hat{w})$. Because of the delta method (see next footnote), $\mathbb{E}[\hat{y}_f \mid X_1, \ldots, X_n]$ is asymptotically equal to $f^{-1}(\mathbb{E}[W \mid X = x])$. However, $f^{-1}(\mathbb{E}[W \mid X = x]) \neq (\mathbb{E}[f^{-1}(W) \mid X = x]) = \mathbb{E}[Y \mid X = x]$, so $\hat{y}_f$ is a biased estimator of $\mathbb{E}[Y \mid X = x]$. Indeed, if $f$ is *strictly convex* (e.g., $f = \log$), then $f^{-1}$ is strictly concave, and Jensen's inequality guarantees that $f^{-1}(\mathbb{E}[W \mid X = x]) \geq \mathbb{E}[f^{-1}(W) \mid X = x]$, that is, $\hat{y}_f$ *underestimates* $\mathbb{E}[Y \mid X = x]$.

[21] **Delta method**. Let $\hat{\theta}$ be an estimator of $\theta$ such that $\sqrt{n}(\hat{\theta} - \theta)$ is asymptotically a $\mathcal{N}(0, \sigma^2)$ when $n \to \infty$. If $g$ is a real-valued function such that $g'(\theta) \neq 0$ exists, then $\sqrt{n}(g(\hat{\theta}) - g(\theta))$ is asymptotically a $\mathcal{N}(0, \sigma^2 (g'(\theta))^2)$.

[22] Given that removing the bias introduced by considering $\hat{Y}_f = f^{-1}(\hat{W})$ is challenging.

### 3.5.2 *Normality*

The assumed normality of the errors $\varepsilon_1, \ldots, \varepsilon_n$ allows us to make *exact* inference in the linear model, in the sense that the distribution of $\hat{\boldsymbol\beta}$ given in (2.11) is exact for *any* $n$, and not asymptotic with $n \to \infty$. If normality does not hold, then the inference we did (CIs for $\beta_j$, hypothesis testing, CIs for prediction) is to be *somehow* suspected. Why just *somehow*? Roughly speaking, the reason is that the central limit theorem will make $\hat{\boldsymbol\beta}$ *asymptotically* normal[23], 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. Hence the next rule of thumb:

> Non-severe[24] departures from normality yield valid (asymptotic) inference for relatively large sample sizes $n$.

Therefore, the failure of normality is typically less problematic than other assumptions.

**How to check it**

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

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

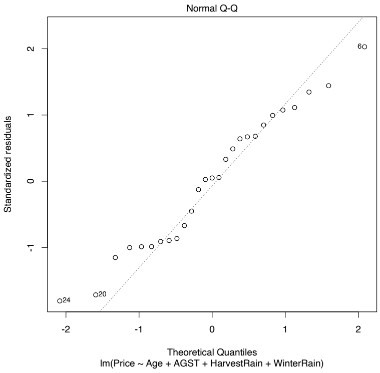


Figure 3.17: QQ-plot for the residuals of the `Price ~ Age + AGST + HarvestRain + WinterRain` model for the `wine` dataset.

**Under normality**, we expect the **points** to align with the **diagonal line**, which represents the expected position of the points if they were sampled from a $\mathcal{N}(0,1)$. It is usual to have larger departures from the diagonal in the extremes[25] 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. Two popular ones are the *Shapiro–Wilk test*, implemented by the `shapiro.test` function, and the *Lilliefors test*,[26] implemented by the `nortest::lillie.test` function:

```r
# Shapiro-Wilk test of normality
shapiro.test(mod$residuals)
##
##  Shapiro-Wilk normality test
##
## data:  mod$residuals
## W = 0.95903, p-value = 0.3512
# We do not reject normality
# shapiro.test allows up to 5000 observations -- if dealing with more data
# points, randomization of the input is a possibility

# Lilliefors test -- the Kolmogorov-Smirnov adaptation for testing normality
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
```

[23] Recall that $\hat\beta_j = \mathbf{e}_j^o p(\mathbf{X}^o p\mathbf{X})^{-1}\mathbf{X}^o p\mathbf{Y} =: \sum_{i=1}^n w_{ij}Y_i$, where $\mathbf{e}_j$ is the canonical vector of $\mathbb{R}^{p+1}$ with 1 in the $j$-th position and 0 elsewhere. Therefore, $\hat\beta_j$ is a weighted sum of the random variables $Y_1, \ldots, Y_n$ (recall that we assume that $\mathbf{X}$ is given and therefore is deterministic). Even if $Y_1, \ldots, Y_n$ are not normal, the central limit theorem entails that $\sqrt{n}(\hat\beta_j - \beta_j)$ is *asymptotically* normally distributed when $n \to \infty$, **provided that linearity holds**.

[24] Distributions that are not heavy tailed, not heavily multimodal, and not heavily skewed.

[25] For $X \sim F$, the $p$-th quantile $x_p = F^{-1}(p)$ of $X$ is estimated through the sample quantile $\hat{x}_p := F_n^{(-1)}(p)$, where $F_n(x) = \frac{1}{n}\sum_{i=1}^n 1_{\{X_i \leq x\}}$ is the empirical cdf of $X_1, \ldots, X_n$ and $F_n^{(-1)}$ is the generalized inverse function of $F_n$. If $X \sim f$ is continuous, then $\sqrt{n}(\hat{x}_p - x_p)$ is asymptotically a $\mathcal{N}\left(0, \frac{p(1-p)}{f(x_p)^2}\right)$. Therefore, the variance of $\hat{x}_p$ grows if $p \to 0$ or $p \to 1$, hence more variability is expected on the extremes of the QQ-plot (see Figure 3.19).

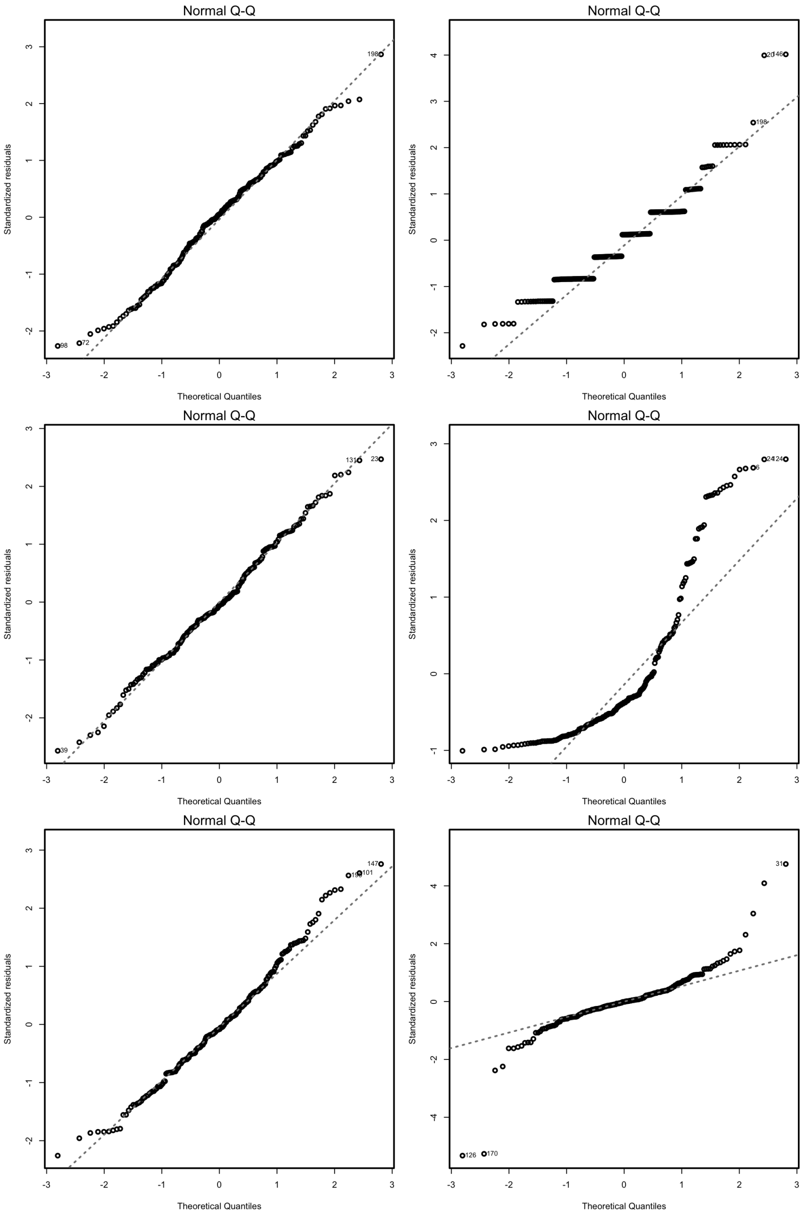


Figure 3.18: QQ-plots for datasets respecting (left column) and violating (right column) the normality assumption.

### What to do if it fails

Patching non-normality is not easy and most of the time requires the consideration of other models, like the ones seen in Chapter 5. If $Y$ is nonnegative, one possibility is to transform $Y$ by means of the **Box–Cox** (Box and Cox, 1964) transformation:

$$y^{(\lambda)} := \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \lambda \neq 0, \\ \log(y), & \lambda = 0. \end{cases} \tag{3.6}$$

This transformation alleviates the skewness[27] of the data, therefore making it more symmetric and hence normal-like. The optimal data-dependent $\hat\lambda$ that makes the data most normal-like can be found through maximum likelihood[28] on the transformed sample $\{Y_i^{(\lambda)}\}_{i=1}^n$. If $Y$ is not nonnegative, (3.6) cannot be applied. A possible patch is to shift the data by a positive constant $m = -\min(Y_1, \ldots, Y_n) + \delta$, $\delta > 0$, such that transformation (3.6) becomes

$$y^{(\lambda,m)} := \begin{cases} \frac{(y+m)^\lambda - 1}{\lambda}, & \lambda \neq 0, \\ \log(y+m), & \lambda = 0. \end{cases}$$

A neat alternative to this shifting is to rely on a transformation that is already designed for *real* $Y$, such as the **Yeo–Johnson** (Yeo and Johnson, 2000) transformation:

$$y^{(\lambda)} := \begin{cases} \frac{(y+1)^\lambda - 1}{\lambda}, & \lambda \neq 0, y \geq 0, \\ \log(y+1), & \lambda = 0, y \geq 0, \\ -\frac{(-y+1)^{2-\lambda} - 1}{2-\lambda}, & \lambda \neq 2, y < 0, \\ -\log(-y+1), & \lambda = 2, y < 0. \end{cases} \tag{3.7}$$

The beauty of the Yeo–Johnson transformation is that it extends neatly the Box–Cox transformation, which appears as a particular case when $Y$ is nonnegative and after performing a shift of 1 units (see Figure 3.20). As with the Box–Cox transformation, the optimal $\hat\lambda$ is estimated through maximum likelihood on the transformed sample $\{Y_i^{(\lambda)}\}_{i=1}^n$.

```
N <- 200
y <- seq(-4, 4, length.out = N)
lambda <- c(0, 0.5, 1, 2, -0.5, -1, -2)
l <- length(lambda)
psi <- sapply(lambda, function(la) car::yjPower(U = y, lambda = la))
matplot(y, psi, type = "l", ylim = c(-4, 4), lwd = 2, lty = 1:l,
        ylab = latex2exp::TeX("$y^{(\\lambda)}$"), col = 1:l, las = 1,
        main = "Yeo-Johnson transformation")
abline(v = 0, h = 0)
abline(v = -1, h = 0, lty = 2)
legend("bottomright", lty = 1:l, lwd = 2, col = 1:l,
       legend = latex2exp::TeX(paste0("$\\lambda = ", lambda, "$")))
```

Let's see how to implement both transformations using the `car::powerTransform`, `car::bcPower`, and `car::yjPower` functions.

```
# Test data
```

[26] This is the Kolmogorov–Smirnov test shown in Section A but adapted to testing the normality of the data with *unknown* mean and variance. More precisely, the test tests the *composite* null hypothesis $H_0 : F = \Phi(\cdot; \mu, \sigma^2)$ with $\mu$ and $\sigma^2$ unknown. Note that this is different from the *simple* null hypothesis $H_0 : F = F_0$ of the Kolmogorov–Smirnov test in which $F_0$ is completely specified. Further tests of normality can be derived by adapting other tests for the simple null hypothesis $H_0 : F = F_0$, such as the Cramér–von Mises and the Anderson–Darling tests, and these are implemented in the functions `nortest::cvm.test` and `nortest::ad.test`.

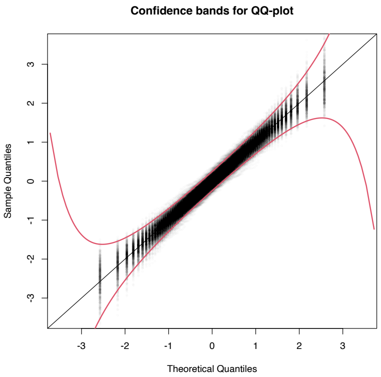


Figure 3.19: The uncertainty behind the QQ-plot. The figure aggregates $M = 1000$ different QQ-plots of $\mathcal{N}(0,1)$ data with $n = 100$, displaying for each of them the pairs $(x_p, \hat{x}_p)$ evaluated at $p = \frac{i-1/2}{n}$, $i = 1, \ldots, n$ (as they result from `ppoints(n)`). The uncertainty is measured by the asymptotic $100(1-\alpha)\%$ CIs for $\hat{x}_p$, given by $\left(x_p \pm \frac{z_{1-\alpha/2}}{\sqrt{n}} \frac{\sqrt{p(1-p)}}{\phi(x_p)}\right)$. These curves are displayed in red for $\alpha = 0.05$. Observe that the vertical strips arise since the $x_p$ coordinate is deterministic.

[27] Precisely, if $\lambda < 1$, *positive skewness* (or skewness to the right) is palliated (large values of $Y$ shrink, small values grow), whereas if $\lambda > 1$ *negative skewness* (or skewness to the left) is corrected (large values of $Y$ grow, small values shrink).

[28] Maximum likelihood for a $\mathcal{N}(\mu, \sigma^2)$ and potentially using the linear model structure if we are performing the transformation to achieve normality in errors of the linear model. Recall that optimally transforming $Y$ such that $Y$ is normal-like or $Y \mid (X_1, \ldots, X_p)$ is normal-like (the assumption in the linear model) are very different goals!

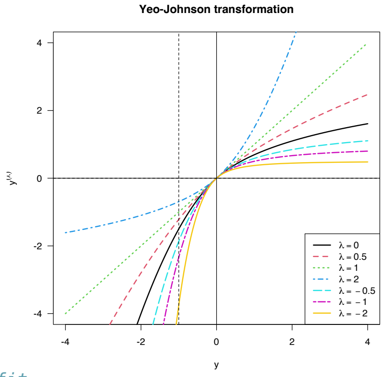
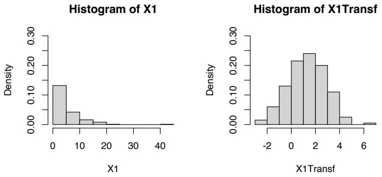
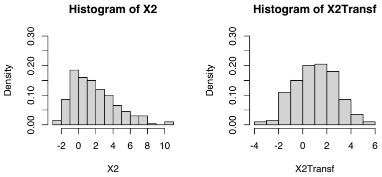


Figure 3.20: Yeo–Johnson transformation for some values of $\lambda$. The Box–Cox transformation for $\lambda$, and shifted by $-1$ units, corresponds to the part $y \geq -1$ of the plot.

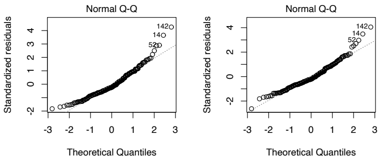
```
# Predictors
n <- 200
set.seed(121938)
X1 <- rexp(n, rate = 1 / 5) # Nonnegative
X2 <- rchisq(n, df = 5) - 3 # Real

# Response of a linear model
epsilon <- rchisq(n, df = 10) - 10 # Centered error, but not normal
Y <- 10 - 0.5 * X1 + X2 + epsilon

# Transformation of non-normal data to achieve normal-like data (no model)

# Optimal lambda for Box-Cox
BC <- car::powerTransform(lm(X1 ~ 1), family = "bcPower") # Maximum-likelihood fit
# Note we use a regression model with no predictors
(lambdaBC <- BC$lambda) # The optimal lambda
##        Y1
## 0.2412419
# lambda < 1, so positive skewness is corrected

# Box-Cox transformation
X1Transf <- car::bcPower(U = X1, lambda = lambdaBC)

# Comparison
par(mfrow = c(1, 2))
hist(X1, freq = FALSE, breaks = 10, ylim = c(0, 0.3))
hist(X1Transf, freq = FALSE, breaks = 10, ylim = c(0, 0.3))
```



```
# Optimal lambda for Yeo-Johnson
YJ <- car::powerTransform(lm(X2 ~ 1), family = "yjPower")
(lambdaYJ <- YJ$lambda)
##        Y1
## 0.5850791

# Yeo-Johnson transformation
X2Transf <- car::yjPower(U = X2, lambda = lambdaYJ)

# Comparison
par(mfrow = c(1, 2))
hist(X2, freq = FALSE, breaks = 10, ylim = c(0, 0.3))
hist(X2Transf, freq = FALSE, breaks = 10, ylim = c(0, 0.3))
```



```
# Transformation of non-normal response in a linear model

# Optimal lambda for Yeo-Johnson
YJ <- car::powerTransform(lm(Y ~ X1 + X2), family = "yjPower")
(lambdaYJ <- YJ$lambda)
##        Y1
## 0.9160924

# Yeo-Johnson transformation
YTransf <- car::yjPower(U = Y, lambda = lambdaYJ)

# Comparison for the residuals
par(mfrow = c(1, 2))
plot(lm(Y ~ X1 + X2), 2)
plot(lm(YTransf ~ X1 + X2), 2) # Slightly better
```



Be aware that using the previous transformations implies modeling the transformed response rather than $Y$. Normality is also possible to patch 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 CIs for prediction will not respect the confidence for which they were built.

**How to check it**

Heteroscedasticity 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, and inspect the deviations in the positive axis.

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

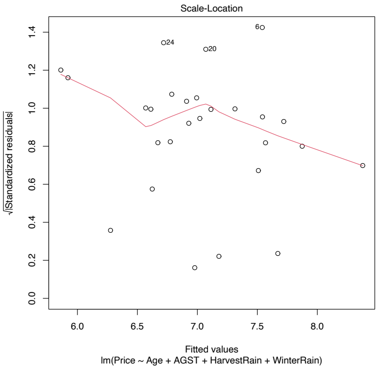


Figure 3.21: Scale-location plot for the `Price ~ Age + AGST + HarvestRain + WinterRain` model for the `wine` dataset.

**Under homoscedasticity**, we expect the **red line**, a flexible fit of the mean of the transformed residuals, to be **almost constant about** 1. If there are clear non-constant patterns, then there is evidence of heteroscedasticity.

There are formal tests to check the null hypothesis of homoscedasticity in our residuals. For example, the *Breusch–Pagan test* implemented in `car::ncvTest`:

```r
# Breusch-Pagan test
car::ncvTest(mod)
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 0.3254683, Df = 1, p = 0.56834
# We do not reject homoscedasticity
```

The Breusch–Pagan test checks homoscedasticity against a non-constant variance in the residuals that is *linearly increasing with respect to the predictors*. This means that the test can be *fooled* by a nonlinear pattern in the variance of the residuals that results in a flat linear fit (e.g., a quadratic pattern). It is advised then to **check the scale-location plot if performing the Breusch–Pagan test**, in order to identify evident non-constant variances hidden behind tricky nonlinearities.

The next code illustrates the previous warning with two examples.

```r
# Heteroscedastic 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)

# Heteroscedasticity not detected
car::ncvTest(modHet1)
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
```

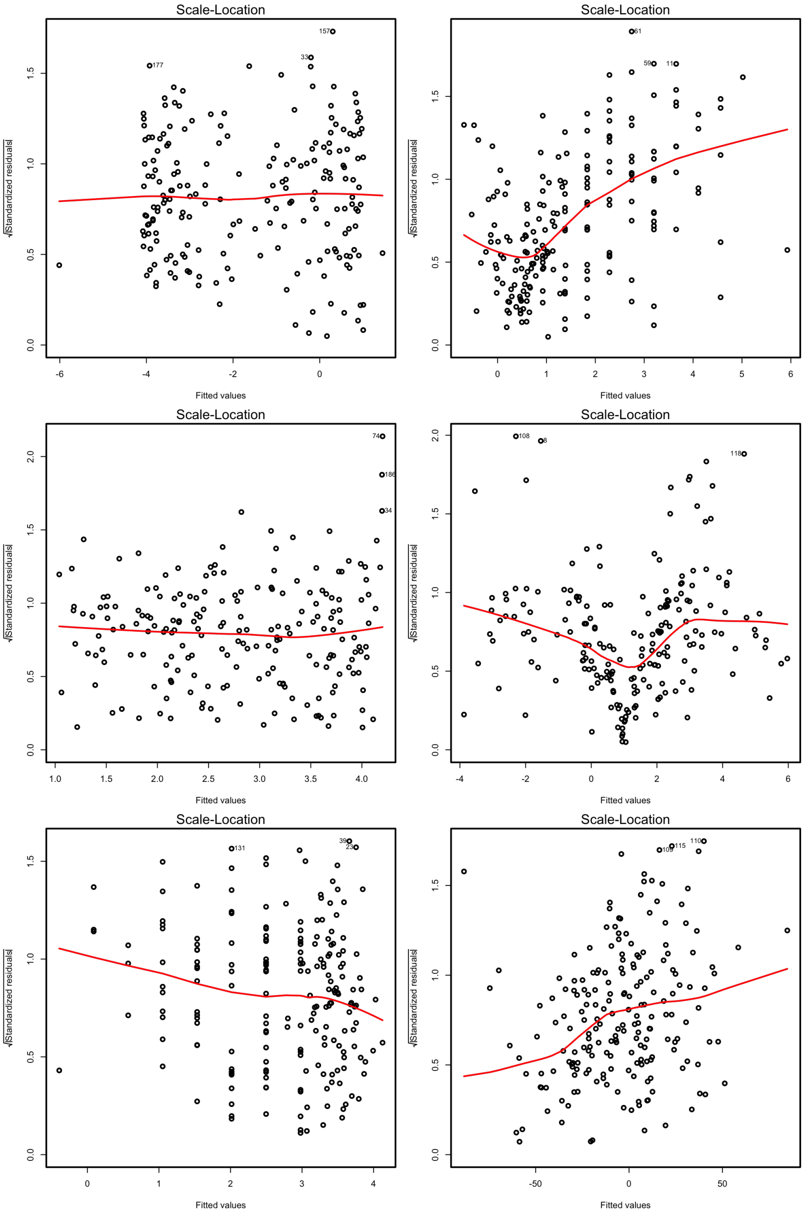


Figure 3.22: Scale-location plots for datasets respecting (left column) and violating (right column) the homoscedasticity assumption.

```
## Chisquare = 2.938652e-05, Df = 1, p = 0.99567
plot(modHet1, 3)

# Heteroscedasticity correctly detected
car::ncvTest(modHet2)
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 41.03562, Df = 1, p = 1.4948e-10
plot(modHet2, 3)
```

### What to do if it fails

Using a nonlinear transformation for the response $Y$ may help to 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 heteroscedasticity. Keep in mind that these transformations, as the Box–Cox transformations, are designed for nonnegative $Y$. The Yeo–Johnson transformation can be used instead if $Y$ is real.

Let's see some examples.

```
# Artificial data with heteroscedasticity
set.seed(12345)
X <- rchisq(500, df = 3)
e <- rnorm(500, sd = sqrt(0.1 + 2 * X))
Y <- 1 + X + e

# Original
plot(lm(Y ~ X), 3) # Very heteroscedastic

# Transformed
plot(lm(I(log(abs(Y))) ~ X), 3) # Much less heteroscedastic, but at the price
# of losing the signs in Y...

# Shifted and transformed
delta <- 1 # This is tunable
m <- -min(Y) + delta
plot(lm(I(log(Y + m)) ~ X), 3) # No signs loss

# Transformed by Yeo-Johnson

# Optimal lambda for Yeo-Johnson
YJ <- car::powerTransform(lm(Y ~ X), family = "yjPower")
(lambdaYJ <- YJ$lambda)
##        Y1
## 0.6932053

# Yeo-Johnson transformation
YTransf <- car::yjPower(U = Y, lambda = lambdaYJ)
plot(lm(YTransf ~ X), 3) # Slightly less heteroscedastic
```

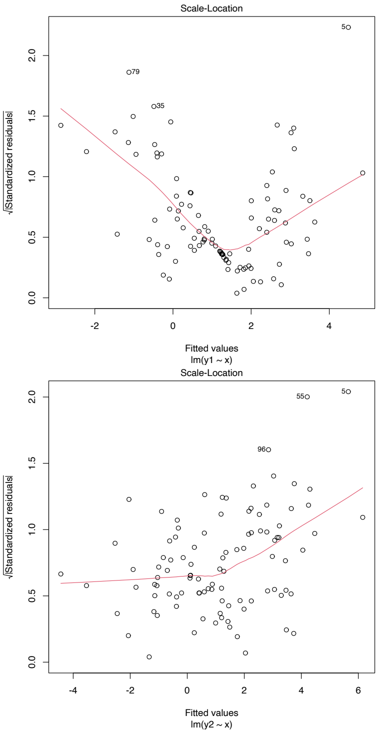


Figure 3.23: Two heteroscedasticity patterns that are undetected and detected, respectively, by the Breusch–Pagan test.

## 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 $X_1, \ldots, X_p$, from $n$ 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, 95% CIs built under the assumption of independence will be shorter 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 $\boldsymbol{\beta}$. Yet, in the $2n$-sample the length of the confidence intervals is $C(2n)^{-1/2}$, whereas in the $n$-sample they have length $Cn^{-1/2}$. 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 is that the information that is actually in the sample is smaller due to dependence.

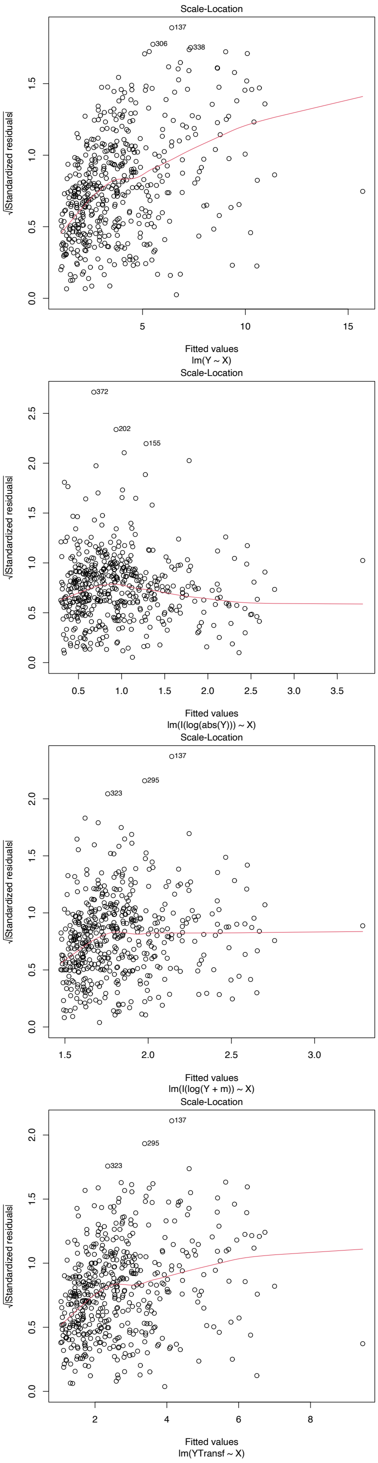


Figure 3.24: Patching of heteroscedasticity for an artificial dataset.

### How to check it

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

```r
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 autocorrelation, manifested as alternating small-large or positive-negative residuals, is also possible. The **lagged plots** of $(\hat{\varepsilon}_{i-\ell}, \hat{\varepsilon}_i)$, $i = \ell + 1, \ldots, n$, obtained through `lag.plot`, allow us to detect both kinds of autocorrelations for a given lag $\ell$. Under independence, we expect no trend in such plot. Here is an example:

```r
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, such as the *Durbin–Watson test* implemented in `car::durbinWatsonTest`:

```r
# Durbin-Watson test
car::durbinWatsonTest(mod)
##  lag Autocorrelation D-W Statistic p-value
##    1      -0.4160168      2.787261   0.054
##  Alternative hypothesis: rho != 0
# Does not reject at alpha = 0.05
```

### What to do if it fails

Little can be done if there is dependence in the data, once this has been collected. If the dependence is temporal, we must rely on the family of statistical models meant to deal with serial dependence: *time series*. Other kinds of dependence such as spatial dependence,

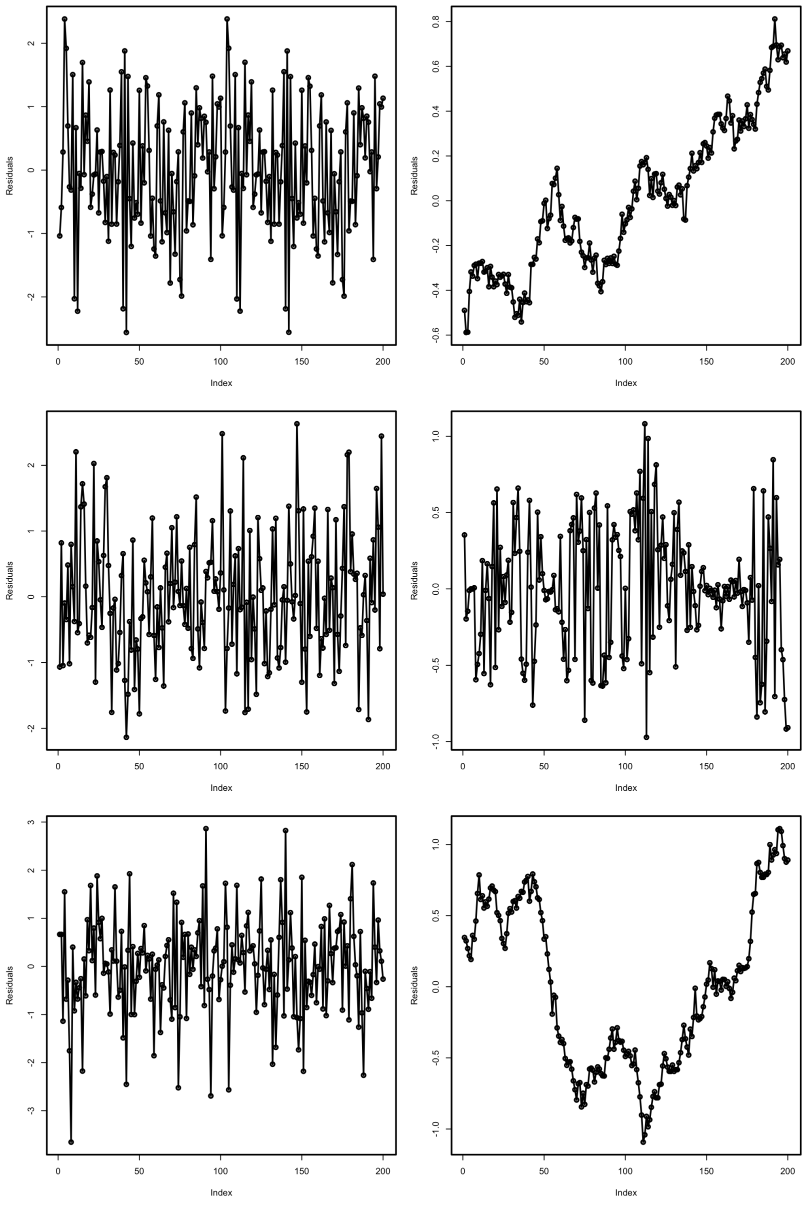


Figure 3.26: Serial plots of the residuals for datasets respecting (left column) and violating (right column) the independence assumption.

spatio-temporal dependence, geometrically-driven dependencies, censorship, truncation, etc. need to be analyzed with a different set of tools to the ones covered in these notes.

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

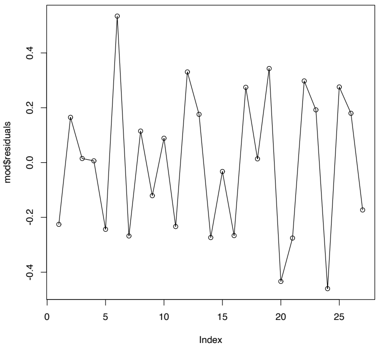
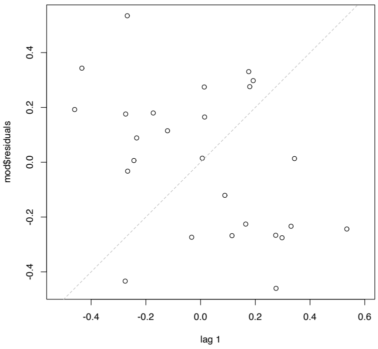


Figure 3.25: Serial plot of the residuals of the `Price ~ Age + AGST + HarvestRain + WinterRain` model for the `wine` dataset.



**Exercise 3.9.** Load the dataset `assumptions3D.RData` and compute the regressions `y.3 ~ x1.3 + x2.3`, `y.4 ~ x1.4 + x2.4`, `y.5 ~ x1.5 + x2.5`, and `y.8 ~ 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, the signs of fitted coefficients may be even reversed and valuable predictors may appear as non-significant. In limit cases, numerical instabilities may appear since $\mathbf{X}^\top\mathbf{X}$ will be almost singular.
- It is **difficult to determine how each of the highly-related predictors affects the response**, since one masks the other.

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 optimal plane.

An approach to detect multicollinearity is to inspect the correlation matrix between the predictors.

```
# Numerically
round(cor(wine), 2)
##              Year Price WinterRain  AGST HarvestRain   Age FrancePop
## Year         1.00 -0.46       0.05 -0.29       -0.06 -1.00      0.99
## Price       -0.46  1.00       0.13  0.67       -0.51  0.46     -0.48
## WinterRain   0.05  0.13       1.00 -0.32       -0.27 -0.05      0.03
## AGST        -0.29  0.67      -0.32  1.00       -0.03  0.29     -0.30
## HarvestRain -0.06 -0.51      -0.27 -0.03        1.00  0.06     -0.03
## Age         -1.00  0.46      -0.05  0.29        0.06  1.00     -0.99
## FrancePop    0.99 -0.48       0.03 -0.30       -0.03 -0.99      1.00

# Graphically
corrplot::corrplot(cor(wine), addCoef.col = "grey")
```

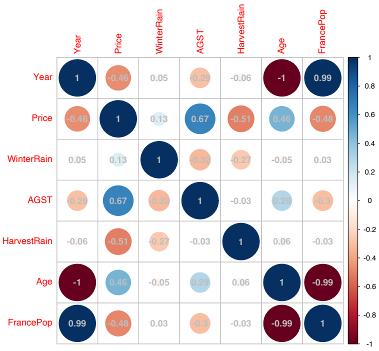


Figure 3.27: Graphical visualization of the correlation matrix.

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

However, it is **not enough** to **inspect pairwise correlations** in order to get rid of multicollinearity. Indeed, it is possible to build counterexamples that show non-suspicious pairwise correlations but problematic complex linear relations that remain hidden. For the sake of illustration, 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
corrplot::corrplot(cor(data), addCoef.col = "grey")
```

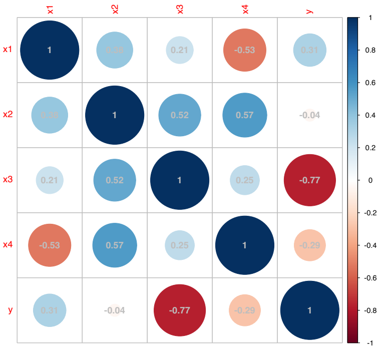


Figure 3.28: Unsuspicious correlation matrix with hidden multicollinearity.

A better approach to detect multicollinearity is to compute the **Variance Inflation Factor** (VIF) of each fitted coefficient $\hat{\beta}_j$. This is a measure of *how linearly dependent is* $X_j$ *on the rest of the predictors*[29] and it is defined as

$$\mathrm{VIF}(\hat{\beta}_j) = \frac{1}{1 - R^2_{X_j|X_{-j}}},$$

where $R^2_{X_j|X_{-j}}$ represents the $R^2$ from the regression of $X_j$ onto the remaining predictors $X_1, \ldots, X_{j-1}, X_{j+1} \ldots, X_p$. Clearly, $\mathrm{VIF}(\hat{\beta}_j) \geq 1$. 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 `car::vif` function, which takes as an argument a linear model. Let's see how it works in the previous example with hidden multicollinearity.

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

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

# Comparison
car::compareCoefs(modMultiCo, modClean)
## Calls:
## 1: lm(formula = y ~ x1 + x2 + x3 + x4)
## 2: lm(formula = y ~ x1 + x2 + x3)
##
```

[29] And therefore more informative on the linear dependence of the predictors than the correlations of $X_j$ with *each* of the remaining predictors.

```
##             Model 1 Model 2
## (Intercept)   1.062   1.058
## SE            0.103   0.103
##
## x1            0.922   1.450
## SE            0.551   0.116
##
## x2            1.640   1.119
## SE            0.546   0.124
##
## x3           -3.165  -3.145
## SE            0.109   0.107
##
## x4           -0.529
## SE            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
car::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. In the extreme case in which the multicollinearity is perfect, then the model will not be **identifiable**, despite being correct.

The next simulation study illustrates how the non-identifiability of $\boldsymbol{\beta}$ worsens with increasing multicollinearity, and therefore so does the error on estimating $\boldsymbol{\beta}$. We consider the linear model

$$Y = X_1 + X_2 + X_3 + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0,1),$$

where

$$X_1, X_2 \sim \mathcal{N}(0,1), \quad X_3 = b(X_1 + X_2) + \delta, \quad \delta \sim \mathcal{N}(0, 0.25)$$

and $b \geq 0$ controls the degree of multicollinearity of $X_3$ with $X_1$ and $X_2$ (if $b = 0$, $X_3$ is uncorrelated with $X_1$ and $X_2$). We aim to study how $\mathbb{E}[\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|]$ behaves when $n$ grows, for different values of $b$. To do so, we consider $n = 2^\ell$, $\ell = 3, \ldots, 12$, $b = 0, 0.5, 1, 5, 10$, and approximate the expected error by Monte Carlo using $M = 500$ replicates. The results of the simulation study are shown in Figure 3.29, which illustrates that multicollinearity notably decreases the accuracy of $\hat{\boldsymbol{\beta}}$, at all sample sizes $n$.

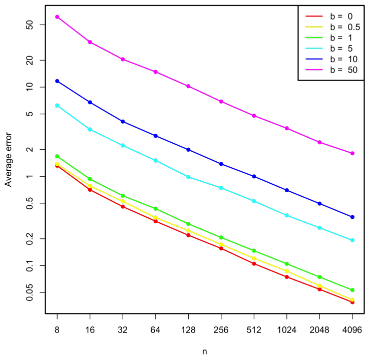


Figure 3.29: Estimated average error $\mathbb{E}[\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|]$, for varying $n$ and degrees of multicollinearity, indexed by $b$. The horizontal and vertical axes are in logarithmic scales.

**Exercise 3.10.** Replicate Figure 3.29 by implementing the simulation study behind it.

### 3.5.6 *Outliers and high-leverage points*

Outliers and high-leverage points are particular observations that have an important impact in the final linear model, either on the estimates or on the properties of the model. They are defined as follows:

- **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 inflate $\hat{\sigma}$ and, as a consequence, they draw down the $R^2$ of the model and expand the CIs.
- **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 linear model fit. The reason is the squared distance in the RSS: an individual *extreme* point may represent a large portion of the RSS.

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

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

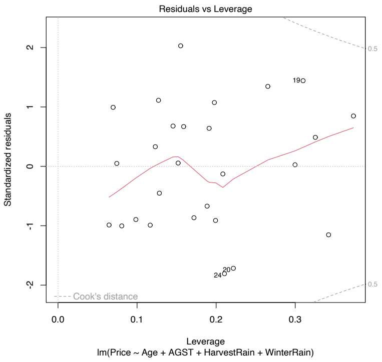


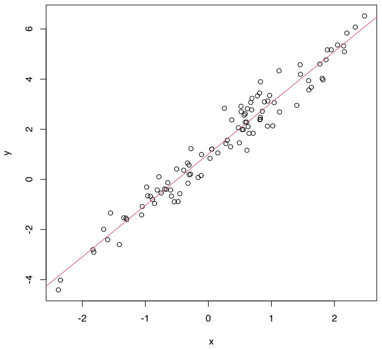
Figure 3.30: Residuals vs. leverage plot for the `Price ~ Age + AGST + HarvestRain + WinterRain` model for the `wine` dataset.

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

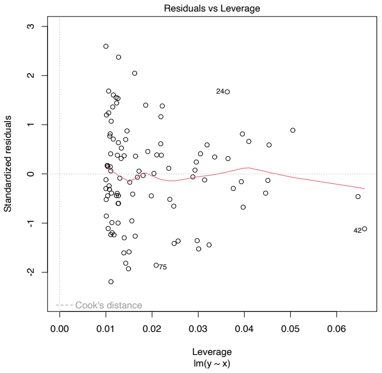
- If the standardized residual of an observation is larger than 3 in absolute value, then it may be an outlier.
- If the leverage statistic $h_i$ is *greatly exceeding* $(p+1)/n$ [30] (see below), then the $i$-th observation may be suspected of having a high leverage.

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)
plot(x, y)
abline(coef = m0$coefficients, col = 2)
```



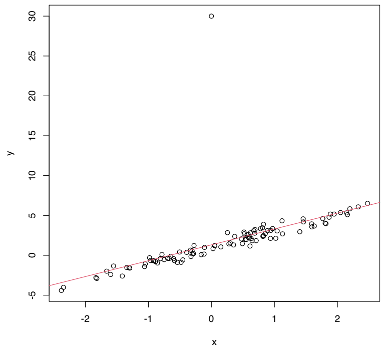
```
plot(m0, 5)
```



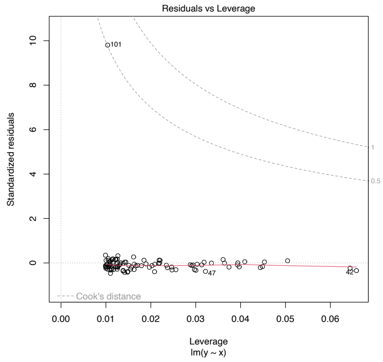
```
summary(m0)
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.10174 -0.30139 -0.00557  0.30949  1.30485
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.01103    0.05176   19.53   <2e-16 ***
## x            2.04727    0.04557   44.93   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

[30] This is the expected value for the leverage statistic $h_i$ if the linear model holds. However, the distribution of $h_i$ depends on the joint distribution of the predictors $(X_1, \ldots, X_p)$.

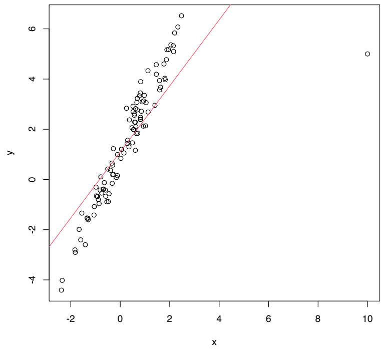
```
## Residual standard error: 0.5054 on 98 degrees of freedom
## Multiple R-squared:  0.9537, Adjusted R-squared:  0.9532
## F-statistic:  2018 on 1 and 98 DF,  p-value: < 2.2e-16

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



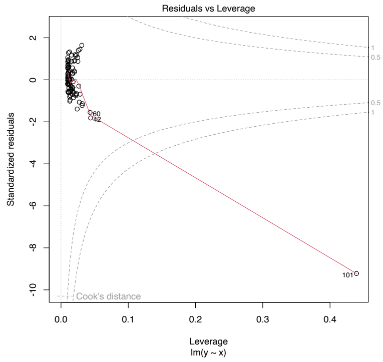
```
plot(m1, 5)
```



```
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)
plot(x, y)
abline(coef = m2$coefficients, col = 2)
```



```
plot(m2, 5)
```



```
summary(m2)
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.2423 -0.6126  0.0373  0.7864  2.1652
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.09830    0.13676   8.031 2.06e-12 ***
## x            1.31440    0.09082  14.473  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.339 on 99 degrees of freedom
## Multiple R-squared:  0.6791, Adjusted R-squared:  0.6758
## F-statistic: 209.5 on 1 and 99 DF,  p-value: < 2.2e-16
```

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

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

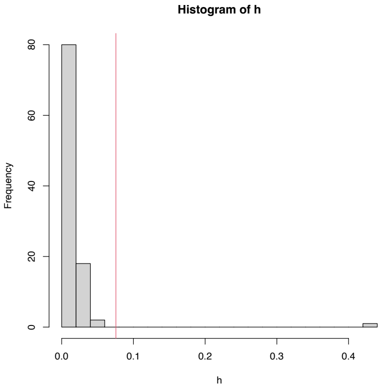
It can be seen that $\frac{1}{n} \leq h_i \leq 1$ and that the mean $\bar{h} = \frac{1}{n}\sum_{i=1}^n h_i = \frac{p+1}{n}$. This can be clearly seen in the case of simple linear regression, where the leverage statistic has the explicit form

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

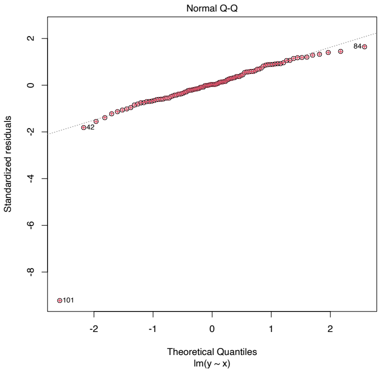
Interestingly, this expression shows that the leverage statistic is directly dependent on the distance to the center of the predictor. A precise threshold for determining when $h_i$ is *greatly exceeding* its expected value $\bar{h}$ can be given if the predictors are assumed to be jointly normal. In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

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

```
# Access leverage statistics
head(influence(model = m2, do.coef = FALSE)$hat)
##          1          2          3          4          5          6
## 0.01017449 0.01052333 0.01083766 0.01281244 0.01022209 0.03137304

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

# 1% most influential points
n <- length(x)
p <- 1
hist(h, breaks = 20)
abline(v = (qchisq(0.99, df = p) + 1) / n, col = 2)
```



```
# 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!

**Exercise 3.11.** The dataset `cpus.txt` (source, retrieved in September 2016) contains the transistor counts for the CPUs appeared in the time range 1971–2015. For this data, do the following:

a. Import conveniently the data and name it as `cpus`.
b. Show a scatterplot of `Transistor.count` vs. `Date.of.introduction` with a linear regression.
c. Are the assumptions verified in `Transistor.count ~ Date.of.introduction`? Which ones are more "problematic"?
d. Create a new variable, named `Log.Transistor.count`, containing the logarithm of `Transistor.count`.
e. Show a scatterplot of `Log.Transistor.count` vs. `Date.of.introduction` with a linear regression.
f. Are the assumptions verified in `Log.Transistor.count ~ Date.of.introduction`? Which ones are which are more "problematic"?
g. Regress `Log.Transistor.count ~ Date.of.introduction`.
h. Summarize the fit. What are the estimates $\hat\beta_0$ and $\hat\beta_1$? Is $\hat\beta_1$ close to $\frac{\log(2)}{2}$?
i. Compute the CI for $\beta_1$ at $\alpha = 0.05$. Is $\frac{\log(2)}{2}$ inside it? What happens at levels $\alpha = 0.10, 0.01$?
j. We want to forecast the average log-number of transistors for the CPUs to be released in 2024. Compute the adequate prediction and CI.
k. A new CPU design is expected for 2025. What is the range of log-number of transistors expected for it, at a 95% level of confidence?
l. Compute the ANOVA table for `Log.Transistor.count ~ Date.of.introduction`. Is $\beta_1$ significant?

**Exercise 3.12.** The dataset `gpus.txt` (source, retrieved in September 2016) contains the transistor counts for the GPUs appeared in the period 1997–2016. Repeat the previous analysis for this dataset.

**Exercise 3.13.** Analyze the bias induced by transforming the response, fitting a linear model, and then untransforming. For that, consider the relation

$$Y = \exp(\beta_0 + \beta_1 X + \varepsilon), \tag{3.8}$$

where $X \sim \mathcal{N}(0, 0.5)$ and $\varepsilon \sim \mathcal{N}(0, 1)$. Set $\beta_0 = -1$ and $\beta_1 = 1$. Then, follow these steps:

1. Compute the exact conditional distribution $Y \mid X = x$. *Hint*: use the lognormal distribution.
2. Based on Step 1, compute the true regression function $m(x) = \mathbb{E}[Y \mid X = x]$ exactly and plot it.
3. Simulate a sample $\{(X_i, Y_i)\}_{i=1}^n$ from (3.8), with $n = 200$.
4. Define $W := \log(Y)$ and fit the linear model $W = \beta_0 + \beta_1 X + \varepsilon$ from the previous sample. Are $(\beta_0, \beta_1)$ properly estimated?
5. Estimate $\mathbb{E}[W \mid X = 0]$ and $\mathbb{E}[W \mid X = 1]$ by $\hat{w}_0 = \hat{\beta}_0$ and $\hat{w}_1 = \hat{\beta}_0 + \hat{\beta}_1$, respectively.
6. Compute $\hat{y}_0 = \exp(\hat{w}_0)$ and $\hat{y}_1 = \exp(\hat{w}_1)$, the predictions with the transformed model of $\mathbb{E}[Y \mid X = 0]$ and $\mathbb{E}[Y \mid X = 1]$. Compare the estimates with $m(0)$ and $m(1)$.
7. Plot the fitted curve $\hat{y} = \exp(\hat{\beta}_0 + \hat{\beta}_1 x)$ and compare with $m$ in Step 2.
8. If $n$ is increased in Step 3, does the estimation in Step 6 improve? Summarize your takeaways from the exercise.

## 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 growth of the set of possible models as $p$ grows, the model space becomes 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 $X_1, \ldots, X_p$. PCA computes a new set of variables, the **principal components** $\Gamma_1, \ldots, \Gamma_p$, that contain the same information as $X_1, \ldots, X_p$ but expressed in a more convenient way. The goal of PCA is to retain only a limited number $\ell$, $1 \leq \ell \leq p$, of principal components that explain most of the information, therefore performing *dimension reduction*.

If $X_1, \ldots, X_p$ are *centered*,[31] then the principal components are *orthonormal linear combinations* of $X_1, \ldots, X_p$:

$$\Gamma_j := a_{1j} X_1 + a_{2j} X_2 + \cdots + a_{pj} X_p = \mathbf{a}_j^\top \mathbf{X}, \quad j = 1, \ldots, p, \tag{3.9}$$

[31] That is, $\mathbb{E}[X_j] = 0$, $j = 1, \ldots, p$. This is important since PCA is sensitive to the centering of the data.

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

$$\mathbf{a}_i^\top \mathbf{a}_j = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Remarkably, PCA computes the principal components in an *ordered* way: $\Gamma_1$ is the principal component that explains the most of the *information* (quantified as the variance) of $X_1, \ldots, X_p$, and then the explained information decreases monotonically down to $\Gamma_p$, the principal component that explains the least information. Precisely:

$$\mathbb{V}\text{ar}[\Gamma_1] \geq \mathbb{V}\text{ar}[\Gamma_2] \geq \ldots \geq \mathbb{V}\text{ar}[\Gamma_p]. \tag{3.10}$$

Mathematically, PCA computes the *spectral decomposition*[32] of the covariance matrix $\boldsymbol{\Sigma} := \mathbb{V}\text{ar}[\mathbf{X}]$:

$$\boldsymbol{\Sigma} = \mathbf{A}\boldsymbol{\Lambda}\mathbf{A}^\top,$$

where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_p)$ contains the eigenvalues of $\boldsymbol{\Sigma}$ and $\mathbf{A}$ is the orthogonal matrix[33] whose columns are the unit-norm eigenvectors of $\boldsymbol{\Sigma}$. The matrix $\mathbf{A}$ gives, thus, the coefficients of the orthonormal linear combinations:

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

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

$$\boldsymbol{\Gamma} = \mathbf{A}^\top(\mathbf{X} - \boldsymbol{\mu}), \tag{3.11}$$

where $\boldsymbol{\Gamma} = (\Gamma_1, \ldots, \Gamma_p)^\top$, $\mathbf{X} = (X_1, \ldots, X_p)^\top$, and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^\top$. Note that, because of (1.3) and (3.11),

$$\mathbb{V}\text{ar}[\boldsymbol{\Gamma}] = \mathbf{A}^\top\boldsymbol{\Sigma}\mathbf{A} = \mathbf{A}^\top\mathbf{A}\boldsymbol{\Lambda}\mathbf{A}^\top\mathbf{A} = \boldsymbol{\Lambda}. \tag{3.12}$$

Therefore, $\mathbb{V}\text{ar}[\Gamma_j] = \lambda_j$, $j = 1, \ldots, p$, and as a consequence (3.10) indeed holds.

Also, from (3.11), it is evident that we can express the random vector $\mathbf{X}$ in terms of $\boldsymbol{\Gamma}$:

$$\mathbf{X} = \boldsymbol{\mu} + \mathbf{A}\boldsymbol{\Gamma}, \tag{3.13}$$

which admits an insightful interpretation: $\boldsymbol{\Gamma}$ is an *uncorrelated*[34] vector that, once *rotated* by $\mathbf{A}$ and *translated* to the location $\boldsymbol{\mu}$, produces exactly $\mathbf{X}$. Therefore, $\boldsymbol{\Gamma}$ contains the same information as $\mathbf{X}$ but rearranged in a more convenient way, because the **principal components are centered and uncorrelated** between them:

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

[32] Recall that the covariance matrix is a real, symmetric, semi-positive definite matrix.

[33] Therefore, $\mathbf{A}^{-1} = \mathbf{A}^\top$.

[34] Since the variance-covariance matrix $\mathbb{V}\text{ar}[\boldsymbol{\Gamma}]$ is diagonal.

Due to the uncorrelation of $\boldsymbol{\Gamma}$, we can measure the *total variance* of $\boldsymbol{\Gamma}$ as $\sum_{j=1}^p \mathbb{V}\mathrm{ar}[\Gamma_j] = \sum_{j=1}^p \lambda_j$. Consequently, we can define the **proportion of variance explained** by the first $\ell$ principal components, $1 \leq \ell \leq p$, as

$$\frac{\sum_{j=1}^\ell \lambda_j}{\sum_{j=1}^p \lambda_j}.$$

In the *sample case*[35] where a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ is given and $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are unknown, $\boldsymbol{\mu}$ is replaced by the sample mean $\bar{\mathbf{X}}$ and $\boldsymbol{\Sigma}$ by the sample variance-covariance matrix $\mathbf{S} = \frac{1}{n}\sum_{i=1}^n (\mathbf{X}_i - \bar{\mathbf{X}})(\mathbf{X}_i - \bar{\mathbf{X}})^\top$. Then, the spectral decomposition of $\mathbf{S}$ is computed[36]. This gives $\hat{\mathbf{A}}$, the matrix of **loadings**, and produces the **scores** of the data:

$$\hat{\boldsymbol{\Gamma}}_1 := \hat{\mathbf{A}}^\top(\mathbf{X}_1 - \bar{\mathbf{X}}), \ldots, \hat{\boldsymbol{\Gamma}}_n := \hat{\mathbf{A}}^\top(\mathbf{X}_n - \bar{\mathbf{X}}).$$

The scores are *centered*, *uncorrelated*, and have sample variances in each vector's entry that are *sorted* in a decreasing way. The scores are the data coordinates with respect to the principal component basis.

The maximum number of principal components that can be determined from a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ is $\min(n-1, p)$, assuming that the matrix formed by $(\mathbf{X}_1 \cdots \mathbf{X}_n)$ is of full rank (i.e., if the rank is $\min(n, p)$). If $n \geq p$ and the variables $X_1, \ldots, X_p$ are such that only $r$ of them are linearly independent, then the maximum is $\min(n-1, r)$.

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

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

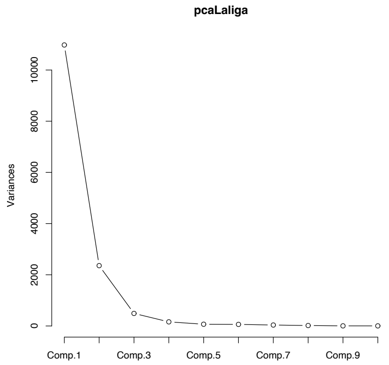
[35] Up to now, the exposition has been focused exclusively on the population case.

[36] Some care is needed here. The matrix $\mathbf{S}$ is obtained from linear combinations of the $n$ vectors $\mathbf{X}_1 - \bar{\mathbf{X}}, \ldots, \mathbf{X}_n - \bar{\mathbf{X}}$. Recall that these $n$ vectors are *not* linearly independent, as they are guaranteed to add $\mathbf{0}$, $\sum_{i=1}^n [\mathbf{X}_i - \bar{\mathbf{X}}] = \mathbf{0}$, so it is possible to express one perfectly on the rest. That implies that the $p \times p$ matrix $\mathbf{S}$ has rank smaller or equal to $n-1$. If $p \leq n-1$, then the matrix has full rank $p$ and it is invertible (excluding degenerate cases in which the $p$ variables are collinear). But if $p \geq n$, then $\mathbf{S}$ is singular and, as a consequence, $\lambda_j = 0$ for $j \geq n$. This implies that the principal components for those eigenvalues cannot be determined univocally.

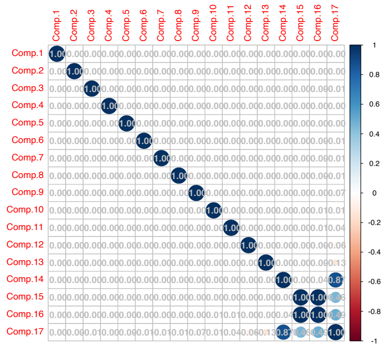
```
##  1st Qu.:483.8   1st Qu.: 7.0              1st Qu.: 97.0   1st Qu.:4.00   1st Qu.: 83.25
##  Median :530.5   Median :10.5              Median :108.5   Median :5.00   Median : 88.00
##  Mean   :517.6   Mean   :10.7              Mean   :106.2   Mean   :5.05   Mean   : 92.60
##  3rd Qu.:552.8   3rd Qu.:13.0              3rd Qu.:115.2   3rd Qu.:6.00   3rd Qu.:103.75
##  Max.   :654.0   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.

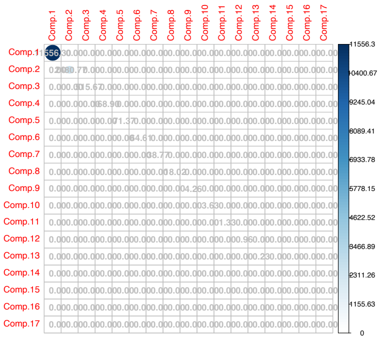
```
# PCA
pcaLaliga <- princomp(laliga, fix_sign = TRUE)
summary(pcaLaliga)
## Importance of components:
##                             Comp.1     Comp.2      Comp.3      Comp.4      Comp.5     Comp.6      Comp.7      Comp.8
## Standard deviation     104.7782561 48.5461449 22.13337511 12.66692413 8.234215354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

# We use fix_sign = TRUE so that the signs of the loadings are
# determined by the first element of each loading, set to be
# nonnegative. Otherwise, the signs could change for different OS /
# R versions yielding to opposite interpretations of the PCs

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



```
# 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

# Alternatively: 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) # By default, cov() computes the
# quasi-variance-covariance matrix that divides by n - 1, but PCA and princomp
# consider the sample variance-covariance matrix that divides by n
A <- eig$vectors

# Same eigenvalues
pcaLaliga$sdev^2 - eig$values
##        Comp.1        Comp.2        Comp.3        Comp.4        Comp.5        Comp.6        Comp.7        Comp.8
## -5.456968e-12 -1.364242e-12  0.000000e+00 -4.263256e-13 -2.415845e-13 -1.278977e-13 -3.552714e-14 -4.263256e-14
##        Comp.9       Comp.10       Comp.11       Comp.12       Comp.13       Comp.14       Comp.15       Comp.16
## -2.025047e-13 -7.549517e-14 -1.088019e-14 -3.108624e-15 -1.626477e-14 -3.685726e-15  1.036604e-15 -2.025796e-13
##       Comp.17
##  1.435137e-12

# The eigenvectors (the a_j vectors) are the column vectors in $loadings
pcaLaliga$loadings
##
## Loadings:
##                           Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10 Comp.11 Comp.12
## Points                     0.125         0.497  0.195  0.139  0.340         0.425         0.379   0.129   0.166
## Wins                                     0.184                0.175         0.134 -0.198                  0.139
## Draws                                                  0.101 -0.186                0.608  0.175   0.185  -0.251
## Loses                                   -0.129        -0.114               -0.157 -0.410 -0.243  -0.166   0.112
## Goals.scored               0.181         0.251 -0.186 -0.169  0.399  0.335 -0.603 -0.155  0.129   0.289  -0.230
## Goals.conceded                          -0.471 -0.493 -0.277  0.257  0.280  0.441 -0.118  0.297
## 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                                                           -0.350  0.258  0.378  -0.661   0.456
## Assistances                0.148         0.198        -0.173  0.362  0.216  0.215  0.356 -0.685  -0.265
## Fouls.made                -0.480  0.844  0.166               -0.110
## Matches.without.conceding                0.151  0.129               -0.182  0.176 -0.369         -0.376  -0.411
## Yellow.cards              -0.141  0.144 -0.363  0.113  0.225  0.637 -0.550 -0.126  0.156
## Red.cards                                                                         -0.123 -0.157   0.405   0.666
## Offsides                         0.108  0.202 -0.696  0.647        -0.106
##                           Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
## Points                      0.138                   0.240   0.345
## Wins                        0.147                          -0.904
## Draws                      -0.304                   0.554  -0.215
## Loses                       0.156                   0.794   0.130
## Goals.scored               -0.153
## Goals.conceded
## Percentage.scored.goals           -0.760  -0.650
## Percentage.conceded.goals          0.650  -0.760
## Shots
## Shots.on.goal
## Penalties.scored           -0.114
## Assistances                -0.102
## Fouls.made
## Matches.without.conceding -0.664
## Yellow.cards
## Red.cards                  -0.587
## Offsides
##
##                 Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10 Comp.11 Comp.12 Comp.13 Comp.14
## SS loadings      1.000  1.000  1.000  1.000  1.000  1.000  1.000  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  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  0.529    0.588    0.647    0.706    0.765    0.824
##                 Comp.15 Comp.16 Comp.17
## SS loadings       1.000   1.000   1.000
## Proportion Var    0.059   0.059   0.059
## Cumulative Var    0.882   0.941   1.000

# The scores is the representation of the data in the principal components -
# it has the same information as laliga
head(round(pcaLaliga$scores, 4))
##                    Comp.1   Comp.2  Comp.3   Comp.4  Comp.5  Comp.6  Comp.7  Comp.8  Comp.9 Comp.10 Comp.11 Comp.12
## Barcelona        242.2392 -21.5810 25.3801 -17.4375 -7.1797  9.0814 -5.5920 -7.3615 -0.3716  1.7161  0.0265 -1.0948
## Real Madrid      313.6026  63.2024 -8.7570   8.5558  0.7119  0.2221  6.7035  2.4456  1.8388 -2.9661 -0.1345  0.3410
## Atlético Madrid   45.9939  -0.6466 38.5410  31.3888  3.9163  3.2904  0.2432  5.0913 -3.0444  2.0975  0.6771 -0.3986
## Villarreal       -96.2201 -42.9329 50.0036 -11.2420 10.4733  2.4294 -3.0183  0.1958  1.2106 -1.7453  0.1351 -0.5735
## Athletic          14.5173 -16.1897 18.8840  -0.4122 -5.6491 -6.9330  8.0653  2.4783 -2.6921  0.8950  0.1542  1.4715
## Celta            -13.0748   6.7925  5.2271  -9.0945  6.1265 11.8795 -2.6148  6.9707  3.0826  0.3130  0.0860  1.9159
##                  Comp.13 Comp.14 Comp.15 Comp.16 Comp.17
## Barcelona         0.1516 -0.0010  -2e-04       0       0
## Real Madrid      -0.0332  0.0015   4e-04       0       0
## Atlético Madrid  -0.1809  0.0005  -1e-04       0       0
## Villarreal       -0.5894 -0.0001  -2e-04       0       0
## Athletic          0.1109 -0.0031  -3e-04       0       0
## Celta             0.3722 -0.0018   3e-04       0       0

# Uncorrelated
corrplot::corrplot(cor(pcaLaliga$scores), addCoef.col = "gray")

# Caution! What happened in the last columns? What happened is that the
# variance for the last principal components is close to zero (because there
# are linear dependencies on the variables; e.g., Points, Wins, Loses, Draws),
```



```
# so the computation of the correlation matrix becomes unstable for those
# variables (a 0/0 division takes place)

# Better to inspect the covariance matrix
corrplot::corrplot(cov(pcaLaliga$scores), addCoef.col = "gray", is.corr = FALSE)
```



```
# 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)
max(abs(abs(pcaLaliga$scores) - abs(scores)))
## [1] 2.278507e-11

# Reconstruct the data from all the principal components
head(
  sweep(pcaLaliga$scores %*% t(pcaLaliga$loadings), 2, pcaLaliga$center, "+")
  )
##                  Points Wins Draws Loses Goals.scored Goals.conceded Percentage.scored.goals Percentage.conceded.goals
## Barcelona            91   29     4     5          112             29                    2.95                      0.76
## Real Madrid          90   28     6     4          110             34                    2.89                      0.89
## Atlético Madrid      88   28     4     6           63             18                    1.66                      0.47
## Villarreal           64   18    10    10           44             35                    1.16                      0.92
## Athletic             62   18     8    12           58             45                    1.53                      1.18
## Celta                60   17     9    12           51             59                    1.34                      1.55
##                  Shots Shots.on.goal Penalties.scored Assistances Fouls.made Matches.without.conceding Yellow.cards
## Barcelona          600           277               11          79        385                        18           66
## Real Madrid        712           299                6          90        420                        14           72
## Atlético Madrid    481           186                1          49        503                        24           91
## Villarreal         346           135                3          32        534                        17          100
## Athletic           450           178                3          42        502                        13           84
## Celta              442           170                4          43        528                        10          116
##                  Red.cards Offsides
## Barcelona                1      120
## Real Madrid              5      114
## Atlético Madrid          3       84
## Villarreal               4      106
## Athletic                 5       92
## Celta                    6      103
```

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.[37] Therefore, when variables with different ranges are mixed, the variability of one may dominate the other merely due to a scale artifact. To prevent this, we **standardize** the dataset prior to doing a PCA.

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

[37] 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!

```
##                              Comp.16 Comp.17
## Standard deviation     9.256461e-09       0
## Proportion of Variance 5.040122e-18       0
## Cumulative Proportion  1.000000e+00       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)
```

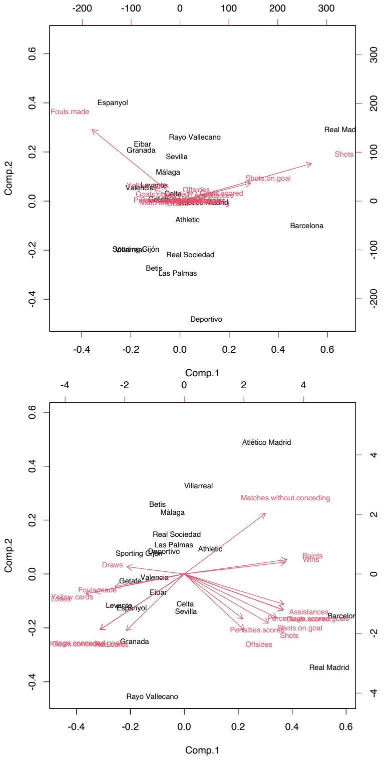


Figure 3.31: Biplots for `laliga` dataset, with unstandardized and standardized data, respectively.

The *biplot*[38] provides a powerful and succinct way of displaying the relevant information contained in the first two principal components. It shows:

1. The **scores** $\{(\hat{\Gamma}_{i1}, \hat{\Gamma}_{i2})\}_{i=1}^n$ by *points* (with optional text labels, depending if there are case names). These are the representations of the data in the first two principal components.
2. The **loadings** $\{(\hat{a}_{j1}, \hat{a}_{j2})\}_{j=1}^p$ by the *arrows*, centered at $(0,0)$. These are the representations of the variables in the first two principal components.

Let's examine the *population*[39] arrow associated to the variable $X_j$. $X_j$ is expressed in terms of $\Gamma_1$ and $\Gamma_2$ by the loadings $a_{j1}$ and $a_{j2}$ [40]:

$$X_j = a_{j1}\Gamma_1 + a_{j2}\Gamma_2 + \cdots + a_{jp}\Gamma_p \approx a_{j1}\Gamma_1 + a_{j2}\Gamma_2.$$

The weights $a_{j1}$ and $a_{j2}$ have the same sign as $\mathrm{Cor}(X_j, \Gamma_1)$ and $\mathrm{Cor}(X_j, \Gamma_2)$, 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 $\Gamma_1$. Analogous if the arrow points left.
- If the arrow is *approximately vertical* ($a_{j1} \approx 0$), there is *uncorrelation* between $X_j$ and $\Gamma_1$.

Analogously:

- If the arrow *points up* ($a_{j2} > 0$), there is *positive correlation* between $X_j$ and $\Gamma_2$. Analogous if the arrow points down.
- If the arrow is *approximately horizontal* ($a_{j2} \approx 0$), there is *uncorrelation* between $X_j$ and $\Gamma_2$.

In addition, the **magnitude of the arrow** informs us about the strength of the **correlation** of $X_j$ with $(\Gamma_1, \Gamma_2)$.

The biplot also informs about the direct relation between variables, at sight of their expressions in $\Gamma_1$ and $\Gamma_2$. The **angle** of the arrows of variable $X_j$ and $X_k$ gives an **approximation to the correlation** between them, $\mathrm{Cor}(X_j, X_k)$:

- 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.

[38] The biplot is implemented in R by `biplot.princomp` (or by `biplot` applied to a `princomp` object). This function applies an internal scaling of the scores and variables to improve the visualization (see `?biplot.princomp`), which can be disabled with the argument `scale = 0`.

[39] For the sample version, replace the variable $X_j$ by its sample $X_{1j}, \ldots, X_{nj}$, the loadings $a_{ji}$ by their estimates $\hat{a}_{ji}$, and the principal component $\Gamma_j$ by the scores $\{\hat{\Gamma}_{ij}\}_{i=1}^n$.

[40] Observe that both $X_j = a_{j1}\Gamma_1 + a_{j2}\Gamma_2 + \cdots + a_{jp}\Gamma_p$ and $\Gamma_j = a_{1j}X_1 + a_{2j}X_2 + \cdots + a_{pj}X_p$, $j = 1, \ldots, p$, hold simultaneously due to the orthogonality of $\mathbf{A}$.

The **insights** obtained from the *approximate* correlations between variables are **as precise as the percentage of variance explained** by the first two principal components.

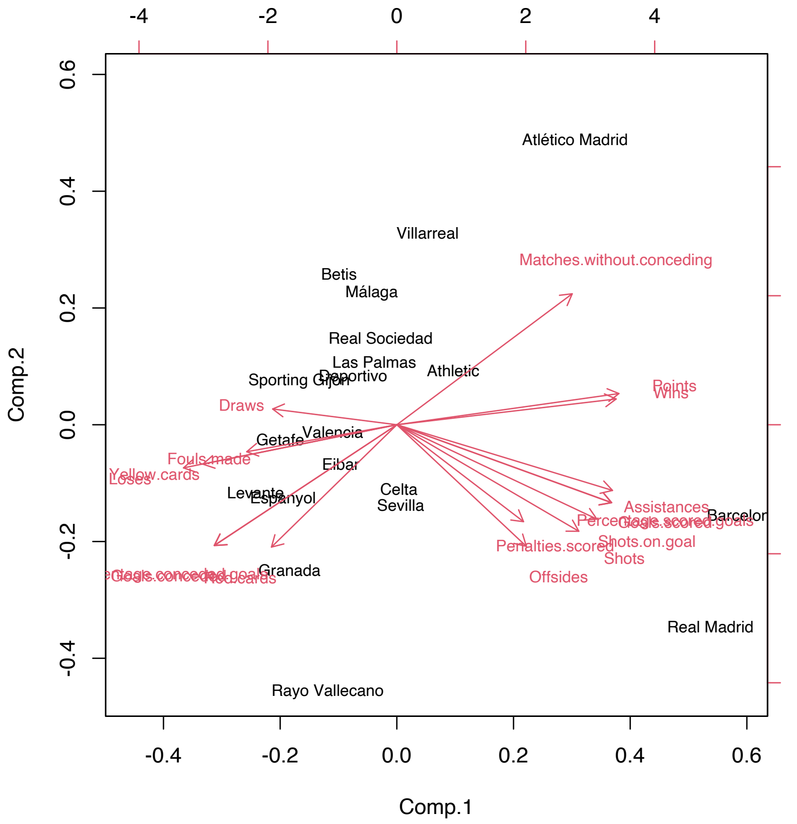


Figure 3.32: Biplot for `laliga` dataset. The interpretations of the first two principal components are driven by the signs of the variables inside them (directions of the arrows) and by the strength of their correlations with the variables (length of the arrows). The scores of the data serve also to cluster similar observations according to their proximity in the biplot.

Some interesting insights in La Liga 2015/2016 biplot, obtained from the previous remarks, are:[41]

- The first component can be regarded as the *performance of a team* during the season. It is positively correlated with `Wins`, `Points`, etc. and negatively correlated with `Draws`, `Loses`, `Yellow.cards`, etc. 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.
- The second component can be seen as the *efficiency of a team* (obtaining points with little participation in the game). Using this interpretation we can see that Atlético Madrid and Villareal were the most efficient teams and that Rayo Vallecano and Real Madrid were the most inefficient.
- `Offsides` is approximately uncorrelated with `Red.cards` and `Matches.without.conceding`.

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

```
pca3d::pca3d(pcaLaligaStd, show.labels = TRUE, biplot = TRUE)
rgl::rglwidget()
```

[41] The first two principal components of the dataset explain the 78% of the variability of the data. Hence, the insights obtained from the biplot should be regarded as "78%-accurate".

Finally, the `biplot` function allows us to construct the biplot using two arbitrary principal components using the `choices` argument. Keep in mind than these pseudo-biplots will explain a lower proportion of variance than the default `choices = 1:2`:

```
biplot(pcaLaligaStd, choices = c(1, 3)) # 0.7138 proportion of variance
biplot(pcaLaligaStd, choices = c(2, 3)) # 0.2180 proportion of variance
```

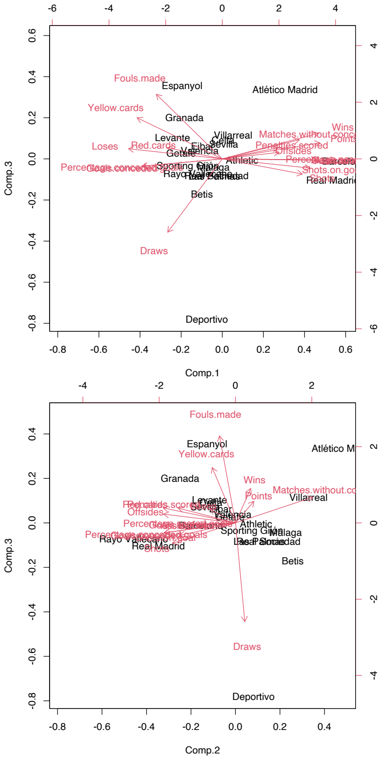


Figure 3.33: Pseudo-biplots for `laliga` dataset based on principal components 1 and 3, and principal components 2 and 3, respectively.

**Exercise 3.14.** At the sight of the previous plots, can you think about an interpretation for the third principal component?

### 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 $\Gamma_1, \ldots, \Gamma_\ell$ obtained from the predictors $X_1, \ldots, X_p$, where $\ell < p$. 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* $Y$.[42] Therefore, we look for fitting the linear model[43]

$$Y = \alpha_0 + \alpha_1\Gamma_1 + \cdots + \alpha_\ell\Gamma_\ell + \varepsilon. \tag{3.14}$$

The main advantages of PCR are two:

1. **Multicollinearity** is **avoided** by design: $\Gamma_1, \ldots, \Gamma_\ell$ are uncorrelated between them.
2. There are **less coefficients to estimate** ($\ell$ instead of $p$), 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 directly related with the predictors*, but with the principal components. Hence, the interpretability of a given coefficient in the regression model is tied to 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.

The first point is elaborated next. The PCR model (3.14) can be seen as a linear model expressed in terms of the original predictors. To make this point clearer, let's re-express (3.14) as

$$Y = \alpha_0 + \mathbf{\Gamma}_{1:\ell}^\top \boldsymbol{\alpha}_{1:\ell} + \varepsilon, \tag{3.15}$$

where the subindex $1:\ell$ denotes the inclusion of the vector entries from 1 to $\ell$. Now, we can express the PCR problem (3.15) in terms of the original predictors:[44]

$$\begin{aligned}
Y &= \alpha_0 + (\mathbf{A}_{1:\ell}^\top(\mathbf{X} - \boldsymbol{\mu}))^\top \boldsymbol{\alpha}_{1:\ell} + \varepsilon \\
&= (\alpha_0 - \boldsymbol{\mu}^\top \mathbf{A}_{1:\ell}\boldsymbol{\alpha}_{1:\ell}) + \mathbf{X}^\top \mathbf{A}_{1:\ell}\boldsymbol{\alpha}_{1:\ell} + \varepsilon \\
&= \gamma_0 + \mathbf{X}^\top \gamma_{1:p} + \varepsilon,
\end{aligned}$$

[42] This is certainly not always true, but it is often the case. See Figure 3.6.3.

[43] For the sake of simplicity, we considering the exposition the *first* $\ell$ principal components, but obviously other combinations of $\ell$ principal components are possible.

[44] Since we know by (3.11) that $\mathbf{\Gamma} = \mathbf{A}^\top(\mathbf{X} - \boldsymbol{\mu})$, where $\mathbf{A}$ is the $p \times p$ matrix of loadings.

where $\mathbf{A}_{1:\ell}$ represents the $\mathbf{A}$ matrix with only its first $\ell$ columns and

$$\gamma_0 := \alpha_0 - \boldsymbol{\mu}^\top \mathbf{A}_{1:\ell}\boldsymbol{\alpha}_{1:\ell}, \quad \gamma_{1:p} := \mathbf{A}_{1:\ell}\boldsymbol{\alpha}_{1:\ell}. \tag{3.16}$$

In other words, the coefficients $\boldsymbol{\alpha}_{1:\ell}$ of the PCR done with $\ell$ principal components in (3.14) translate into the coefficients (3.16) of the linear model based on the $p$ original predictors:

$$Y = \gamma_0 + \gamma_1 X_1 + \cdots + \gamma_p X_p + \varepsilon.$$

In the sample case, we have that

$$\hat{\gamma}_0 = \hat{\alpha}_0 - \bar{\mathbf{X}}^\top \hat{\mathbf{A}}_{1:\ell}\hat{\boldsymbol{\alpha}}_{1:\ell}, \quad \hat{\gamma}_{1:p} = \hat{\mathbf{A}}_{1:\ell}\hat{\boldsymbol{\alpha}}_{1:\ell}. \tag{3.17}$$

Notice that $\hat{\boldsymbol{\gamma}}$ is **not the least squares estimator** performed with the original predictors, that we use to denote by $\hat{\boldsymbol{\beta}}$. $\hat{\boldsymbol{\gamma}}$ contains the coefficients of the PCR that are associated to the *original predictors*. Consequently, $\hat{\boldsymbol{\gamma}}$ is useful for the interpretation of the linear model produced by PCR, as it can be interpreted in the same way $\hat{\boldsymbol{\beta}}$ was.[45]

Finally, remember that the **usefulness** of PCR relies on how well we are able to reduce the dimensionality[46] of the predictors and the veracity of the assumption that the $\ell$ 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 common objective is to predict `Points` using the remaining variables (excluding those directly related: `Wins`, `Draws`, `Loses`, and `Matches.without.conceding`) in order to quantify, explain, and predict the final points of a team from its performance.

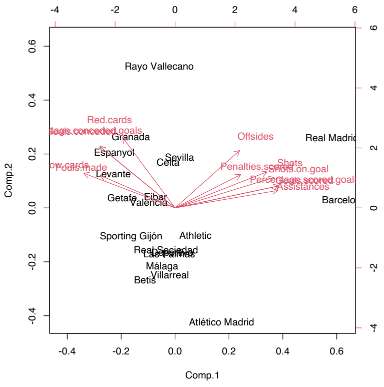
The first approach combines the use of the `princomp` and `lm` functions. Its strong points are that is both able to predict and explain, and is linked with techniques we have employed so far. The weak point is that it requires extra coding.

```
# 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:
```

[45] For example, thanks to $\hat{\boldsymbol{\gamma}}$ we know that the estimated conditional response of $Y$ precisely increases $\hat{\gamma}_j$ units for a marginal unit increment in $X_j$.

[46] If $\ell = p$, then PCR is equivalent to least squares estimation.

```
##                             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
## Assistances                  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 <- MASS::stepAIC(mod, k = log(nrow(laliga)), trace = 0)
summary(modBIC) # Better, but still unsatisfactory
##
## Call:
## lm(formula = Points ~ Goals.scored + Goals.conceded + Percentage.scored.goals +
##     Percentage.conceded.goals + Shots + Assistances + 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 .
## Assistances                 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

# Also, huge multicollinearity
car::vif(modBIC)
##              Goals.scored            Goals.conceded   Percentage.scored.goals Percentage.conceded.goals
##              77998.044760              22299.952547              76320.612579              22322.307151
##                     Shots               Assistances              Yellow.cards
##                  6.505748                 32.505831                  3.297224

# A quick way of removing columns without knowing its position
laligaRed <- subset(laliga, select = -c(Points, Wins, Draws, Loses,
                                        Matches.without.conceding))
# PCA without Points, Wins, Draws, Loses, and Matches.without.conceding
pcaLaligaRed <- princomp(x = laligaRed, cor = TRUE, fix_sign = TRUE)
summary(pcaLaligaRed) # l = 3 gives 86% of variance explained
## Importance of components:
##                           Comp.1    Comp.2     Comp.3    Comp.4     Comp.5    Comp.6      Comp.7      Comp.8
## Standard deviation     2.7437329 1.4026745 0.91510249 0.8577839 0.65747209 0.5310954 0.332556029 0.263170555
## Proportion of Variance 0.6273392 0.1639580 0.06978438 0.0613161 0.03602246 0.0235052 0.009216126 0.005771562
## Cumulative Proportion  0.6273392 0.7912972 0.86108155 0.9223977 0.95842012 0.9819253 0.991141438 0.996913000
##                             Comp.9     Comp.10      Comp.11      Comp.12
## Standard deviation     0.146091551 0.125252621 3.130311e-03 1.801036e-03
## Proportion of Variance 0.001778562 0.001307352 8.165704e-07 2.703107e-07
## Cumulative Proportion  0.998691562 0.999998913 9.999997e-01 1.000000e+00

# Interpretation of PC1 and PC2
biplot(pcaLaligaRed)
```

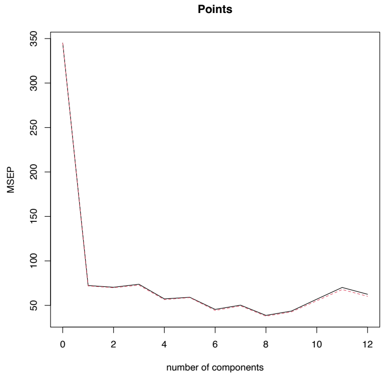


```
# PC1: attack performance of the team

# Create a new dataset with the response + principal components
laligaPCA <- data.frame("Points" = laliga$Points, pcaLaligaRed$scores)

# Regression on all the principal components
modPCA <- lm(Points ~ ., data = laligaPCA)
summary(modPCA) # Predictors clearly significant -- 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
car::vif(modPCA) # No problems at all
##  Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7  Comp.8  Comp.9 Comp.10 Comp.11 Comp.12
##       1       1       1       1       1       1       1       1       1       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 with each original predictor (gamma)
alpha <- modPCA3$coefficients
gamma <- pcaLaligaRed$loadings[, 1:3] %*% alpha[-1] # Slopes
gamma <- c(alpha[1] - pcaLaligaRed$center %*% gamma, gamma) # Intercept
names(gamma) <- c("Intercept", rownames(pcaLaligaRed$loadings))
gamma
##                 Intercept              Goals.scored            Goals.conceded   Percentage.scored.goals
##               -44.2288551                 1.7305124                -3.4048178                 1.7416378
## Percentage.conceded.goals                     Shots             Shots.on.goal          Penalties.scored
##                -3.3944235                 0.2347716                 0.8782162                 2.6044699
##               Assistances                Fouls.made              Yellow.cards                 Red.cards
##                 1.4548813                -0.1171732                -1.7826488                -2.6423211
##                  Offsides
##                 1.3755697

# We can overpenalize to have a simpler model -- also one single
# principal component does quite well
modPCABIC <- MASS::stepAIC(modPCA, k = 2 * log(nrow(laliga)), trace = 0)
summary(modPCABIC)
##
## 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 that 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) # Centered 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(modPCABIC, newdata = newScores, interval = "prediction")
##                  fit      lwr      upr
## Barcelona   93.64950 80.35115 106.9478
## Real Madrid 90.05622 77.11876 102.9937
```

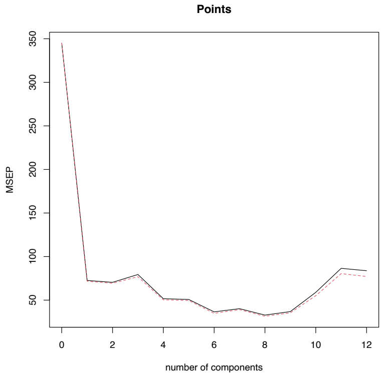
```
# Reality
laliga[1:2, 1]
## [1] 91 90
```

The second approach employs the function `pls::pcr` and is more direct, yet less connected with the techniques we have seen so far. It employs a model object that is different from the `lm` object and, as a consequence, functions like `summary`, `BIC`, `MASS::stepAIC`, or `plot` will not work properly. This implies that inference, model selection, and model diagnostics are not so straightforward. In exchange, `pls::pcr` allows for model fitting in an easier way and model selection through the use of cross-validation. In overall, this is a more *pure predictive* approach than *predictive and explicative*.

```
# Create a dataset without the problematic predictors and with the response
laliga Red2 <- subset(laliga, select = -c(Wins, Draws, Loses,
                                         Matches.without.conceding))

# 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 predictors are scaled internally before computing
# 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  8 comps  9 comps  10 comps  11 comps  12 comps
## X          62.73    79.13    86.11    92.24    95.84    98.19    99.11    99.69    99.87    100.00       100    100.00
## Points     80.47    84.23    87.38    90.45    90.99    93.94    94.36    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 that 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"     "projection"    "Xmeans"        "Ymeans"
##  [8] "fitted.values" "residuals"     "Xvar"          "Xtotvar"       "fit.time"      "ncomp"         "method"
## [15] "center"        "scale"         "call"          "terms"         "model"

# The coefficients of the original predictors (gammas), not of the components!
modPcr$coefficients[, , 12]
##              Goals.scored            Goals.conceded   Percentage.scored.goals Percentage.conceded.goals
##             -602.85050765             -383.07010184              600.61255371              374.38729000
##                     Shots             Shots.on.goal         Penalties.scored               Assistances
##               -8.27239221                0.88174787               -2.14313238               24.42240486
##                Fouls.made              Yellow.cards                 Red.cards                  Offsides
##               -2.96044265                5.51983512                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  7 comps  8 comps  9 comps  10 comps  11 comps
## CV           18.57    8.505    8.390    8.588    7.571    7.688    6.743     7.09    6.224    6.603     7.547     8.375
## adjCV        18.57    8.476    8.356    8.525    7.513    7.663    6.655     7.03    6.152    6.531     7.430     8.236
##        12 comps
## CV        7.905
## adjCV     7.760
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X         62.73    79.13    86.11    92.24    95.84    98.19    99.11    99.69    99.87    100.00       100    100.00
## Points    80.47    84.23    87.38    90.45    90.99    93.94    94.36    96.17    96.32     96.84        97     97.95

# View cross-validation Mean Squared Error in Prediction
validationplot(modPcrCV1, val.type = "MSEP") # l = 8 gives the minimum CV
```



```
# The black is the CV loss, the dashed red line is the adjCV loss, a bias
# corrected version of the MSEP (not described in the notes)

# Selecting the number of components to retain by 10-fold Cross-Validation
# (k = 10 is the default, this can be changed with the argument segments)
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  7 comps  8 comps  9 comps  10 comps  11 comps
## CV           18.57    8.520    8.392    8.911    7.187    7.124    6.045    6.340    5.728    6.073     7.676     9.300
## adjCV        18.57    8.464    8.331    8.768    7.092    7.043    5.904    6.252    5.608    5.953     7.423     8.968
##        12 comps
## CV        9.151
## adjCV     8.782
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X         62.73    79.13    86.11    92.24    95.84    98.19    99.11    99.69    99.87    100.00       100    100.00
## Points     80.47    84.23    87.38    90.45    90.99    93.94    94.36    96.17    96.32     96.84         97     97.95
validationplot(modPcrCV10, val.type = "MSEP") # l = 8 gives the minimum CV
```



`pcr()` 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 `pcr()` **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 (3.17) are obtained by **dividing** the `coefficients` of `pcr()` by $\sqrt{\frac{n}{n-1}}$.

The next chunk of code illustrates the previous warning.

```
# Equality of loadings from princomp() and pcr()
max(abs(abs(pcaLaligaRed$loadings[, 1:3]) - abs(modPcr$loadings[, 1:3])))
## [1] 2.664535e-15

# Equality of scores from princomp() and pcr() (with the same standardization)
max(abs(abs(pcaLaligaRed$scores[, 1:3]) -
          abs(modPcr$scores[, 1:3] * sqrt(n / (n - 1)))))
## [1] 9.298118e-15

# Equality of the gamma coefficients obtained previously for 3 PCA
# (with the same standardization)
modPcr$coefficients[, , 3] / sqrt(n / (n - 1))
##             Goals.scored            Goals.conceded   Percentage.scored.goals Percentage.conceded.goals
##                1.7305124                -3.4048178                 1.7416378                -3.3944235
##                    Shots             Shots.on.goal          Penalties.scored               Assistances
##                0.2347716                 0.8782162                 2.6044699                 1.4548813
##               Fouls.made              Yellow.cards                 Red.cards                  Offsides
##               -0.1171732                -1.7826488                -2.6423211                 1.3755697
gamma[-1]
##             Goals.scored            Goals.conceded   Percentage.scored.goals Percentage.conceded.goals
##                1.7305124                -3.4048178                 1.7416378                -3.3944235
##                    Shots             Shots.on.goal          Penalties.scored               Assistances
##                0.2347716                 0.8782162                 2.6044699                 1.4548813
##               Fouls.made              Yellow.cards                 Red.cards                  Offsides
##               -0.1171732                -1.7826488                -2.6423211                 1.3755697

# Coefficients associated with 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 $\ell$ by **cross-validation** attempts to minimize the *Mean Squared Error in Prediction* (MSEP) or, equivalently, the *Root MSEP* (RMSEP) of the model.[47] This is a **Swiss army knife 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 $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$, **leave-one-out cross-validation** considers the tuning parameter

$$\hat{\lambda}_{\mathrm{CV}} := \arg\min_{\lambda\geq 0} \sum_{i=1}^n (Y_i - \hat{m}_{\lambda,-i}(\mathbf{X}_i))^2, \tag{3.18}$$

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 $F_1, \ldots, F_k$ 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:

$$\hat{\lambda}_{k\text{-CV}} := \arg\min_{\lambda\geq 0} \sum_{j=1}^k \sum_{i\in F_j} (Y_i - \hat{m}_{\lambda,-F_j}(\mathbf{X}_i))^2, \tag{3.19}$$

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 $k$-fold cross-validation is **more general than leave-one-out cross-validation**, since the latter is a particular case of the former with $k = n$.

[47] Alternatively, the “in Prediction” part of the latter terms is dropped and they are just referred to as the MSE and RMSE.

$k$-fold cross validation with $k < n$ 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, in a convenient way, the results of *several* $k$-fold cross-validations done in different random random partitions. This problem is *not* present in leave-one-out cross-validation (where $k = n$).

Inference in PCR can be carried out as in the standard linear model. The key point is to realize that it is about the coefficients $\boldsymbol{\alpha}_{0:\ell}$ associated with the $\ell$ principal components, and that it can be carried out by the `summary()` function on the output of `lm()`. The coefficients $\boldsymbol{\alpha}_{0:\ell}$ are the ones estimated by least squares when considering the scores of the $\ell$ principal components as the predictors. This transformation of the predictors does not affect the validity of the inferential results of Section 2.4 (derived conditionally on the predictors). But recall that this **inference is not about** $\boldsymbol{\gamma}$.

Inference in PCR is based on the assumptions of the linear model being satisfied for the considered principal components. The evaluation of the assumptions can be done using the exact same tools described in Section 3.5. However, keep in mind that **PCA is a linear transformation of the data**. Therefore:

- If the **linearity** assumption fails for the predictors $X_1, \ldots, X_p$, then it will also likely fail for $\Gamma_1, \ldots, \Gamma_\ell$, since the transformation will not introduce nonlinearities able to capture the nonlinear effects.
- Similarly, if the **homoscedasticity, normality, or independence** assumptions fail for $X_1, \ldots, X_p$, then they will also likely fail for $\Gamma_1, \ldots, \Gamma_\ell$.

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 regression

PCR works by replacing the predictors $X_1, \ldots, X_p$ by a set of principal components $\Gamma_1, \ldots, \Gamma_\ell$, under the hope that these directions, that explain most of the variability of the predictors, are *also* the best directions for predicting the response $Y$. While this is a reasonable belief, it is not a guaranteed fact. *Partial Least Squares Regression* (PLSR) precisely tackles this point with the idea of regressing the response $Y$ on a set of new variables $P_1, \ldots, P_\ell$ that are constructed with the objective of *predicting $Y$ from $X_1, \ldots, X_p$ in the best linear way*.

As with PCA, the idea is to find linear combinations of the predictors $X_1, \ldots, X_p$, that is, to have:

$$P_1 := \sum_{j=1}^p a_{1j} X_j, \ldots, P_p := \sum_{j=1}^p a_{pj} X_j. \tag{3.20}$$

These new predictors are called *PLS components*.[48]

The question is how to choose the coefficients $a_{kj}$, $j, k = 1, \ldots, p$. PLSR does it by placing the **most weight** in the **predictors that are most strongly correlated with** $Y$ and in such a way that the resulting $P_1, \ldots, P_p$ are **uncorrelated** between them. After standardizing the variables, for $P_1$ this is achieved by setting $a_{1j}$ equal to the theoretical slope coefficient of regressing $Y$ into $X_j$, that is:[49]

$$a_{1j} := \frac{\mathrm{Cov}[X_j, Y]}{\mathbb{V}\mathrm{ar}[X_j]},$$

where $a_{1j}$ stems from $Y = a_0 + a_{1j} X_j + \varepsilon$.

The second partial least squares direction, $P_2$, is computed in a similar way, but *once the linear effects of $P_1$ on $X_1, \ldots, X_p$ are removed*. This is achieved by:

1. Regressing each of $X_1, \ldots, X_p$ on $P_1$. That is, fit the $p$ simple linear models

$$X_j = \alpha_0 + \alpha_{1j} P_1 + \varepsilon_j, \quad j = 1, \ldots, p.$$

2. Regress $Y$ on each of the $p$ random errors $\varepsilon_1, \ldots, \varepsilon_p$ [50] from the above regressions, yielding

$$a_{2j} := \frac{\mathrm{Cov}[\varepsilon_j, Y]}{\mathbb{V}\mathrm{ar}[\varepsilon_j]},$$

where $a_{2j}$ stems from $Y = a_0 + a_{2j} \varepsilon_j + \varepsilon$.

The coefficients for $P_j$, $j > 2$, are computed[51] by iterating the former process. Once $P_1, \ldots, P_\ell$ are obtained, then PLSR proceeds as PCR and fits the model

$$Y = \beta_0 + \beta_1 P_1 + \cdots + \beta_\ell P_\ell + \varepsilon.$$

The implementation of PLSR can be done by the function `pls::plsr`, which has an analogous syntax to `pls::pcr`.

[48] Compare (3.20) with (3.9).

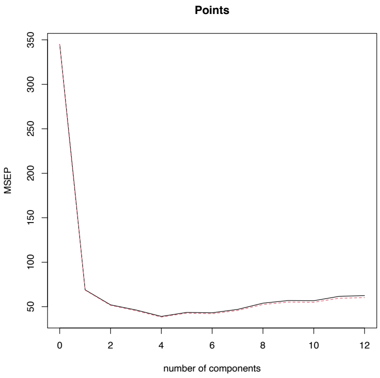
[49] Recall (2.3) for the sample version.

[50] They play the role of $X_1, \ldots, X_p$ in the computation of $P_1$ and can be regarded as $X_1, \ldots, X_p$ after *filtering* the linear information explained by $P_1$.

[51] Of course, in practice, the computations need to be done in terms of the sample versions of the presented population versions.

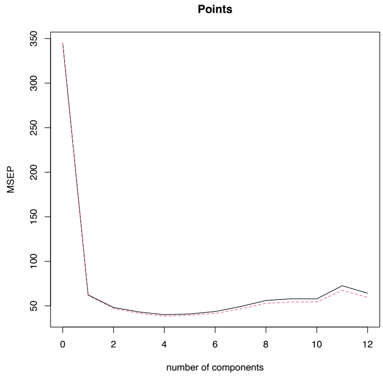
```
# Simple call to plsr -- very similar to pcr
modPlsr <- plsr(Points ~ ., data = laligaRed2, scale = TRUE)

# The summary of the model
summary(modPlsr)
## Data:    X dimension: 20 12
##  Y dimension: 20 1
## Fit method: kernelpls
## Number of components considered: 12
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X         62.53    76.07    84.94    88.29    92.72    95.61    98.72    99.41    99.83    100.00    100.00    100.00
## Points    83.76    91.06    93.93    95.43    95.94    96.44    96.55    96.73    96.84     96.84     97.69     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 12 components as in the linear model

# Slots of information in the model
names(modPlsr)
##  [1] "coefficients"    "scores"          "loadings"        "loading.weights" "Yscores"         "Yloadings"
##  [7] "projection"      "Xmeans"          "Ymeans"          "fitted.values"   "residuals"       "Xvar"
## [13] "Xtotvar"         "fit.time"        "ncomp"           "method"          "center"          "scale"
## [19] "call"            "terms"           "model"

# PLS scores
head(modPlsr$scores)
##                       Comp 1     Comp 2     Comp 3     Comp 4     Comp 5      Comp 6     Comp 7     Comp 8      Comp 9
## Barcelona        7.36407868 -0.6592285 -0.4262822 -0.1357589  0.8668796 -0.52701770 -0.1547704 -0.3104798 -0.15841110
## Real Madrid      6.70659816 -1.2848902  0.7191143  0.5679398 -0.5998745  0.30663290  0.5382173  0.2110337  0.15242836
## Atlético Madrid  2.45577740  2.6469837  0.4573647  0.6179274 -0.4307465  0.13368949  0.2951566 -0.1597672 -0.15232469
## Villarreal       0.06913485  2.0275895  0.6548014 -0.2358589  0.8350426  0.61305618 -0.6684330  0.0556628  0.07919102
## Athletic         0.96513341  0.3860060 -0.2718465 -0.1580349 -0.2649106  0.01822319 -0.1899969  0.3846156 -0.28926416
## Celta           -0.39588401 -0.5296417  0.6148093  0.1494496  0.5817902  0.69106828  0.1586472  0.2396975  0.41575546
##                      Comp 10       Comp 11       Comp 12
## Barcelona       -0.18952142 -0.0001005279  0.0011993985
## Real Madrid      0.15826286 -0.0011752690 -0.0030013550
## Atlético Madrid  0.05857664  0.0028149603  0.0012838990
## Villarreal      -0.05343433  0.0012705829 -0.0001652195
## Athletic         0.03702926 -0.0008077855  0.0052995290
## Celta           -0.03488514  0.0003208900  0.0022585552

# Also uncorrelated
head(cov(modPlsr$scores))
##               Comp 1        Comp 2        Comp 3        Comp 4        Comp 5        Comp 6        Comp 7        Comp 8
## Comp 1  7.393810e+00  3.973430e-16 -2.337312e-16 -7.304099e-18  9.115515e-16  1.149665e-15  3.272236e-16  7.011935e-17
## Comp 2  3.973430e-16  1.267859e+00 -2.980072e-16 -1.840633e-16 -2.103580e-16 -1.920978e-16 -2.337312e-17 -9.495329e-17
## Comp 3 -2.337312e-16 -2.980072e-16  9.021126e-01 -8.399714e-18  1.256305e-16  1.003401e-16 -2.921640e-18  5.843279e-18
## Comp 4 -7.304099e-18 -1.840633e-16 -8.399714e-18  3.130984e-01 -5.660677e-17 -4.660928e-17 -3.652049e-19  2.282531e-17
## Comp 5  9.115515e-16 -2.103580e-16  1.256305e-16 -5.660677e-17  2.586132e-01  9.897054e-17  8.180591e-17  3.432926e-17
## Comp 6  1.149665e-15 -1.920978e-16  1.003401e-16 -4.660928e-17  9.897054e-17  2.792408e-01  1.132135e-17  1.314738e-17
##               Comp 9       Comp 10       Comp 11       Comp 12
## Comp 1 -4.090295e-17 -2.571043e-16 -1.349090e-16 -3.928806e-16
## Comp 2 -1.358562e-16 -3.652049e-17 -1.000234e-16  2.990686e-17
## Comp 3 -1.168656e-17 -2.848599e-17  8.478176e-18 -9.205732e-18
## Comp 4  2.246010e-17  2.757297e-17 -2.658007e-17 -7.249889e-18
## Comp 5  4.382459e-18  1.679943e-17 -5.937433e-18  7.983152e-18
## Comp 6  1.150396e-17  2.191230e-18 -2.137412e-17 -5.567770e-18

# The coefficients of the original predictors, not of the components!
modPlsr$coefficients[, , 2]
##              Goals.scored            Goals.conceded   Percentage.scored.goals Percentage.conceded.goals
##                 1.8192870                -4.4038213                 1.8314760                -4.4045722
##                     Shots             Shots.on.goal          Penalties.scored               Assistances
##                 0.4010902                 0.9369002                -0.2006251                 2.3688050
##                Fouls.made              Yellow.cards                 Red.cards                  Offsides
##                 0.2807601                -1.6677725                -2.4952503                 1.2187529

# Obtaining the coefficients of the PLS components
```

```
lm(formula = Points ~., data = data.frame("Points" = laliga$Points,
                                          modPlsr$scores[, 1:3]))
##
## Call:
## lm(formula = Points ~ ., data = data.frame(Points = laliga$Points,
##     modPlsr$scores[, 1:3]))
##
## Coefficients:
## (Intercept)       Comp.1       Comp.2       Comp.3
##      52.400        6.093        4.341        3.232

# Prediction
predict(modPlsr, newdata = laligaRed2[1:2, ], ncomp = 12)
## , , 12 comps
##
##               Points
## Barcelona   92.01244
## Real Madrid 91.38026

# Selecting the number of components to retain
modPlsr2 <- plsr(Points ~ ., data = laligaRed2, scale = TRUE, ncomp = 2)
summary(modPlsr2)
## Data:    X dimension: 20 12
##  Y dimension: 20 1
## Fit method: kernelpls
## Number of components considered: 2
## TRAINING: % variance explained
##         1 comps  2 comps
## X         62.53    76.07
## Points    83.76    91.06

# Selecting the number of components to retain by Leave-One-Out cross-validation
modPlsrCV1 <- plsr(Points ~ ., data = laligaRed2, scale = TRUE,
                   validation = "LOO")
summary(modPlsrCV1)
## Data:    X dimension: 20 12
##  Y dimension: 20 1
## Fit method: kernelpls
## 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  7 comps  8 comps  9 comps  10 comps  11 comps
## CV           18.57    8.307    7.221    6.807    6.254    6.604    6.572    6.854    7.348    7.548     7.532     7.854
## adjCV        18.57    8.282    7.179    6.742    6.193    6.541    6.490    6.764    7.244    7.430     7.416     7.717
##        12 comps
## CV        7.905
## adjCV     7.760
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X         62.53    76.07    84.94    88.29    92.72    95.61    98.72    99.41    99.83    100.00    100.00    100.00
## Points    83.76    91.06    93.93    95.43    95.94    96.44    96.55    96.73    96.84     96.84     97.69     97.95

# View cross-validation Mean Squared Error Prediction
validationplot(modPlsrCV1, val.type = "MSEP") # l = 4 gives the minimum CV
```



```
# Selecting the number of components to retain by 10-fold Cross-Validation
# (k = 10 is the default)
modPlsrCV10 <- plsr(Points ~ ., data = laligaRed2, scale = TRUE,
                    validation = "CV")
summary(modPlsrCV10)
## Data:    X dimension: 20 12
##  Y dimension: 20 1
## Fit method: kernelpls
## 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  7 comps  8 comps  9 comps  10 comps  11 comps
## CV           18.57    7.895    6.944    6.571    6.330    6.396    6.607    7.018    7.487    7.612     7.607     8.520
## adjCV        18.57    7.852    6.868    6.452    6.201    6.285    6.444    6.830    7.270    7.372     7.368     8.212
##        12 comps
## CV        8.014
## adjCV     7.714
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X         62.53    76.07    84.94    88.29    92.72    95.61    98.72    99.41    99.83    100.00    100.00    100.00
## Points    83.76    91.06    93.93    95.43    95.94    96.44    96.55    96.73    96.84     96.84     97.69     97.95
validationplot(modPlsrCV10, 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$Points, cbind(modPlsr$scores))

# Regression on the first two PLS
modPlsr <- lm(Points ~ Comp.1 + Comp.2, data = laligaPLS)
summary(modPlsr) # Predictors clearly significant -- same R^2 as in modPlsr2
##
## Call:
## lm(formula = Points ~ Comp.1 + Comp.2, data = laligaPLS)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.3565 -3.6157  0.4508  2.3288 12.3116
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  52.4000     1.2799  40.941  < 2e-16 ***
## Comp.1        6.0933     0.4829  12.618 4.65e-10 ***
## Comp.2        4.3413     1.1662   3.723  0.00169 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.724 on 17 degrees of freedom
## Multiple R-squared:  0.9106, Adjusted R-squared:    0.9
## F-statistic: 86.53 on 2 and 17 DF,  p-value: 1.225e-09
car::vif(modPlsr) # No problems at all
## Comp.1 Comp.2
##      1      1
```

**Exercise 3.15.** Let's perform PCR and PLSR 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/PLSR (it is transformed internally to two dummy variables). Do the following:

a. Compute the PCA of iris, excluding Species. What is the percentage of variability explained with two components?
b. Draw the biplot and look for interpretations of the principal components.
c. Plot the PCA scores in a car::scatterplotMatrix plot such that the scores are colored by the levels in Species (you can use the groups argument).
d. Compute the PCR of Petal.Width with $\ell = 2$ (exclude Species). Inspect by leave-one-out cross-validation the best selection of $\ell$.
e. Repeat the previous point with PLSR.

f. Plot the PLS scores of the data in a `car::scatterplotMatrix` plot such that the scores are colored by the levels in `Species`. Compare with the PCA scores. Is there any interesting interpretation? (A quick way of converting the `scores` to a matrix is with `rbind`.)

In theory, the dimensionality reduction of PLSR is more adequate for linear regression than that of PCR. However, **in many practical situations PLSR and PCR tend to perform similarly**, since the variance of the PLSR predictors $P_1, \ldots, P_\ell$ is larger (due to the iterative linear-filtering procedure on which they are constructed) than the variance of the PCR predictors $\Gamma_1, \ldots, \Gamma_\ell$.

**Inference for PLSR is more involved** since, differently to what happens in PCR, the **PLS directions are dependent on the response** $Y$. This directly breaks a core assumption made in Section 2.4: that the randomness of the regression model came only from the error terms and not from the predictors.

We conclude by illustrating the differences between PLSR and PCR with a small numerical example. For that, we consider two correlated predictors

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim \mathcal{N}_2\left( \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right), \quad -1 \leq \rho \leq 1, \tag{3.21}$$

and the linear model

$$Y = \beta_1 X_1 + \beta_2 X_2 + \varepsilon, \quad \boldsymbol{\beta} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}, \tag{3.22}$$

where $\theta \in [0, 2\pi)$ and $\varepsilon \sim \mathcal{N}(0,1)$. Therefore:

- The **correlation** $\rho$ controls the **linear dependence between** $X_1$ **and** $X_2$. This parameter is the only one that affects the two principal components $\Gamma_1 = a_{11}X_1 + a_{21}X_2$ and $\Gamma_2 = a_{12}X_1 + a_{22}X_2$ (recall (3.9)). The loadings $(\hat{a}_{11}, \hat{a}_{21})^\top$ and $(\hat{a}_{12}, \hat{a}_{22})^\top$ are the (sample) **PC1 and PC2 directions**.
- The **angle** $\theta$ controls the direction of $\boldsymbol{\beta}$, that is, the **linear relation between** $Y$ **and** $(X_1, X_2)$. Consequently, $\theta$ affects the PLS components $P_1 = b_{11}X_1 + b_{21}X_2$ and $P_2 = b_{12}X_1 + b_{22}X_2$ (recall (3.20), with adapted notation). The loadings $(\hat{b}_{11}, \hat{b}_{21})^\top$ and $(\hat{b}_{12}, \hat{b}_{22})^\top$ are the (sample) **PLS1 and PLS2 directions**.

The animation in Figure 3.6.3 contains samples from (3.21) colored according to their expected value under (3.22). It illustrates the following insights:

- PC directions are unaware of $\boldsymbol{\beta}$; PLS directions are affected by $\boldsymbol{\beta}$.

[52] Recall that we do not care about whether the directions of PLS1 and $\boldsymbol{\beta}$ are close, but rather if the *axes* spanned by PLS1 and $\boldsymbol{\beta}$ are close. The signs of $\boldsymbol{\beta}$ are learned during the model fitting!

- PC directions are *always* orthogonal; PLS directions are *not*.
- PLS1 aligns[52] with $\boldsymbol{\beta}$ when $\rho = 0$. This does not happen when $\rho \neq 0$ (remember Figure 2.5).
- Under high correlation, the PLS and PC directions are very similar.
- Under low-moderate correlation, PLS1 better explains[53] $\boldsymbol{\beta}$ than PC1.

Illustration of the differences and similarities between PCR and PLSR. The points are sampled from (3.21) and colored according to their expected value under (3.22): yellow for larger values; dark violet for smaller. The color gradient is thus controlled by the direction of $\boldsymbol{\beta}$ (black arrow) and informs on the position of the plane $y = \beta_1 x_1 + \beta_2 x_2$. The PC/PLS directions are described in the text. Application available here.

[53] In the sense that the absolute value of the projection of $\boldsymbol{\beta}$ along $(\hat{b}_{11}, \hat{b}_{21})^\top$, $|(\hat{b}_{11}, \hat{b}_{21})\boldsymbol{\beta}|$, is usually larger than that along $(\hat{a}_{11}, \hat{a}_{21})^\top$.

# 4
# Linear models III: shrinkage, multivariate response, and big data

We explore in this chapter several extensions of the linear model for certain non-classical settings such as: high-dimensional data ($p \gg n$) that requires shrinkage methods, *big data* (large $n$) that demands thoughtful computations, and the multivariate response situation in which the interest lies in explaining a *vector* of responses $\mathbf{Y} = (Y_1, \ldots, Y_q)$.

## 4.1 Shrinkage

As we saw in Section 2.4.1, the least squares estimates $\hat{\boldsymbol{\beta}}$ of the linear model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

were the minimizers of the residual sum of squares

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \cdots - \beta_p X_{ip})^2.$$

Under the validity of the assumptions of Section 2.3, in Section 2.4 we saw that

$$\hat{\boldsymbol{\beta}} \sim \mathcal{N}_{p+1}\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}\right).$$

A particular consequence of this result is that $\hat{\boldsymbol{\beta}}$ is *unbiased* in estimating $\boldsymbol{\beta}$, that is, $\hat{\boldsymbol{\beta}}$ does not make any systematic error in estimating $\boldsymbol{\beta}$. However, bias is only one part of the quality of an estimate: variance is also important. Indeed, the *bias-variance trade-off*[1] 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

$$\mathrm{MSE}[\hat{\beta}_j] := \mathbb{E}[(\hat{\beta}_j - \beta_j)^2] = \underbrace{(\mathbb{E}[\hat{\beta}_j] - \beta_j)^2}_{\text{Bias}^2} + \underbrace{\mathbb{V}\mathrm{ar}[\hat{\beta}_j]}_{\text{Variance}}. \tag{4.1}$$

*Shrinkage* methods pursue the following idea:

> Add an amount of *smart bias* to $\hat{\boldsymbol{\beta}}$ in order to reduce its variance, in such a way that we obtain *simpler interpretations* from the biased version of $\hat{\boldsymbol{\beta}}$.

[1] See Section 1.1.

This idea is implemented by enforcing *sparsity*, that is, by biasing the estimates of $\boldsymbol{\beta}$ towards being non-null only for the most important predictors. The two main methods covered in this section, **ridge regression** and **lasso** (*least absolute shrinkage and selection operator*), use this idea in a different way. However, it is important to realize that both methods do consider the standard linear model; what they do differently is *the way of estimating* $\boldsymbol{\beta}$.

The way they enforce sparsity in the estimates is by minimizing the RSS plus a *penalty* term that favors sparsity on the estimated coefficients. For example, the **ridge regression** enforces a quadratic penalty to the candidate *slope* coefficients $\boldsymbol{\beta}_{-1} = (\beta_1, \ldots, \beta_p)^\top$ and seeks to minimize[2]

$$\mathrm{RSS}(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS}(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}_{-1}\|_2^2, \tag{4.2}$$

where $\lambda \geq 0$ is the penalty parameter. On the other hand, the **lasso** considers an absolute penalty:

$$\mathrm{RSS}(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j| = \mathrm{RSS}(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}_{-1}\|_1. \tag{4.3}$$

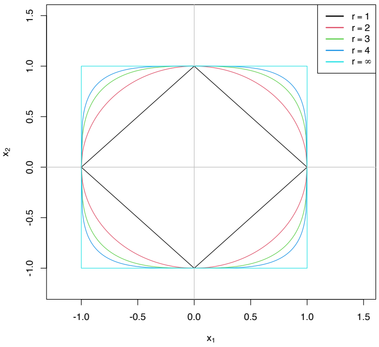


Figure 4.1: The "unit circle" $\|(x_1, x_2)\|_r = 1$ for $r = 1, 2, 3, 4, \infty$.

Among other possible joint representations for (4.2) and (4.3),[3] the one based on the *elastic nets* is particularly convenient, as it aims to combine the strengths of both methods in a computationally tractable way and is the one employed in the reference package `glmnet`. Considering a proportion $0 \leq \alpha \leq 1$, the elastic net is defined as

$$\mathrm{RSS}(\boldsymbol{\beta}) + \lambda(\alpha\|\boldsymbol{\beta}_{-1}\|_1 + (1-\alpha)\|\boldsymbol{\beta}_{-1}\|_2^2). \tag{4.4}$$

Clearly, **ridge** regression corresponds to $\alpha = 0$ (quadratic penalty) and **lasso** to $\alpha = 1$ (absolute penalty). Obviously, if $\lambda = 0$, we are back to the least squares problem and theory. The optimization of (4.4) gives

$$\hat{\boldsymbol{\beta}}_{\lambda,\alpha} := \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \left\{ \mathrm{RSS}(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} (\alpha|\beta_j| + (1-\alpha)|\beta_j|^2) \right\}, \tag{4.5}$$

which is the penalized estimation of $\boldsymbol{\beta}$. Note that the sparsity is enforced in the slopes, not in the intercept, since this depends on the mean of $Y$. Note also that the optimization problem is *convex*[4] and therefore it is guaranteed the existence and uniqueness of a minimum. However, in general,[5] there are no explicit formulas for $\hat{\boldsymbol{\beta}}_{\lambda,\alpha}$ and the optimization problem needs to be solved numerically. Finally, $\lambda$ is a tuning parameter that will need to be chosen suitably and that we will discuss later.[6] What it is important now is to recall that the *predictors need to be standardized*, or otherwise its scale will distort the optimization of (4.4).

An equivalent way of viewing (4.5) that helps in visualizing the differences between the ridge and lasso regressions is that they try to solve the equivalent optimization problem[7] of (4.5):

[2] Remember that, for a vector $\mathbf{x} \in \mathbb{R}^m$, $r \geq 1$, the $\ell^r$ norm (usually referred to as $\ell^p$ norm, but renamed here to avoid confusion with the number of predictors) is defined as $\|\mathbf{x}\|_r := \left(\sum_{j=1}^m |x_j|^r\right)^{1/r}$. The $\ell^\infty$ norm is defined as $\|\mathbf{x}\|_\infty := \max_{1 \leq j \leq m} |x_j|$ and it is satisfied that $\lim_{r\to\infty} \|\mathbf{x}\|_r = \|\mathbf{x}\|_\infty$, for all $\mathbf{x} \in \mathbb{R}^m$. A visualization of these norms for $m = 2$ is given in Figure 4.1.

[3] Such as, for example, $\mathrm{RSS}(\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}_{-1}\|_r^r$, for $r \geq 1$.

[4] The function $\|\mathbf{x}\|_r^r$ is convex if $r \geq 1$. If $0 < r < 1$, $\|\mathbf{x}\|_r^r$ is not convex.

[5] The main exception being the ridge regression, as seen in Section 4.1.1.

[6] In addition, $\alpha$ can also be regarded as a tuning parameter. We do not deal with its data-driven selection in these notes, as this will imply a more costly optimization of the cross-validation on the pair $(\lambda, \alpha)$.

[7] Recall that (4.5) can be seen as the Lagrangian of (4.6). Because of the convexity of the optimization problem, the minimizers of (4.5) and (4.6) do actually coincide.

$$\hat{\boldsymbol{\beta}}_{s_\lambda,\alpha} := \arg \min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}:\sum_{j=1}^{p}(\alpha|\beta_j|+(1-\alpha)|\beta_j|^2)\leq s_\lambda} \mathrm{RSS}(\boldsymbol{\beta}), \tag{4.6}$$

where $s_\lambda$ is certain scalar that does not depend on $\boldsymbol{\beta}$.

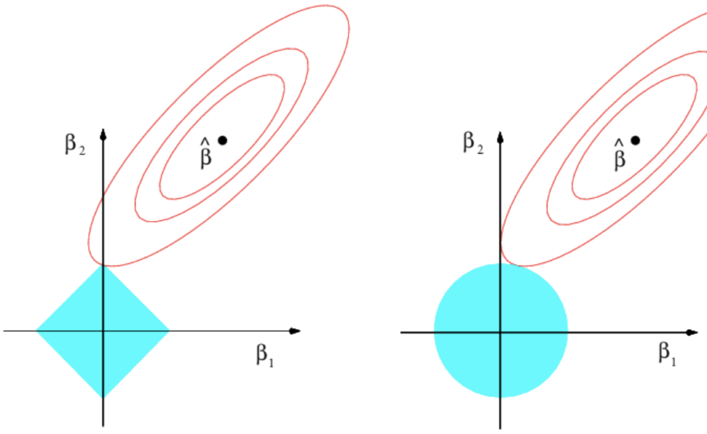


Figure 4.2: Comparison of ridge and lasso solutions from the optimization problem (4.6) with $p = 2$. The elliptical contours show the regions with equal $\mathrm{RSS}(\beta_1, \beta_2)$, the objective function, for $(\beta_1, \beta_2) \in \mathbb{R}^2$ ($\beta_0 = 0$ is assumed). The diamond ($\alpha = 1$) and circular ($\alpha = 0$) regions show the feasibility regions determined by $\sum_{j=1}^{p}(\alpha|\beta_j| + (1-\alpha)|\beta_j|^2) \leq s_\lambda$ for the optimization problem. The *sharpness* of the diamond makes the lasso attain solutions with many coefficients *exactly* equal to zero, in a similar situation to the one depicted. Extracted from James et al. (2013).

The `glmnet` package is the reference implementation of shrinkage estimators based on elastic nets. In order to illustrate how to apply the ridge and lasso regression in practice, we will work with the `ISLR::Hitters` dataset. This dataset 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
data(Hitters, package = "ISLR")

# 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 ~ 0 + ., data = Hitters)
# 0 + to exclude a column of 1's for the intercept, since the intercept will be
# added by default in glmnet::glmnet and if we do not exclude it here we will
# end with two intercepts, one of them resulting in NA. In the newer versions of
# glmnet this step is luckily not necessary

# Interestingly, note that in Hitters there are two-level factors and these
# are automatically transformed into dummy variables in x -- the main advantage
# of model.matrix
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])
##                   LeagueA LeagueN DivisionW PutOuts Assists Errors
## -Alan Ashby             0       1         1     632      43     10
## -Alvin Davis            1       0         1     880      82     14
## -Andre Dawson           0       1         0     200      11      3
## -Andres Galarraga       0       1         0     805      40      4
## -Alfredo Griffin        1       0         1     282     421     25
## -Al Newman              0       1         0      76     127      7
```

```r
# We also need the vector of responses
y <- Hitters$Salary
```

`model.matrix` removes by default the observations with any `NA`s, returning only the complete cases. This may be undesirable in certain circumstances. If `NA`s are to be preserved, an option is to use `na.action = "na.pass"` but with the function `model.matrix.lm` (not `model.matrix`, as it ignores the argument!).

The next code illustrates the previous warning.

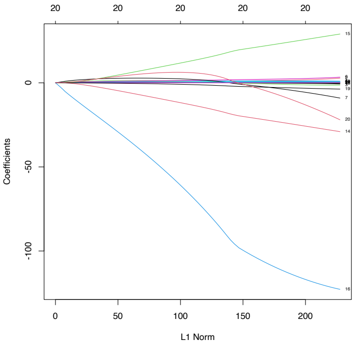
```r
# Data with NA in the first observation
data_na <- data.frame("x1" = rnorm(3), "x2" = rnorm(3), "y" = rnorm(3))
data_na$x1[1] <- NA

# The first observation disappears!
model.matrix(y ~ 0 + ., data = data_na)
##           x1       x2
## 2  0.5136652 1.435410
## 3 -0.6558154 1.085212
## attr(,"assign")
## [1] 1 2

# Still ignores NA's
model.matrix(y ~ 0 + ., data = data_na, na.action = "na.pass")
##           x1       x2
## 2  0.5136652 1.435410
## 3 -0.6558154 1.085212
## attr(,"assign")
## [1] 1 2

# Does not ignore NA's
model.matrix.lm(y ~ 0 + ., data = data_na, na.action = "na.pass")
##           x1        x2
## 1         NA -1.329132
## 2  0.5136652  1.435410
## 3 -0.6558154  1.085212
## attr(,"assign")
## [1] 1 2
```

## 4.1.1 *Ridge regression*

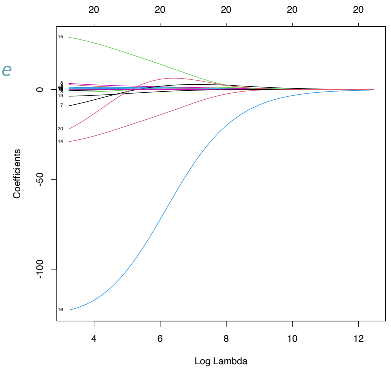
We describe next how to do the fitting, tuning parameter selection, prediction, and the computation of the analytical form for the ridge regression. The first three topics are very similar for the lasso or for other elastic net fits (i.e., without $\alpha = 0$).

### Fitting

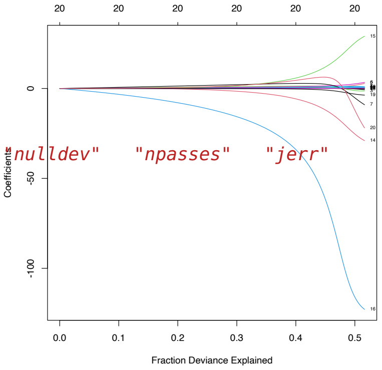
```r
# Call to the main function -- use alpha = 0 for ridge regression
library(glmnet)
ridgeMod <- glmnet(x = x, y = y, alpha = 0)
# By default, it computes the ridge solution over a set of lambdas
# automatically chosen. It also standardizes the variables by default to make
# the model fitting since 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)
```



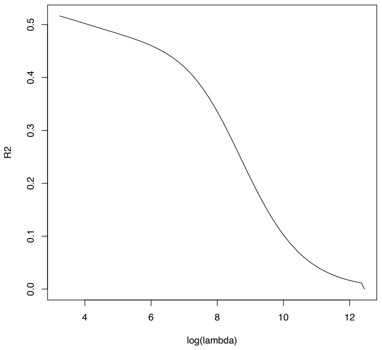
```
# xvar = "norm" is the default: L1 norm of the coefficients sum_j abs(beta_j)

# Versus lambda
plot(ridgeMod, label = TRUE, xvar = "lambda")
```



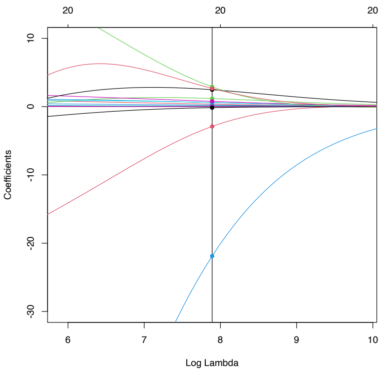
```
# Versus the percentage of deviance explained -- this is a generalization of the
# R^2 for generalized linear models. Since we have a linear model, this is the
# same as the R^2
plot(ridgeMod, label = TRUE, xvar = "dev")
```



```
# 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 16, 14, and 15
colnames(x)[c(16, 14, 15)]
## [1] "DivisionW" "LeagueA"   "LeagueN"

# What is inside glmnet's output?
names(ridgeMod)
##  [1] "a0"        "beta"      "df"        "dim"       "lambda"    "dev.ratio" "nulldev"   "npasses"   "jerr"
## [10] "offset"    "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] 0.5164752
# Slightly different to lm's because it compromises 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$beta)
## [1]  20 100
length(ridgeMod$lambda) # 100 lambda's were automatically chosen
## [1] 100

# Inspecting the coefficients associated with the 50th lambda
coef(ridgeMod)[, 50]
##   (Intercept)         AtBat          Hits         HmRun          Runs           RBI         Walks         Years
## 214.720400777   0.090210299   0.371622563   1.183305701   0.597288606   0.595291960   0.772639340   2.474046387
##        CAtBat         CHits        CHmRun         CRuns          CRBI        CWalks       LeagueA       LeagueN
##   0.007597343   0.029269640   0.217470479   0.058715486   0.060728347   0.058696981  -2.903863555   2.903558868
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -21.886726912   0.052629591   0.007406370  -0.147635449   2.663300534
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$lambda[50]), nrow(ridgeMod$beta)), ridgeMod$beta[, 50],
       pch = 16, col = 1:6)
```

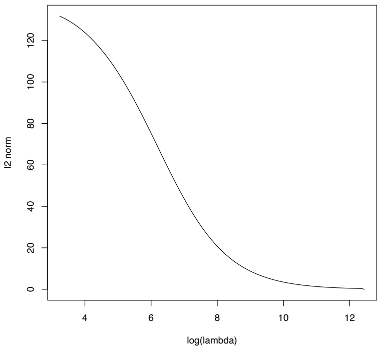


```
# The squared l2-norm of the coefficients decreases as lambda increases
plot(log(ridgeMod$lambda), sqrt(colSums(ridgeMod$beta^2)), type = "l",
     xlab = "log(lambda)", ylab = "l2 norm")
```

## Tuning parameter selection

The selection of the penalty parameter $\lambda$ is usually done by $k$-fold cross-validation, following the general principle described at the end of Section 3.6. This data-driven selector is denoted by $\hat{\lambda}_{k\text{-CV}}$ and has the form given[8] in (3.19) (or (3.18) if $k = n$):

$$\hat{\lambda}_{k\text{-CV}} := \arg\min_{\lambda \geq 0} \mathrm{CV}_k(\lambda), \quad \mathrm{CV}_k(\lambda) := \sum_{j=1}^{k} \sum_{i \in F_j} (Y_i - \hat{m}_{\lambda,-F_j}(\mathbf{X}_i))^2.$$



A very interesting variant for the $\hat{\lambda}_{k\text{-CV}}$ selector is the so-called ***one standard error rule***. This rule is based on a parsimonious principle:

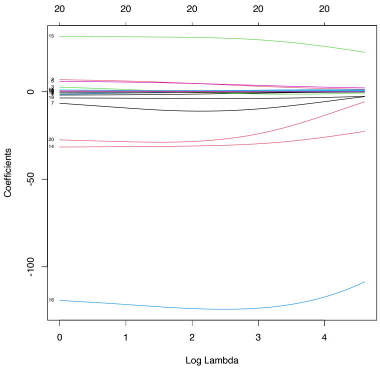
> "favor simplicity within the set of most likely optimal models".

It arises from observing that the objective function to minimize, $\mathrm{CV}_k$, is random. Thus, its minimizer $\hat{\lambda}_{k\text{-CV}}$ is subjected to variability. Then, the parsimonious approach proceeds by selecting not $\hat{\lambda}_{k\text{-CV}}$, but the *largest* $\lambda$ (hence, the *simplest* model) that is still *likely* optimal, i.e., that is "close" to $\hat{\lambda}_{k\text{-CV}}$. This closeness is quantified by the estimation of the standard deviation of the *random variable* $\mathrm{CV}_k(\hat{\lambda}_{k\text{-CV}})$, which is obtained thanks to the folding splitting of the sample. Mathematically, $\hat{\lambda}_{k\text{-1SE}}$ is defined as

$$\hat{\lambda}_{k\text{-1SE}} := \max\left\{\lambda \geq 0 : \mathrm{CV}_k(\lambda) \in \left(\mathrm{CV}_k(\hat{\lambda}_{k\text{-CV}}) \pm \hat{\mathrm{SE}}\left(\mathrm{CV}_k(\hat{\lambda}_{k\text{-CV}})\right)\right)\right\}.$$

The $\hat{\lambda}_{k\text{-1SE}}$ selector often offers a good trade-off between model fitness and interpretability in practice.[9] The code below gives all the details.

```
# 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!
```

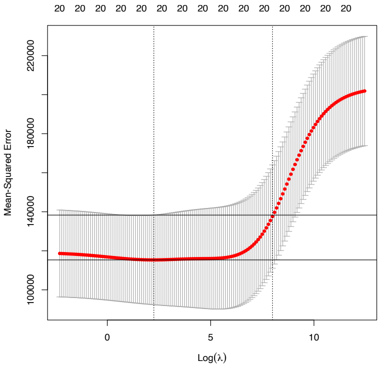


```
# Lambda is a tuning parameter that can be chosen by cross-validation, using as
# error the MSE (other possible error can be considered for generalized models
# using the argument type.measure)

# 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 that minimizes the CV error is
kcvRidge$lambda.min
## [1] 25.52821

# Equivalent to
indMin <- which.min(kcvRidge$cvm)
kcvRidge$lambda[indMin]
## [1] 25.52821

# The minimum CV error
kcvRidge$cvm[indMin]
## [1] 115034
min(kcvRidge$cvm)
## [1] 115034

# Potential problem! Minimum occurs at one extreme of the lambda grid in which
```

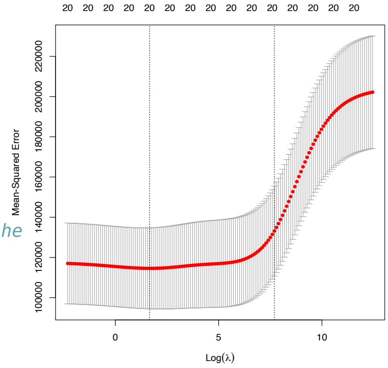
[8] If the linear model is employed, for generalized linear models the metric in the cross-validation function is not the squared difference between observations and fitted values.

[9] Indeed, Section 4.1.3 shows that $\hat{\lambda}_{k\text{-1SE}}$ seems to be surprisingly good in terms of consistently identifying the underlying model.

```
# CV is done. The grid was automatically selected, but can be manually inputted
range(kcvRidge$lambda)
## [1]     25.52821 255282.09651
lambdaGrid <- 10^seq(log10(kcvRidge$lambda[1]), log10(0.1),
                     length.out = 150) # log-spaced grid
kcvRidge2 <- cv.glmnet(x = x, y = y, nfolds = 10, alpha = 0,
                       lambda = lambdaGrid)

# Much better
plot(kcvRidge2)
kcvRidge2$lambda.min
## [1] 9.506186

# 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 favor simplicity of the model
# around the optimal lambda value. This is known as the "one standard error rule"
kcvRidge2$lambda.1se
## [1] 2964.928

# 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)
indMin2 <- which.min(kcvRidge2$cvm)
abline(h = kcvRidge2$cvm[indMin2] + c(0, kcvRidge2$cvsd[indMin2]))
```

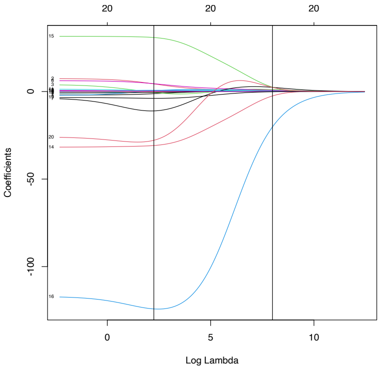


```
# The consideration of the one standard error rule for selecting lambda makes
# special sense when the CV function is quite flat around the minimum (hence an
# overpenalization that gives more sparsity does not affect so much the CV loss)

# 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)

# Location of both optimal lambdas in the CV loss function
plot(ncvRidge)
```

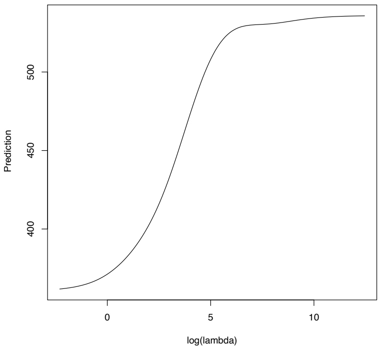


## Prediction

```
# Inspect the best models (the glmnet fit is inside the output of cv.glmnet)
plot(kcvRidge2$glmnet.fit, label = TRUE, xvar = "lambda")
abline(v = log(c(kcvRidge2$lambda.min, kcvRidge2$lambda.1se)))
```



```
# The model associated with `lambda.1se` (or any other lambda not included in the
# original path solution -- obtained by an interpolation) can be retrieved with
predict(kcvRidge2, type = "coefficients", s = kcvRidge2$lambda.1se)
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) 229.758314334
## AtBat         0.086325740
## Hits          0.351303930
## HmRun         1.142772275
## Runs          0.567245068
## RBI           0.568056880
## Walks         0.731144713
## Years         2.389248929
## CAtBat        0.007261489
## CHits         0.027854683
## CHmRun        0.207220032
## CRuns         0.055877337
## CRBI          0.057777505
## CWalks        0.056352113
## LeagueA      -2.509251990
## LeagueN       2.509060248
## DivisionW   -20.162700810
## PutOuts       0.048911039
## Assists       0.006973696
## Errors       -0.128351187
## NewLeagueN    2.373103450

# Alternatively, we can use coef()
coef(kcvRidge2, s = kcvRidge2$lambda.1se)
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept) 229.758314334
## AtBat         0.086325740
## Hits          0.351303930
## HmRun         1.142772275
## Runs          0.567245068
## RBI           0.568056880
## Walks         0.731144713
## Years         2.389248929
## CAtBat        0.007261489
## CHits         0.027854683
## CHmRun        0.207220032
## CRuns         0.055877337
## CRBI          0.057777505
## CWalks        0.056352113
## LeagueA      -2.509251990
## LeagueN       2.509060248
## DivisionW   -20.162700810
## PutOuts       0.048911039
## Assists       0.006973696
## Errors       -0.128351187
## NewLeagueN    2.373103450

# Predictions for the first two observations
predict(kcvRidge2, type = "response", s = kcvRidge2$lambda.1se,
        newx = x[1:2, ])
##                     s1
## -Alan Ashby  530.8080
## -Alvin Davis 577.8485

# Predictions for the first observation, for all the lambdas. We can see how
# the prediction for one observation changes according to lambda
plot(log(kcvRidge2$lambda),
     predict(kcvRidge2, type = "response", newx = x[1, , drop = FALSE],
             s = kcvRidge2$lambda),
     type = "l", xlab = "log(lambda)", ylab = " Prediction")
```



### Analytical form

The optimization problem (4.5) has an explicit solution for $\alpha = 0$. To see it, assume that both the response $Y$ and the predictors $X_1, \ldots, X_p$ are *centered*, and that the sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$ is also centered.[10] In this case, there is no intercept $\beta_0$ $(= 0)$ to estimate by $\hat\beta_0$ $(= 0)$ and the linear model is simply

$$Y = \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon.$$

Then, the ridge regression estimator $\hat{\boldsymbol\beta}_{\lambda,0} \in \mathbb{R}^p$ is

$$\begin{aligned}\hat{\boldsymbol\beta}_{\lambda,0} &= \arg\min_{\boldsymbol\beta\in\mathbb{R}^p} \mathrm{RSS}(\boldsymbol\beta) + \lambda\|\boldsymbol\beta\|_2^2\\ &= \arg\min_{\boldsymbol\beta\in\mathbb{R}^p} \sum_{i=1}^n (Y_i - \mathbf{X}_i\boldsymbol\beta)^2 + \lambda\boldsymbol\beta^\top\boldsymbol\beta\\ &= \arg\min_{\boldsymbol\beta\in\mathbb{R}^p} (\mathbf{Y} - \mathbf{X}\boldsymbol\beta)^\top(\mathbf{Y} - \mathbf{X}\boldsymbol\beta) + \lambda\boldsymbol\beta^\top\boldsymbol\beta, \qquad (4.7)\end{aligned}$$

[10] That is, that $\bar Y = 0$ and $\bar X_j = 0$, $j = 1, \ldots, p$.

where $\mathbf{X}$ is the design matrix but now excluding the column of ones (thus of size $n \times p$). Nicely, (4.7) is a continuous quadratic optimization problem that is easily solved with the same arguments we employed for obtaining (2.7), resulting in[11]

$$\hat{\boldsymbol{\beta}}_{\lambda,0} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top\mathbf{Y}. \tag{4.8}$$

The form (4.8) neatly connects with the least squares estimator ($\lambda = 0$) and yields many interesting insights. First, notice how the ridge regression estimator is *always computable*, even if $p \gg n$ [12] and the matrix $\mathbf{X}^\top\mathbf{X}$ is not invertible, or if $\mathbf{X}^\top\mathbf{X}$ is singular due to perfect multicollinearity. Second, as it was done with (2.11), it is straightforward to see that, under the assumptions of the linear model,

$$\hat{\boldsymbol{\beta}}_{\lambda,0} \sim \mathcal{N}_p\left((\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta}, \sigma^2(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\right). \tag{4.9}$$

The distribution (4.9) is revealing: it shows that $\hat{\boldsymbol{\beta}}_{\lambda,0}$ is no longer unbiased and that its variance is smaller[13] than the least squares estimator $\hat{\boldsymbol{\beta}}$. This is much more clear in the case where the predictors are also *uncorrelated* and *standardized* (in addition to being centered), hence the sample covariance matrix is $\mathbf{S} = \mathbf{I}_p$ and, equivalently, $\mathbf{X}^\top\mathbf{X} = n\mathbf{I}_p$. This is precisely the case of the PCA or PLS scores if these are standardized to have unit variance. In this situation, then (4.8) and (4.9) simplify to[14]

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{\lambda,0} &= \frac{1}{n+\lambda}\mathbf{X}^\top\mathbf{Y} = \frac{n}{n+\lambda}\hat{\boldsymbol{\beta}},\\ \hat{\boldsymbol{\beta}}_{\lambda,0} &\sim \mathcal{N}_p\left(\frac{n}{n+\lambda}\boldsymbol{\beta}, \sigma^2\frac{n}{(n+\lambda)^2}\mathbf{I}_p\right).\end{aligned} \tag{4.10}$$

The shrinking effect of $\lambda$ is yet more evident from (4.10): when the predictors are uncorrelated, we shrink *equally* the least squares estimator $\hat{\boldsymbol{\beta}}$ by the factor $n/(n+\lambda)$, which results in a reduction of the variance by a factor of $n/(n+\lambda)^2$. Furthermore, notice an important point: due to the explicit control of the distribution of $\hat{\boldsymbol{\beta}}_{\lambda,0}$, **inference** about $\boldsymbol{\beta}$ can be done in a relatively straightforward way in this special case[15] from $\hat{\boldsymbol{\beta}}_{\lambda,0}$, just as it was done from $\hat{\boldsymbol{\beta}}$ in Section 2.4. This tractability, both on the explicit form of the estimator and on the associated inference, is one of the main advantages of ridge regression with respect to other shrinkage methods.

Finally, just as we did for the least squares estimator, we can define the *hat matrix*

$$\mathbf{H}_\lambda := \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top$$

that predicts $\hat{\mathbf{Y}}$ from $\mathbf{Y}$. This hat matrix becomes especially useful now, as it can be employed to define the *effective degrees of freedom* associated with a ridge regression with penalty $\lambda$. These are defined as the trace of the hat matrix:

$$\mathrm{df}(\lambda) := \mathrm{tr}(\mathbf{H}_\lambda).$$

[11] If the data was not centered, then (4.8) would translate into $\hat{\boldsymbol{\beta}}_{\lambda,0} = (\mathbf{X}^\top\mathbf{X} + \mathrm{diag}(0, \lambda, \overset{p}{\ldots}, \lambda))^{-1}\mathbf{X}^\top\mathbf{Y}$, where $\mathbf{X}$ is the $n \times (p+1)$ design matrix with the first column consisting of ones.

[12] This was the original motivation for ridge regression, a way of estimating $\boldsymbol{\beta}$ beyond $p \leq n$. This property also holds for any other elastic net estimator (e.g., lasso) as long as $\lambda > 0$.

[13] If the eigenvalues of $\mathbf{X}^\top\mathbf{X}$ are $\eta_1 \geq \ldots \geq \eta_p > 0$, then the eigenvalues of $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p$ are $\eta_1 + \lambda \geq \ldots \geq \eta_p + \lambda > 0$ (because the addition involves a constant diagonal matrix). Therefore, the determinant of $(\mathbf{X}^\top\mathbf{X})^{-1}$ is $\prod_{j=1}^p \eta_j^{-1}$ and the determinant of $(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}$ is $\prod_{j=1}^p \eta_j(\eta_j + \lambda)^{-2}$, which is smaller than $\prod_{j=1}^p \eta_j^{-1}$ since $\lambda \geq 0$.

[14] Observe that the expectation of $\hat{\boldsymbol{\beta}}_{\lambda,0}$ depends on $n$! This was not the case for the least squares estimator $\hat{\boldsymbol{\beta}}$. This dependency can be removed if we reparametrize the penalty parameter $\lambda$ as $\tilde{\lambda} := \lambda/n$, so that $\lambda = n\tilde{\lambda}$. In the scale $\tilde{\lambda}$, the expectation of $\hat{\boldsymbol{\beta}}_{\tilde{\lambda},0}$ is fixed with respect to $n$. The scale $\tilde{\lambda}$ is indeed the one used by `glmnet`: $\lambda_{\texttt{glmnet}} = \tilde{\lambda}$.

[15] Recall that, if the predictors are centered, scaled, and uncorrelated, $\frac{n+\lambda}{n}\hat{\boldsymbol{\beta}}_{\lambda,0} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \frac{\sigma^2}{n}\mathbf{I}_p\right)$ and the $t$-tests and CIs for $\beta_j$ follow easily from here. In general, from (4.9) it follows that $\left(\mathbf{I}_p + \lambda(\mathbf{X}^\top\mathbf{X})^{-1}\right)\hat{\boldsymbol{\beta}}_{\lambda,0} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}\right)$ as long as $\mathbf{X}^\top\mathbf{X}$ is invertible, from where $t$-tests and CIs for $\beta_j$ can be derived.

[16] We employ the cyclic property of the trace operator: $\mathrm{tr}(\mathbf{ABC}) = \mathrm{tr}(\mathbf{CAB}) = \mathrm{tr}(\mathbf{BCA})$ for any matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ for which the multiplications are possible.

The motivation behind is that, for the unrestricted least squares fit[16]

$$\mathrm{tr}(\mathbf{H}_0) = \mathrm{tr}\big(\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\big) = \mathrm{tr}\big(\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\big) = p$$

and thus indeed $\mathrm{df}(0) = p$ is representing the degrees of freedom of the fit, understood as the number of parameters employed (keep in mind that the intercept was excluded). For a constrained fit with $\lambda > 0$, $\mathrm{df}(\lambda) < p$ because, even if we are estimating $p$ parameters in $\hat{\boldsymbol{\beta}}_{\lambda,0}$, these are restricted to satisfy $\|\hat{\boldsymbol{\beta}}_{\lambda,0}\|_2^2 \leq s_\lambda$ (for a certain $s_\lambda$, recall (4.6)). The function df is monotonically decreasing and such that $\lim_{\lambda\to\infty}\mathrm{df}(\lambda) = 0$, see Figure 4.3. Recall that, due to the imposed constraint on the coefficients, we could choose $\lambda$ such that $\mathrm{df}(\lambda) = r$, where $r$ is an integer smaller than $p$: this would correspond to effectively employing exactly $r$ parameters in the regression, despite considering $p$ predictors.

The next chunk of code implements $\hat{\boldsymbol{\beta}}_{\lambda,0}$ and shows that is equivalent to the output of `glmnet::glmnet`, with certain quirks.[17,18]

```r
# Random data
p <- 5
n <- 200
beta <- seq(-1, 1, l = p)
set.seed(123124)
x <- matrix(rnorm(n * p), n, p)
y <- 1 + x %*% beta + rnorm(n)

# Mimic internal standardization of y done in glmnet, which affects the scale
# of lambda in the regularization
y <- scale(y, center = TRUE, scale = TRUE) * sqrt(n / (n - 1))

# Unrestricted fit
fit <- glmnet(x, y, alpha = 0, lambda = 0, intercept = TRUE,
              standardize = FALSE)
beta0Hat <- rbind(fit$a0, fit$beta)
beta0Hat
## 6 x 1 sparse Matrix of class "dgCMatrix"
##              s0
##    -0.007815871
## V1 -0.521693181
## V2 -0.288052685
## V3 -0.019294336
## V4  0.239902118
## V5  0.535110595

# Unrestricted fit matches least squares -- but recall glmnet uses an
# iterative method so it is inexact (convergence threshold thresh = 1e-7 by
# default)
X <- model.matrix(y ~ x) # A way of constructing a design matrix that is a
# data.frame and has a column of ones
solve(crossprod(X)) %*% t(X) %*% y
##                    [,1]
## (Intercept) -0.007815868
## x1          -0.521693158
## x2          -0.288052685
## x3          -0.019294338
## x4           0.239902117
## x5           0.535110595

# Restricted fit
# glmnet considers as the regularization parameter "lambda" the value
# lambda / n (lambda being here the penalty parameter employed in the theory)
lambda <- 2
fit <- glmnet(x, y, alpha = 0, lambda = lambda / n, intercept = TRUE,
```

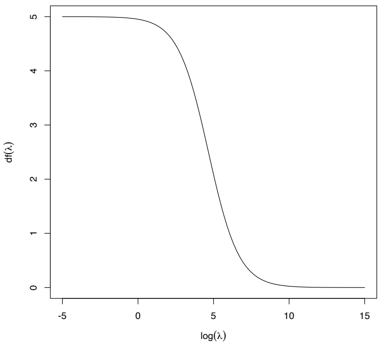


Figure 4.3: The effective degrees of freedom $\mathrm{df}(\lambda)$ as a function of $\log(\lambda)$ for a ridge regression with $p = 5$.

[17] One quirk is that `glmnet::glmnet` considers $\frac{1}{2n}\mathrm{RSS}(\boldsymbol{\beta}) + \lambda\left(\alpha\|\boldsymbol{\beta}_{-1}\|_1 + \frac{1-\alpha}{2}\|\boldsymbol{\beta}_{-1}\|_2^2\right)$ instead of (4.4), since $\frac{1}{2n}\mathrm{RSS}(\boldsymbol{\beta})$ is more related with the log-likelihood. This rescaling of $\mathrm{RSS}(\boldsymbol{\beta})$ affects the scale of $\lambda$.

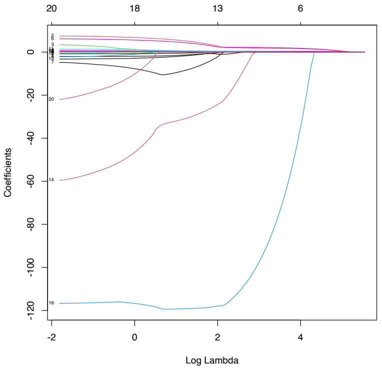
[18] Another quirk is that "`glmnet::glmnet` standardizes `y` to have unit variance (using $1/n$ rather than $1/(n-1)$ formula) before computing its lambda sequence (and then unstandardizes the resulting coefficients)".

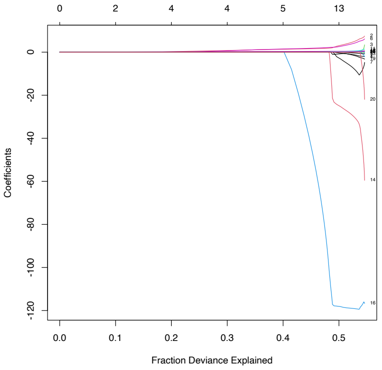
```
              standardize = FALSE, thresh = 1e-10)
betaLambdaHat <- rbind(fit$a0, fit$beta)
betaLambdaHat
## 6 x 1 sparse Matrix of class "dgCMatrix"
##               s0
##    -0.007775041
## V1 -0.517108905
## V2 -0.285489929
## V3 -0.018898839
## V4  0.236836125
## V5  0.530343255

# Analytical form with intercept
solve(crossprod(X) + diag(c(0, rep(lambda, p)))) %*% t(X) %*% y
##                      [,1]
## (Intercept) -0.007775041
## x1          -0.517108905
## x2          -0.285489929
## x3          -0.018898839
## x4           0.236836125
## x5           0.530343255
```

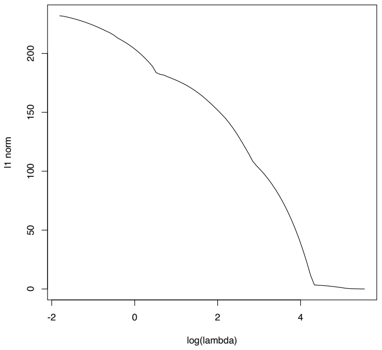
### 4.1.2 *Lasso*

The main novelty in lasso with respect to ridge is its ability to exactly zeroing coefficients and the lack of analytical solution. Fitting, tuning parameter selection, and prediction are completely analogous to ridge regression.

#### Fitting

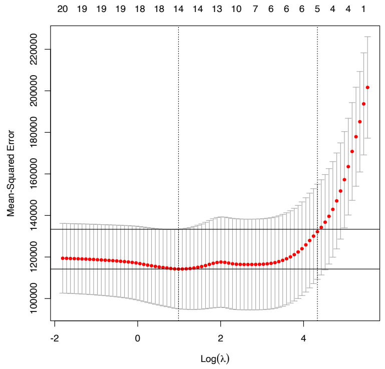




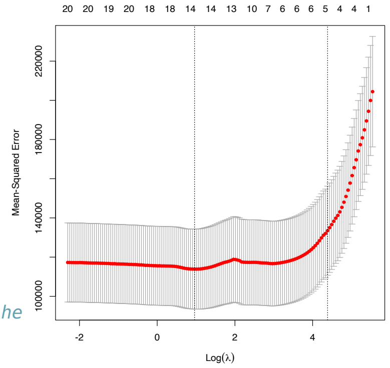


```
# Get the Hitters data back
x <- model.matrix(Salary ~ 0 + ., data = Hitters)
y <- Hitters$Salary

# 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 16 and 14

# 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 minimizes the CV error
kcvLasso$lambda.min
## [1] 2.674375

# The "one standard error rule" for lambda
kcvLasso$lambda.1se
```

```
## [1] 76.16717

# Location of both optimal lambdas in the CV loss function
indMin <- which.min(kcvLasso$cvm)
plot(kcvLasso)
abline(h = kcvLasso$cvm[indMin] + c(0, kcvLasso$cvsd[indMin]))
```

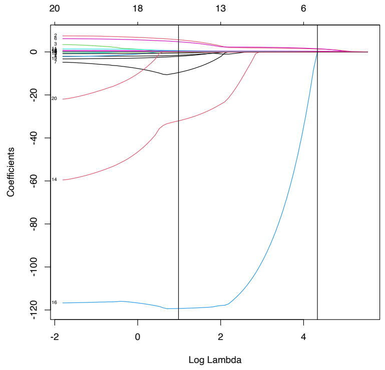


```
# No problems now: the minimum does not occur at one extreme
# Interesting: note that the numbers on top of the figure give the number of
# coefficients *exactly* different from zero -- the number of predictors
# effectively considered in the model!
# In this case, the one standard error rule makes also sense

# Leave-one-out cross-validation
lambdaGrid <- 10^seq(log10(kcvLasso$lambda[1]), log10(0.1),
                     length.out = 150) # log-spaced grid
ncvLasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = nrow(Hitters),
                      lambda = lambdaGrid)

# Location of both optimal lambdas in the CV loss function
plot(ncvLasso)
```



## Prediction

```
# Inspect the best models
plot(kcvLasso$glmnet.fit, label = TRUE, xvar = "lambda")
abline(v = log(c(kcvLasso$lambda.min, kcvLasso$lambda.1se)))
```



```
# The model associated with `lambda.min` (or any other lambda not included in the
# original path solution -- obtained by an interpolation) can be retrieved with
predict(kcvLasso, type = "coefficients",
        s = c(kcvLasso$lambda.min, kcvLasso$lambda.1se))
## 21 x 2 sparse Matrix of class "dgCMatrix"
##                        s1           s2
## (Intercept)  1.558172e+02 144.37970458
## AtBat       -1.547343e+00   .
## Hits         5.660897e+00   1.36380384
## HmRun        .              .
## Runs         .              .
## RBI          .              .
## Walks        4.729691e+00   1.49731098
## Years       -9.595837e+00   .
## CAtBat       .              .
## CHits        .              .
## CHmRun       5.108207e-01   .
## CRuns        6.594856e-01   0.15275165
## CRBI         3.927505e-01   0.32833941
## CWalks      -5.291586e-01   .
## LeagueA     -3.206508e+01   .
## LeagueN      2.108435e-12   .
## DivisionW   -1.192990e+02   .
## PutOuts      2.724045e-01   0.06625755
## Assists      1.732025e-01   .
## Errors      -2.058508e+00   .
## NewLeagueN   .              .

# Predictions for the first two observations
predict(kcvLasso, type = "response",
        s = c(kcvLasso$lambda.min, kcvLasso$lambda.1se),
        newx = x[1:2, ])
##                    s1       s2
## -Alan Ashby  427.8822 540.0835
## -Alvin Davis 700.1705 615.3311
```

### 4.1.3 Variable selection with lasso

Thanks to its ability to exactly zeroing coefficients, lasso is a **powerful device** for performing **variable/model selection** within its fit. The practical approach is really simple and amounts to identify the entries of $\hat{\beta}_{\lambda,1}$ different from zero, after $\lambda$ is appropriately selected.

```
# We can use lasso for model selection!
selPreds <- predict(kcvLasso, 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
modLassoSel1 <- lm(y ~ x1)
modLassoSel2 <- lm(y ~ x2)
summary(modLassoSel1)
##
## Call:
## lm(formula = y ~ x1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -940.10 -174.20  -25.94  127.05 1890.12
##
## Coefficients: (1 not defined because of singularities)
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  224.24408   84.45013   2.655 0.008434 **
## x1AtBat       -2.30798    0.56236  -4.104  5.5e-05 ***
## x1Hits         7.34602    1.71760   4.277  2.7e-05 ***
## x1Walks        6.08610    1.57008   3.876 0.000136 ***
## x1Years      -13.60502   10.38333  -1.310 0.191310
## x1CHmRun       0.83633    0.84709   0.987 0.324457
## x1CRuns        0.90924    0.27662   3.287 0.001159 **
## x1CRBI         0.35734    0.36252   0.986 0.325229
## x1CWalks      -0.83918    0.27207  -3.084 0.002270 **
## x1LeagueA    -36.68460   40.73468  -0.901 0.368685
## x1LeagueN           NA         NA      NA       NA
## x1DivisionW -119.67399   39.32485  -3.043 0.002591 **
## x1PutOuts      0.29296    0.07632   3.839 0.000157 ***
## x1Assists      0.31483    0.20460   1.539 0.125142
## x1Errors      -3.23219    4.29443  -0.753 0.452373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 314 on 249 degrees of freedom
## Multiple R-squared:  0.5396, Adjusted R-squared:  0.5156
## F-statistic: 22.45 on 13 and 249 DF,  p-value: < 2.2e-16
summary(modLassoSel2)
##
## 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 <- MASS::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 variable selection is 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

# Another interesting possibility is to run a stepwise selection starting from
# the set of predictors selected by lasso. In this search, it is important to
# use direction = "both" (default) and define the scope argument adequately
f <- formula(paste("Salary ~", paste(names(which(selPreds[, 2])),
                                     collapse = " + ")))
start <- lm(f, data = Hitters) # Model with predictors selected by lasso
scope <- list(lower = lm(Salary ~ 1, data = Hitters), # No predictors
              upper = lm(Salary ~ ., data = Hitters)) # All the predictors
modBICFromLasso <- MASS::stepAIC(object = start, k = log(nrow(Hitters)),
                                 scope = scope, trace = 0)
summary(modBICFromLasso)
##
## Call:
## lm(formula = Salary ~ Hits + Walks + CRBI + PutOuts + AtBat +
##     Division, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -873.11 -181.72  -25.91  141.77 2040.47
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   91.51180   65.00006   1.408 0.160382
## Hits           7.60440    1.66254   4.574 7.46e-06 ***
## Walks          3.69765    1.21036   3.055 0.002488 **
## CRBI            0.64302    0.06443   9.979  < 2e-16 ***
## PutOuts         0.26431    0.07477   3.535 0.000484 ***
## AtBat          -1.86859    0.52742  -3.543 0.000470 ***
## DivisionW    -122.95153   39.82029  -3.088 0.002239 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 319.9 on 256 degrees of freedom
## Multiple R-squared:  0.5087, Adjusted R-squared:  0.4972
## F-statistic: 44.18 on 6 and 256 DF,  p-value: < 2.2e-16

# Comparison in terms of BIC, slight improvement with modBICFromLasso
BIC(modLassoSel1, modLassoSel2, modBICFromLasso, modBIC)
##                 df      BIC
## modLassoSel1    15 3839.690
## modLassoSel2     7 3834.434
## modBICFromLasso  8 3817.785
## modBIC          10 3818.320
```

**Exercise 4.1.** Consider `la-liga-2015-2016.xlsx` dataset. We aim to predict `Points` after removing the perfectly related linear variables with `Points`. Do the following:

a. Lasso regression. Select $\lambda$ by cross-validation. Obtain the estimated coefficients for the chosen lambda.
b. Use the predictors with non-null coefficients for creating a model with `lm`.
c. Summarize the model and check for multicollinearity.

It may happen that the cross-validation curve has an **"L"-shaped form** without a well-defined global minimum. This usually happens when only the intercept is significant and **none of the predictors are relevant** for explaining $Y$.

The code below illustrates the previous warning.

```
# Random data with predictors unrelated to the response
p <- 100
n <- 300
set.seed(123124)
x <- matrix(rnorm(n * p), n, p)
y <- 1 + rnorm(n)

# CV
lambdaGrid <- exp(seq(-10, 3, l = 200))
plot(cv.glmnet(x = x, y = y, alpha = 1, nfolds = n, lambda = lambdaGrid))
```

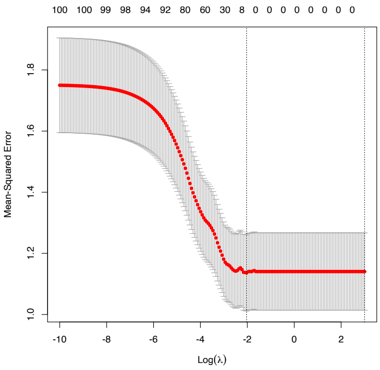


Figure 4.4: "L"-shaped form of a cross-validation curve with unrelated response and predictors.

The lasso-selection of variables is conceptually and practically straightforward. But, is this a consistent model selection procedure? As seen in Section 3.2.2, the answer to this question may be sometimes surprising and may have important practical consequences.

Zhao and Yu (2006) proved that **lasso is consistent** on the selection of the true model[19] **if a certain condition**, known as the *strong irrepresentable condition*, **holds for the predictors**. It is also required that the regularization parameter $\lambda \equiv \lambda_n$ is such that $\lambda_n \to 0$, at some specific speeds,[20] as $n \to \infty$. The strong irrepresentable condi-

[19] Understanding this statement as that the probability of lasso-selecting the predictors with slopes different from zero converges to one when $n \to \infty$.

[20] The details are quite technical and can be checked in Theorems 1–4 in Zhao and Yu (2006).

tion is quite technical, but essentially is satisfied if the correlations between the predictors are appropriately controlled. In particular, Zhao and Yu (2006) identify several simple situations for the predictors that guarantee that the strong irrepresentable condition holds:

- If the **predictors are uncorrelated**.
- If the **correlations of the predictors are bounded** by a certain constant. Precisely, if $\boldsymbol{\beta}_{-1}$ has $q$ entries different from zero, it must be satisfied that $\mathrm{Cor}[X_i, X_j] \leq \frac{c}{2q-1}$ for a constant $0 \leq c < 1$, for all $i, j = 1, \ldots, p$, $i \neq j$.[21]
- If the **correlations of the predictors are power-decaying**. That is, if $\mathrm{Cor}[X_i, X_j] = \rho^{|i-j|}$, for $|\rho| < 1$ and for all $i, j = 1, \ldots, p$.

Despite the usefulness of these three conditions, they do not inform directly on the consistency of the lasso in the more complex situation in which a data-driven penalizing parameter $\hat{\lambda}$ is used (as opposed to a deterministic sequence $\lambda_n \to 0$, as in Zhao and Yu (2006)). Let's explore this situation by retaking the simulation study behind Figure 3.5. Within the same settings described there, we lasso-select the predictors with estimated coefficients different from zero using $\hat{\lambda}_{n\text{-CV}}$ and $\hat{\lambda}_{n\text{-1SE}}$, then approximate by Monte Carlo the probability of selecting the true model. The results are collected in Figure 4.5.

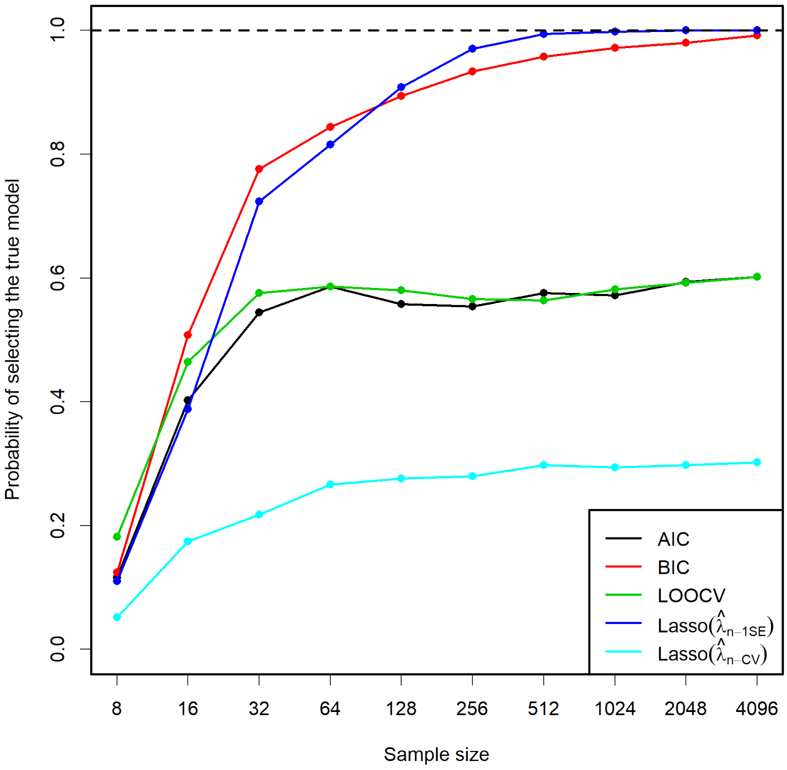


Figure 4.5: Estimation of the probability of selecting the correct model by lasso selection based on $\hat{\lambda}_{n\text{-CV}}$ and $\hat{\lambda}_{n\text{-1SE}}$, and by minimizing the AIC, BIC, and LOOCV criteria in an exhaustive search (see Figure 3.5). There are $p = 5$ independent predictors and the correct model contained two predictors. The probability was estimated with $M = 500$ Monte Carlo runs.

In addition to the insights from Figure 3.5, Figure 4.5 shows

[21] An example for $p = 5$ predictors follows. If $\boldsymbol{\beta}_{-1} = (1, -1, 0, 0, 0)^\top$, then there are $q = 2$ non-zero slopes. Therefore, $\frac{c}{2q-1} < \frac{1}{3}$ and the strong irrepresentable condition holds if all the correlations between the predictors are strictly smaller than $\frac{1}{3}$.

interesting results, described next. Recall that the simulation study was performed with *independent* predictors.

- Lasso-selection based on $\hat{\lambda}_{n\textbf{-CV}}$ **is inconsistent**. It tends to select more predictors than required, as AIC does, yet its performance is much worse (about a 0.25 probability of recovering the true model!). This result is somehow coherent what it would be expected from Shao (1993)'s result on the inconsistency of LOOCV.
- Lasso-selection based on $\hat{\lambda}_{n\textbf{-1SE}}$ ***seems to be***[22] **consistent** and imitates the performance of the BIC.[23] Observe that the overpenalization given by the one standard error rule somehow resembles the overpenalization of BIC with respect to AIC.

**Exercise 4.2.** Implement the lasso part of the simulation study behind Figure 4.5:

1. Sample from (3.4).
2. Compute the $\hat{\lambda}_{n\text{-CV}}$ and $\hat{\lambda}_{n\text{-1SE}}$.
3. Identify the estimated coefficients for $\hat{\lambda}_{n\text{-CV}}$ and $\hat{\lambda}_{n\text{-1SE}}$ that are different from zero.
4. Repeat Steps 1–3 $M = 200$ times. Estimate by Monte Carlo the probability of selecting the true model.
5. Move $n = 2^\ell$, $\ell = 3, \ldots, 12$.

Once you have a working solution, increase $M$ to approach the settings in Figure 4.5 (or go beyond!). You can increase also $p$ and pad $\boldsymbol{\beta}$ with zeros.

**Exercise 4.3.** Investigate what happens if Step 2 of the previous exercise is replaced by the computation of $\hat{\lambda}_{k\text{-CV}}$ and $\hat{\lambda}_{k\text{-1SE}}$, where:

a. $k = 2$.
b. $k = 4$.
c. $k = 8$.
d. $k = 16$.

Take $\ell = 3, \ldots, 12$ and overlay the eight curves together.

**Exercise 4.4.** Investigate what happens if Step 2 of the previous exercise is replaced by the computation of $\hat{\lambda}_{k\text{-CV}}$ and $\hat{\lambda}_{k\text{-1SE}}$, where:

a. $k = n/8$.
b. $k = n/4$.
c. $k = n/2$.
d. $k = n$.

Take $\ell = 3, \ldots, 12$ and overlay the eight curves together.

**Exercise 4.5.** Investigate what happens if Step 1 is replaced by sampling from *dependent* predictors. Particularly, sample from (3.4) but with $(X_1, \ldots, X_5)^\top \sim \mathcal{N}_5(\mathbf{0}, \boldsymbol{\Sigma})$, with $\boldsymbol{\Sigma} = (\sigma_{ij})$ such that:

a. $\sigma_{ij} = \rho^{|i-j|}$ for $i, j = 1, \ldots, 5$ and $\rho = 0.25, 0.50, 0.99$. *Hint*: use the `toeplitz` function.
b. $\sigma_{ij} = \frac{c}{2q-1}$ and $\sigma_{ii} = 1$, for $i, j = 1, \ldots, 5$, $i \neq j$, where $q = 2$ (number of non-zero entries of $\boldsymbol{\beta}$). Take $c = 0.75$.
c. Same as b., but with $c = 2$.

[22] Based on limited empirical evidence!

[23] Yet $\hat{\lambda}_{n\text{-1SE}}$ is much faster to compute that doing an exhaustive search with BIC!

## 4.2 Constrained linear models

As outlined in the previous section, after doing variable selection with lasso,[24] two possibilities are: (*i*) fit a linear model on the lasso-selected predictors; (*ii*) run a stepwise selection starting from the lasso-selected model to try to further improve the model.[25]

Let's explore the intuitive idea behind (*i*) in more detail. For the sake of exposition, assume that among $p$ predictors, lasso zeroed out the first $q$ of them.[26] Then, once $q$ is known, we would seek to fit the model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon, \quad \text{subject to} \quad \beta_1 = \cdots = \beta_q = 0.$$

This is a very simple constraint that we know how to solve: just include the $p - q$ remaining predictors in the model and fit it. It is however a specific case of a *linear constraint* on $\boldsymbol{\beta}$, since $\beta_1 = \cdots = \beta_q = 0$ is expressible as

$$\begin{pmatrix}\mathbf{I}_q & \mathbf{0}_{q\times(p-q)}\end{pmatrix}_{q\times p} \boldsymbol{\beta}_{-1} = \mathbf{0}_q, \tag{4.11}$$

where $\mathbf{I}_q$ is an $q \times q$ identity matrix and $\boldsymbol{\beta}_{-1} = (\beta_1, \ldots, \beta_p)^\top$. The constraint in (4.11) can be generalized as $\mathbf{A}\boldsymbol{\beta}_{-1} = \mathbf{c}$, which results in the (linearly) *constrained linear model*

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon, \quad \text{subject to} \quad \mathbf{A}\boldsymbol{\beta}_{-1} = \mathbf{c}, \tag{4.12}$$

where $\mathbf{A}$ is an $q \times p$ matrix[27] of *rank* $q$ and $\mathbf{c} \in \mathbb{R}^q$. The constrained linear model (4.12) is useful when there is prior information available about a linear relation that the coefficients of the linear model must satisfy (e.g., in piecewise polynomial fitting).

Before fitting the model (4.12), let's assume from now on that the variables $Y$ and $X_1, \ldots, X_p$, as well the sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$, are *centered* (see the tip at the end of Section 2.4.4). This means that $\bar{Y} = 0$ and that $\bar{\mathbf{X}} := (\bar{X}_1, \ldots, \bar{X}_p)^\top$ is zero. More importantly, it also means that $\beta_0$ and $\hat{\beta}_0$ are null, hence they are not included in the model. That is, that the model

$$Y = \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon \tag{4.13}$$

is considered. In this setting, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^\top$ [28] and $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y}$ is the least squares estimator, with the design matrix $\mathbf{X}$ now *omitting the first column of ones*.

Now, the estimator of $\boldsymbol{\beta}$ in (4.13) from a sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$ under the linear constraint $\mathbf{A}\boldsymbol{\beta} = \mathbf{c}$ is defined as

$$\hat{\boldsymbol{\beta}}_{\mathbf{A}} := \arg\min_{\substack{\boldsymbol{\beta}\in\mathbb{R}^p\\ \mathbf{A}\boldsymbol{\beta}=\mathbf{c}}} \mathrm{RSS}_0(\boldsymbol{\beta}), \quad \mathrm{RSS}_0(\boldsymbol{\beta}) := \sum_{i=1}^n (Y_i - \beta_1 X_{i1} - \cdots - \beta_p X_{ip})^2. \tag{4.14}$$

Solving (4.14) analytically is possible using Lagrange multipliers, and the explicit solution to (4.14) can be seen to be

$$\hat{\boldsymbol{\beta}}_{\mathbf{A}} = \hat{\boldsymbol{\beta}} + (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top[\mathbf{A}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top]^{-1}(\mathbf{c} - \mathbf{A}\hat{\boldsymbol{\beta}}). \tag{4.15}$$

[24] For example, based on the data-driven penalization parameters $\hat{\lambda}_{k\text{-CV}}$ or $\hat{\lambda}_{k\text{-1SE}}$.

[25] Note that with this approach we assign to the more computationally efficient lasso the "hard work" of coming up with a set of relevant predictors from the whole dataset, whereas the betterment of that model is done with the more demanding stepwise regression (if the number of predictors is smaller than $n$).

[26] Note that this is a random quantity, but we ignore this fact for the sake of exposition.

[27] Therefore, usually not invertible.

[28] We do not require the previous notation $\boldsymbol{\beta}_{-1}$!

For the general case given in (4.12), in which neither $Y$ and $\mathbf{X}$ nor the sample are centered, the estimator of $\boldsymbol{\beta}$ in (4.12) is unaltered for the slopes and equals (4.15). The intercept is given by

$$\hat{\beta}_{\mathbf{A},0} = \bar{Y} - \bar{\mathbf{X}}^\top \hat{\boldsymbol{\beta}}_{\mathbf{A}}.$$

The next code illustrates how to fit a linear model with constraints in practice.

```
# Simulate data
set.seed(123456)
n <- 50
p <- 3
x1 <- rnorm(n, mean = 1)
x2 <- rnorm(n, mean = 2)
x3 <- rnorm(n, mean = 3)
eps <- rnorm(n, sd = 0.5)
y <- 1 + 2 * x1 - 3 * x2 + x3 + eps

# Center the data and compute design matrix
x1Cen <- x1 - mean(x1)
x2Cen <- x2 - mean(x2)
x3Cen <- x3 - mean(x3)
yCen <- y - mean(y)
X <- cbind(x1Cen, x2Cen, x3Cen)

# Linear restriction: use that
# beta_1 + beta_2 + beta_3 = 0
# beta_2 = -3
# In this case q = 2. The restriction is codified as
A <- rbind(c(1, 1, 1),
           c(0, 1, 0))
c <- c(0, -3)

# Fit model without intercept
S <- solve(crossprod(X))
beta_hat <- S %*% t(X) %*% yCen
beta_hat
##             [,1]
## x1Cen  1.9873776
## x2Cen -3.1449015
## x3Cen  0.9828062

# Restricted fit enforcing A * beta = c
beta_hat_A <- beta_hat +
  S %*% t(A) %*% solve(A %*% S %*% t(A)) %*% (c - A %*% beta_hat)
beta_hat_A
##             [,1]
## x1Cen  2.0154729
## x2Cen -3.0000000
## x3Cen  0.9845271

# Intercept of the constrained fit
beta_hat_A_0 <- mean(y) - c(mean(x1), mean(x2), mean(x3)) %*% beta_hat_A
beta_hat_A_0
##         [,1]
## [1,] 1.02824
```

What about inference? In principle, it can be obtained analogously to how the inference for the unconstrained linear model was obtained in Section 2.4, since the distribution of $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$ under the assumptions of the linear model is straightforward to obtain. We keep assuming that the model is centered. Then, recall that (4.15) can be expressed as

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{\mathbf{A}} =& (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top[\mathbf{A}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top]^{-1}\mathbf{c}\\ &+\left(\mathbf{I}-(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top[\mathbf{A}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top]^{-1}\mathbf{A}\right)\hat{\boldsymbol{\beta}}.\end{aligned}$$

Therefore, using (1.4) and proceeding similarly to (2.11),

$$\hat{\boldsymbol{\beta}}_{\mathbf{A}} \sim \mathcal{N}_p\left(\boldsymbol{\beta}+b(\boldsymbol{\beta},\mathbf{A},\mathbf{c},\mathbf{X}),\sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}-v(\sigma^2,\mathbf{A},\mathbf{X})\right), \tag{4.16}$$

where

$$\begin{aligned}b(\boldsymbol{\beta},\mathbf{A},\mathbf{c},\mathbf{X}) &:= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top[\mathbf{A}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top]^{-1}(\mathbf{c}-\mathbf{A}\boldsymbol{\beta}),\\ v(\sigma^2,\mathbf{A},\mathbf{X}) &:= \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top[\mathbf{A}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{A}^\top]^{-1}\mathbf{A}(\mathbf{X}^\top\mathbf{X})^{-1}.\end{aligned}$$

The inference for constrained linear models is not built within base R. Therefore, we just give a couple of insights about (4.16) and do not pursue inference further. Note that:

- The **variances** of $\hat{\beta}_{\mathbf{A},j}$, $j=1,\ldots,p$, **decrease** with respect to the variances of $\hat{\beta}_j$, given by the diagonal elements of $\sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}$. This is perfectly coherent, after all we are constraining the possible values that the estimator of $\boldsymbol{\beta}$ can take in order to accommodate $\mathbf{A}\boldsymbol{\beta}=\mathbf{c}$. More importantly, these variances remain the same irrespective of whether $\mathbf{A}\boldsymbol{\beta}=\mathbf{c}$ holds or not.[29]
- The **bias of $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$ depends on the veracity of $\mathbf{A}\boldsymbol{\beta}=\mathbf{c}$**. If the restriction is verified, then $b(\boldsymbol{\beta},\mathbf{A},\mathbf{c},\mathbf{X})=\mathbf{0}$ and $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$ is *still* unbiased. However, if $\mathbf{A}\boldsymbol{\beta}\neq\mathbf{c}$, then $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$ is severely biased in estimating $\boldsymbol{\beta}$.

**Exercise 4.6.** Verify by Monte Carlo that the covariance matrix in (4.16) is correct. To do so:

1. Choose $\boldsymbol{\beta}$, $\mathbf{A}$, and $\mathbf{c}$ at your convenience.
2. Sample $n=50$ observations for the predictors.
3. Sample $n=50$ observations for the responses from a linear model based on $\boldsymbol{\beta}$. Use the same $n$ observations for the predictors from Step 2.
4. Compute $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$.
5. Repeat Steps 3–4 $M=500$ times, saving each time $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$.
6. Compute the sample covariance matrix of the $\hat{\boldsymbol{\beta}}_{\mathbf{A}}$'s.
7. Compare it with the covariance matrix in (4.16).

**Exercise 4.7.** Do the same study for checking the expectation in (4.16), for the cases in which $\mathbf{A}\boldsymbol{\beta}=\mathbf{c}$ and $\mathbf{A}\boldsymbol{\beta}\neq\mathbf{c}$.

## 4.3 *Multivariate multiple linear model*

So far, we have been interested in predicting/explaining a *single* response $Y$ from a set of predictors $X_1,\ldots,X_p$. However, we might want to predict/explain *several* responses $Y_1,\ldots,Y_q$.[30] As we will see, the model construction and estimation are quite analogous to the univariate multiple linear model, yet more cumbersome in notation.

[29] They do not depend on $\mathbf{c}$!

[30] Do not confuse them with $Y_1,\ldots,Y_n$, the notation employed in the rest of the sections for denoting the sample of *the* response $Y$.

### 4.3.1 Model formulation and least squares

The *centered*[31] population version of the *multivariate multiple linear model* is

$$\begin{aligned}Y_1 &= \beta_{11}X_1 + \cdots + \beta_{p1}X_p + \varepsilon_1,\\ &\vdots\\ Y_q &= \beta_{1q}X_1 + \cdots + \beta_{pq}X_p + \varepsilon_q,\end{aligned}$$

or, equivalently in matrix form,

$$\mathbf{Y} = \mathbf{B}^\top \mathbf{X} + \boldsymbol{\varepsilon} \tag{4.17}$$

where $\boldsymbol{\varepsilon} := (\varepsilon_1, \ldots, \varepsilon_q)^\top$ is a random vector with null expectation, $\mathbf{Y} = (Y_1, \ldots, Y_q)^\top$ and $\mathbf{X} = (X_1, \ldots, X_p)^\top$ are random vectors, and

$$\mathbf{B} = \begin{pmatrix}\beta_{11} & \ldots & \beta_{1q}\\ \vdots & \ddots & \vdots\\ \beta_{p1} & \ldots & \beta_{pq}\end{pmatrix}_{p\times q}.$$

Clearly, this construction implies that the conditional expectation of the random vector $\mathbf{Y}$ is

$$\mathbb{E}[\mathbf{Y} \mid \mathbf{X} = \mathbf{x}] = \mathbf{B}^\top \mathbf{x}.$$

Given a sample $\{(\mathbf{X}_i, \mathbf{Y}_i)\}_{i=1}^n$ of observations of $(X_1, \ldots, X_p)$ and $(Y_1, \ldots, Y_q)$, the sample version of (4.17) is

$$\begin{pmatrix}Y_{11} & \ldots & Y_{1q}\\ \vdots & \ddots & \vdots\\ Y_{n1} & \ldots & Y_{nq}\end{pmatrix}_{n\times q} = \begin{pmatrix}X_{11} & \ldots & X_{1p}\\ \vdots & \ddots & \vdots\\ X_{n1} & \ldots & X_{np}\end{pmatrix}_{n\times p} \begin{pmatrix}\beta_{11} & \ldots & \beta_{1q}\\ \vdots & \ddots & \vdots\\ \beta_{p1} & \ldots & \beta_{pq}\end{pmatrix}_{p\times q} + \begin{pmatrix}\varepsilon_{11} & \ldots & \varepsilon_{1q}\\ \vdots & \ddots & \vdots\\ \varepsilon_{n1} & \ldots & \varepsilon_{nq}\end{pmatrix}_{n\times q}, \tag{4.18}$$

or, equivalently in matrix form,

$$\mathbb{Y} = \mathbb{X}\mathbf{B} + \mathbf{E}, \tag{4.19}$$

where $\mathbb{Y}$, $\mathbb{X}$, and $\mathbf{E}$ are clearly identified by comparing (4.19) with (4.18).[32]

The approach for estimating $\mathbf{B}$ is really similar to the univariate multiple linear model: minimize the sum of squared distances between the responses $\mathbf{Y}_1, \ldots, \mathbf{Y}_n$ and their explanations $\mathbf{B}^\top\mathbf{X}_1, \ldots, \mathbf{B}^\top\mathbf{X}_n$. These distances are now measured by the $\|\cdot\|_2$ norm in $\mathbb{R}^q$, resulting

$$\begin{aligned}\mathrm{RSS}(\mathbf{B}) &:= \sum_{i=1}^n \|\mathbf{Y}_i - \mathbf{B}^\top\mathbf{X}_i\|_2^2\\ &= \sum_{i=1}^n (\mathbf{Y}_i - \mathbf{B}^\top\mathbf{X}_i)^\top(\mathbf{Y}_i - \mathbf{B}^\top\mathbf{X}_i)\\ &= \mathrm{tr}\left((\mathbb{Y} - \mathbb{X}\mathbf{B})^\top(\mathbb{Y} - \mathbb{X}\mathbf{B})\right).\end{aligned}$$

[31] Centering the responses and predictors is useful for removing the intercept term, allowing for simpler matricial versions.

[32] The notation $\mathbb{X}$ is introduced to avoid confusions between the design matrix $\mathbb{X}$ and the *random vector* $\mathbf{X}$ and observations $\mathbf{X}_i$, $i = 1, \ldots, n$.

[33] Employing the centered version of the univariate multiple linear model, as we have done in this section for the multivariate version.

The similarities with (2.6) are clear and it is immediate to see that (2.6) appears as a special case[33] for $q = 1$. The minimizer of $\mathrm{RSS}(\mathbf{B})$ is obtained analogously to how the least squares estimator was obtained when $q = 1$:

$$\hat{\mathbf{B}} := \arg\min_{\mathbf{B}\in\mathcal{M}_{p\times q}} \mathrm{RSS}(\mathbf{B}) = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\mathbb{Y}. \tag{4.20}$$

Recall that if the responses and predictors are not centered, then the estimate of the intercept is simply obtained from the sample means $\bar{\mathbf{Y}} := (\bar{Y}_1, \ldots, \bar{Y}_q)^\top$ and $\bar{\mathbf{X}} = (\bar{X}_1, \ldots, \bar{X}_p)^\top$:

$$\hat{\boldsymbol{\beta}}_0 = \bar{\mathbf{Y}} - \hat{\mathbf{B}}^\top\bar{\mathbf{X}}.$$

Equation (4.20) reveals that **fitting a $q$-multivariate linear model amounts to fitting $q$ univariate linear models separately!** Indeed, recall that $\mathbf{B} = (\boldsymbol{\beta}_1 \cdots \boldsymbol{\beta}_q)$, where the column vector $\boldsymbol{\beta}_j$ represents the vector of coefficients of the $j$-th univariate linear model. Then, comparing (4.20) with (2.7) (where $\mathbb{Y}$ consisted of a single column) and by block matrix multiplication, we can clearly see that $\hat{\mathbf{B}}$ is just the concatenation of the columns of $\hat{\boldsymbol{\beta}}_j$, $j = 1, \ldots, q$, i.e., $\hat{\mathbf{B}} = (\hat{\boldsymbol{\beta}}_1 \cdots \hat{\boldsymbol{\beta}}_q)$.

As happened in the univariate linear model, if $p > n$ then the inverse of $\mathbb{X}^\top\mathbb{X}$ in (4.20) does not exist. In that case, one should either remove predictors or resort to a shrinkage method that avoids inverting $\mathbb{X}^\top\mathbb{X}$. It is interesting to note, though, that **$q$ has no effect on the feasibility of the fitting, only $p$ does**. In particular it is possible to compute (4.20) with $q \gg n$, and hence $pq \gg n$, i.e., **the number of estimated parameters can be much larger than $n$.**

We see next how to do multivariate multiple linear regression in R with a simulated example.

```r
# Dimensions and sample size
p <- 3
q <- 2
n <- 100

# A quick way of creating a non-diagonal (valid) covariance matrix for the
# errors
Sigma <- 3 * toeplitz(seq(1, 0.1, l = q))
set.seed(12345)
X <- mvtnorm::rmvnorm(n = n, mean = 1:p, sigma = diag(0.5, nrow = p, ncol = p))
E <- mvtnorm::rmvnorm(n = n, mean = rep(0, q), sigma = Sigma)

# Linear model
B <- matrix((-1)^(1:p) * (1:p), nrow = p, ncol = q, byrow = TRUE)
Y <- X %*% B + E

# Fitting the model (note: Y and X are matrices!)
```

```
mod <- lm(Y ~ X)
mod
##
## Call:
## lm(formula = Y ~ X)
##
## Coefficients:
##              [,1]      [,2]
## (Intercept)   0.05017  -0.36899
## X1           -0.54770   2.06905
## X2           -3.01547  -0.78308
## X3            1.88327  -3.00840
# Note that the intercept is markedly different from zero -- that is because
# X is not centered

# Compare with B
B
##      [,1] [,2]
## [1,]   -1    2
## [2,]   -3   -1
## [3,]    2   -3

# Summary of the model: gives q separate summaries, one for each fitted
# univariate model
summary(mod)
## Response Y1 :
##
## Call:
## lm(formula = Y1 ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0432 -1.3513  0.2592  1.1325  3.5298
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05017    0.96251   0.052   0.9585
## X1          -0.54770    0.24034  -2.279   0.0249 *
## X2          -3.01547    0.26146 -11.533  < 2e-16 ***
## X3           1.88327    0.21537   8.745 7.38e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.695 on 96 degrees of freedom
## Multiple R-squared:  0.7033, Adjusted R-squared:  0.694
## F-statistic: 75.85 on 3 and 96 DF,  p-value: < 2.2e-16
##
##
## Response Y2 :
##
## Call:
## lm(formula = Y2 ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1385 -0.7922 -0.0486  0.8987  3.6599
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.3690     0.8897  -0.415  0.67926
## X1            2.0691     0.2222   9.314 4.44e-15 ***
## X2           -0.7831     0.2417  -3.240  0.00164 **
## X3           -3.0084     0.1991 -15.112  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.567 on 96 degrees of freedom
## Multiple R-squared:  0.7868, Adjusted R-squared:  0.7801
## F-statistic: 118.1 on 3 and 96 DF,  p-value: < 2.2e-16

# Exactly equivalent to
summary(lm(Y[, 1] ~ X))
##
## Call:
## lm(formula = Y[, 1] ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0432 -1.3513  0.2592  1.1325  3.5298
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05017    0.96251   0.052   0.9585
## X1          -0.54770    0.24034  -2.279   0.0249 *
## X2          -3.01547    0.26146 -11.533  < 2e-16 ***
## X3           1.88327    0.21537   8.745 7.38e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.695 on 96 degrees of freedom
## Multiple R-squared:  0.7033, Adjusted R-squared:  0.694
## F-statistic: 75.85 on 3 and 96 DF,  p-value: < 2.2e-16
summary(lm(Y[, 2] ~ X))
##
## Call:
## lm(formula = Y[, 2] ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1385 -0.7922 -0.0486  0.8987  3.6599
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.3690     0.8897  -0.415  0.67926
## X1            2.0691     0.2222   9.314 4.44e-15 ***
## X2           -0.7831     0.2417  -3.240  0.00164 **
## X3           -3.0084     0.1991 -15.112  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.567 on 96 degrees of freedom
## Multiple R-squared:  0.7868, Adjusted R-squared:  0.7801
## F-statistic: 118.1 on 3 and 96 DF,  p-value: < 2.2e-16
```

Let's see another quick example using the `iris` dataset.

```
# When we want to add several variables of a dataset as responses through a
# formula interface, we have to use cbind() in the response. Doing
# "Petal.Width + Petal.Length ~ ..." is INCORRECT, as lm will understand
# "I(Petal.Width + Petal.Length) ~ ..." and do one single regression

# Predict Petal's measurements from Sepal's
modIris <- lm(cbind(Petal.Width, Petal.Length) ~
                Sepal.Length + Sepal.Width + Species, data = iris)
summary(modIris)
## Response Petal.Width :
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length + Sepal.Width + Species,
##     data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.50805 -0.10042 -0.01221  0.11416  0.46455
##
## Coefficients:
```

```
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.86897    0.16985  -5.116 9.73e-07 ***
## Sepal.Length       0.06360    0.03395   1.873    0.063 .
## Sepal.Width        0.23237    0.05145   4.516 1.29e-05 ***
## Speciesversicolor  1.17375    0.06758  17.367  < 2e-16 ***
## Speciesvirginica   1.78487    0.07779  22.944  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1797 on 145 degrees of freedom
## Multiple R-squared:  0.9459, Adjusted R-squared:  0.9444
## F-statistic: 634.3 on 4 and 145 DF,  p-value: < 2.2e-16
##
##
## Response Petal.Length :
##
## Call:
## lm(formula = Petal.Length ~ Sepal.Length + Sepal.Width + Species,
##     data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.75196 -0.18755  0.00432  0.16965  0.79580
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -1.63430    0.26783  -6.102 9.08e-09 ***
## Sepal.Length       0.64631    0.05353  12.073  < 2e-16 ***
## Sepal.Width       -0.04058    0.08113  -0.500    0.618
## Speciesversicolor  2.17023    0.10657  20.364  < 2e-16 ***
## Speciesvirginica   3.04911    0.12267  24.857  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2833 on 145 degrees of freedom
## Multiple R-squared:  0.9749, Adjusted R-squared:  0.9742
## F-statistic:  1410 on 4 and 145 DF,  p-value: < 2.2e-16

# The fitted values and residuals are now matrices
head(modIris$fitted.values)
##   Petal.Width Petal.Length
## 1   0.2687095     1.519831
## 2   0.1398033     1.410862
## 3   0.1735565     1.273483
## 4   0.1439590     1.212910
## 5   0.2855861     1.451142
## 6   0.3807391     1.697490
head(modIris$residuals)
##   Petal.Width Petal.Length
## 1 -0.06870951 -0.119831001
## 2  0.06019672 -0.010861533
## 3  0.02644348  0.026517420
## 4  0.05604099  0.287089900
## 5 -0.08558613 -0.051141525
## 6  0.01926089  0.002510054

# The individual models
modIris1 <- lm(Petal.Width ~Sepal.Length + Sepal.Width + Species, data = iris)
modIris2 <- lm(Petal.Length ~Sepal.Length + Sepal.Width + Species, data = iris)
summary(modIris1)
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length + Sepal.Width + Species,
##     data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.50805 -0.10042 -0.01221  0.11416  0.46455
```

```
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.86897    0.16985  -5.116 9.73e-07 ***
## Sepal.Length       0.06360    0.03395   1.873    0.063 .
## Sepal.Width        0.23237    0.05145   4.516 1.29e-05 ***
## Speciesversicolor  1.17375    0.06758  17.367  < 2e-16 ***
## Speciesvirginica   1.78487    0.07779  22.944  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1797 on 145 degrees of freedom
## Multiple R-squared:  0.9459, Adjusted R-squared:  0.9444
## F-statistic: 634.3 on 4 and 145 DF,  p-value: < 2.2e-16
summary(modIris2)
##
## Call:
## lm(formula = Petal.Length ~ Sepal.Length + Sepal.Width + Species,
##     data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.75196 -0.18755  0.00432  0.16965  0.79580
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -1.63430    0.26783  -6.102 9.08e-09 ***
## Sepal.Length       0.64631    0.05353  12.073  < 2e-16 ***
## Sepal.Width       -0.04058    0.08113  -0.500    0.618
## Speciesversicolor  2.17023    0.10657  20.364  < 2e-16 ***
## Speciesvirginica   3.04911    0.12267  24.857  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2833 on 145 degrees of freedom
## Multiple R-squared:  0.9749, Adjusted R-squared:  0.9742
## F-statistic:  1410 on 4 and 145 DF,  p-value: < 2.2e-16
```

### 4.3.2 *Assumptions and inference*

As deduced from what we have seen so far, *fitting* a multivariate linear regression is more practical than doing $q$ separate univariate fits (especially if the number of responses $q$ is large). However, it is not conceptually different. The discussion becomes more interesting in the *inference* for the multivariate linear regression, where the dependence between the responses has to be taken into account.

In order to achieve inference, we will require some assumptions, these being natural extensions of the ones seen in Section 2.3:

i. **Linearity**: $\mathbb{E}[\mathbf{Y} \mid \mathbf{X} = \mathbf{x}] = \mathbf{B}^\top \mathbf{x}$.
ii. **Homoscedasticity**: $\mathbb{V}\mathrm{ar}[\boldsymbol{\varepsilon} \mid X_1 = x_1, \ldots, X_p = x_p] = \boldsymbol{\Sigma}$.
iii. **Normality**: $\boldsymbol{\varepsilon} \sim \mathcal{N}_q(\mathbf{0}, \boldsymbol{\Sigma})$.
iv. **Independence of the errors**: $\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_n$ are independent (or uncorrelated, $\mathbb{E}[\boldsymbol{\varepsilon}_i \boldsymbol{\varepsilon}_j^\top] = \mathbf{0}$, $i \neq j$, since they are assumed to be normal).

Then, a good one-line summary of the multivariate multiple linear model is (independence is implicit)

$$\mathbf{Y} \mid \mathbf{X} = \mathbf{x} \sim \mathcal{N}_q(\mathbf{B}^\top \mathbf{x}, \boldsymbol{\Sigma}).$$

Based on these assumptions, the key result for rooting inference is the distribution of $\hat{\mathbf{B}} = (\hat{\boldsymbol\beta}_1 \cdots \hat{\boldsymbol\beta}_q)$ as an estimator of $\mathbf{B} = (\boldsymbol\beta_1 \cdots \boldsymbol\beta_q)$. This result is now more cumbersome,[34] but we can state it as

$$\hat{\boldsymbol\beta}_j \sim \mathcal{N}_p\left(\boldsymbol\beta_j, \sigma_j^2(\mathbb{X}^\top\mathbb{X})^{-1}\right), \quad j = 1, \ldots, q, \tag{4.21}$$

$$\begin{pmatrix}\hat{\boldsymbol\beta}_j\\ \hat{\boldsymbol\beta}_k\end{pmatrix} \sim \mathcal{N}_{2p}\left(\begin{pmatrix}\boldsymbol\beta_j\\ \boldsymbol\beta_k\end{pmatrix}, \begin{pmatrix}\sigma_j^2(\mathbb{X}^\top\mathbb{X})^{-1} & \sigma_{jk}(\mathbb{X}^\top\mathbb{X})^{-1}\\ \sigma_{jk}(\mathbb{X}^\top\mathbb{X})^{-1} & \sigma_k^2(\mathbb{X}^\top\mathbb{X})^{-1}\end{pmatrix}\right), \quad j,k = 1, \ldots, q, \tag{4.22}$$

where $\boldsymbol\Sigma = (\sigma_{ij})$ and $\sigma_{ii} = \sigma_i^2$.[35]

The results (4.21)–(4.22) open the way for obtaining hypothesis tests on the *joint* significance of a predictor in the model (for the $q$ responses, not just for one), confidence intervals for the coefficients, prediction confidence *regions* for the conditional expectation and the conditional response, the *Multivariate* ANOVA (MANOVA) decomposition, multivariate extensions of the $F$-test, and others. However, due to the **correlation between responses** and the multivariateness, these inferential tools are **more complex** than in the univariate linear model. Therefore, given the increased complexity, we do not go into more details and refer the interested reader to, e.g., Chapter 8 in Seber (1984). We illustrate with code, though, the most important practical aspects.

```
# Confidence intervals for the parameters
confint(modIris)
##                                        2.5 %     97.5 %
## Petal.Width:(Intercept)         -1.204674903 -0.5332662
## Petal.Width:Sepal.Length        -0.003496659  0.1307056
## Petal.Width:Sepal.Width          0.130680383  0.3340610
## Petal.Width:Speciesversicolor    1.040169583  1.3073259
## Petal.Width:Speciesvirginica     1.631118293  1.9386298
## Petal.Length:(Intercept)        -2.163654566 -1.1049484
## Petal.Length:Sepal.Length        0.540501864  0.7521177
## Petal.Length:Sepal.Width        -0.200934599  0.1197646
## Petal.Length:Speciesversicolor   1.959595164  2.3808588
## Petal.Length:Speciesvirginica    2.806663658  3.2915610
# Warning! Do not confuse Petal.Width:Sepal.Length with an interaction term!
# It is meant to represent the Response:Predictor coefficient

# Prediction -- now more limited without confidence intervals implemented
predict(modIris, newdata = iris[1:3, ])
##   Petal.Width Petal.Length
## 1   0.2687095     1.519831
## 2   0.1398033     1.410862
## 3   0.1735565     1.273483

# MANOVA table
manova(modIris)
## Call:
##    manova(modIris)
##
## Terms:
##                  Sepal.Length Sepal.Width  Species Residuals
## Petal.Width           57.9177      6.3975  17.5745    4.6802
## Petal.Length         352.8662     50.0224  49.7997   11.6371
## Deg. of Freedom             1           1        2       145
##
## Residual standard errors: 0.1796591 0.2832942
## Estimated effects may be unbalanced
```

[34] Indeed, specifying the full distribution of $\hat{\mathbf{B}}$ would require introducing the *matrix* normal distribution, a generalization of the $p$-dimensional normal seen in Section 1.2.

[35] Observe how the covariance of the errors $\varepsilon_j$ and $\varepsilon_k$, denoted by $\sigma_{jk}$, is the responsible of the correlation between $\hat{\boldsymbol\beta}_j$ and $\hat{\boldsymbol\beta}_k$ in (4.22). If $\sigma_{jk} = 0$, then $\hat{\boldsymbol\beta}_j$ and $\hat{\boldsymbol\beta}_k$ would be uncorrelated, thus independent because of their joint normality. Therefore, inference on $\boldsymbol\beta_j$ and $\boldsymbol\beta_k$ could be carried out *separately*.

```
# "Same" as the "Sum Sq" and "Df" entries of
anova(modIris1)
## Analysis of Variance Table
##
## Response: Petal.Width
##               Df Sum Sq Mean Sq F value    Pr(>F)
## Sepal.Length   1 57.918  57.918 1794.37 < 2.2e-16 ***
## Sepal.Width    1  6.398   6.398  198.21 < 2.2e-16 ***
## Species        2 17.574   8.787  272.24 < 2.2e-16 ***
## Residuals    145  4.680   0.032
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
anova(modIris2)
## Analysis of Variance Table
##
## Response: Petal.Length
##               Df Sum Sq Mean Sq F value    Pr(>F)
## Sepal.Length   1 352.87  352.87 4396.78 < 2.2e-16 ***
## Sepal.Width    1  50.02   50.02  623.29 < 2.2e-16 ***
## Species        2  49.80   24.90  310.26 < 2.2e-16 ***
## Residuals    145  11.64    0.08
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# anova() serves for assessing the significance of including a new predictor
# for explaining all the responses. This is based on an extension of the
# *sequential* ANOVA table briefly covered in Section 2.6. The hypothesis test
# is by default conducted with the Pillai statistic (an extension of the F-test)
anova(modIris)
## Analysis of Variance Table
##
##               Df  Pillai approx F num Df den Df    Pr(>F)
## (Intercept)    1 0.99463  13332.6      2    144 < 2.2e-16 ***
## Sepal.Length   1 0.97030   2351.9      2    144 < 2.2e-16 ***
## Sepal.Width    1 0.81703    321.5      2    144 < 2.2e-16 ***
## Species        2 0.89573     58.8      4    290 < 2.2e-16 ***
## Residuals    145
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 4.3.3 *Shrinkage*

Applying shrinkage is also possible in multivariate linear models. In particular, this allows to fit models with $p \gg n$ predictors.

The `glmnet` package implements the elastic net regularization for multivariate linear models. It features extensions of the $\|\cdot\|_1$ and $\|\cdot\|_2$ norm penalties for a *vector* of parameters in $\mathbb{R}^p$, as considered in Section 4.1, to norms for a matrix of parameters of size $p \times q$. Precisely:

- The **ridge penalty** $\|\boldsymbol{\beta}_{-1}\|_2^2$ extends to $\|\mathbf{B}\|_\mathrm{F}^2$, where $\|\mathbf{B}\|_\mathrm{F} = \sqrt{\sum_{i=1}^p \sum_{j=1}^q \beta_{ij}^2}$ is the *Frobenius norm* of $\mathbf{B}$. This is a *global penalty* to shrink $\mathbf{B}$.
- The **lasso penalty** $\|\boldsymbol{\beta}_{-1}\|_1$ extends[36] to $\sum_{j=1}^p \|\mathbf{B}_j\|_2$, where $\mathbf{B}_j$ is the $j$-th *row* of $\mathbf{B}$. This is a *rowwise penalty* that seeks to effectively remove rows of $\mathbf{B}$, thus eliminating predictors.[37]

Taking these two extensions into account, the elastic net loss is

[36] If $q = 1$, then $\sum_{j=1}^p \|\mathbf{B}_j\|_2 = \sum_{j=1}^p \sqrt{\beta_{j1}^2} = \sum_{j=1}^p |\beta_{j1}|$.

[37] This property would not hold if the penalty $\sum_{i=1}^p \sum_{j=1}^q |\beta_{ij}|$ was considered.

defined as:

$$\mathrm{RSS}(\mathbf{B}) + \lambda \left( \alpha \sum_{j=1}^{p} \|\mathbf{B}_j\|_2 + (1-\alpha)\|\mathbf{B}\|_{\mathrm{F}}^2 \right). \tag{4.23}$$

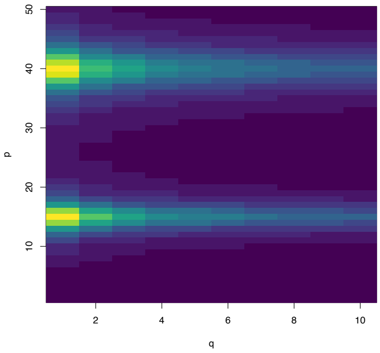
Clearly, ridge regression corresponds to $\alpha = 0$ (quadratic penalty) and lasso to $\alpha = 1$ (rowwise penalty). And if $\lambda = 0$, we are back to the least squares problem and theory. The optimization of (4.23) gives

$$\hat{\mathbf{B}}_{\lambda,\alpha} := \arg \min_{\mathbf{B} \in \mathcal{M}_{p \times q}} \left\{ \mathrm{RSS}(\mathbf{B}) + \lambda \left( \alpha \sum_{j=1}^{p} \|\mathbf{B}_j\|_2 + (1-\alpha)\|\mathbf{B}\|_{\mathrm{F}}^2 \right) \right\}.$$

From here, the workflow is very similar to the univariate linear model: we have to be aware that a standardization of $\mathbb{X}$ and $\mathbb{Y}$ takes place in `glmnet`; there are explicit formulas for the ridge regression estimator, but not for lasso; tuning parameter selection of $\lambda$ is done by $k$-fold cross-validation and its one standard error variant; variable selection (zeroing of rows in $\mathbf{B}$) can be done with lasso.

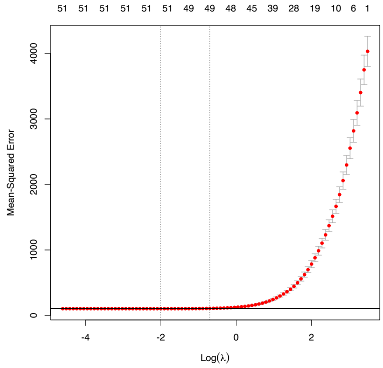
The following chunk of code illustrates some of these points using `glmnet::glmnet` with `family = "mgaussian"` (do not forget this argument!).

```
# Simulate data
n <- 500
p <- 50
q <- 10
set.seed(123456)
X <- mvtnorm::rmvnorm(n = n, mean = p:1, sigma = 5 * 0.5^toeplitz(1:p))
E <- mvtnorm::rmvnorm(n = n, mean = rep(0, q), sigma = toeplitz(q:1))
B <- 5 * (2 / (0.5 * (1:p - 15)^2 + 2) + 1 / (0.1 * (1:p - 40)^2 + 1)) %*%
  t(1 / sqrt(1:q))
Y <- X %*% B + E

# Visualize B -- dark violet is close to 0
image(1:q, 1:p, t(B), col = viridisLite::viridis(20), xlab = "q", ylab = "p")
```



```
# Lasso path fit
mfit <- glmnet(x = X, y = Y, family = "mgaussian", alpha = 1)

# A list of models for each response
str(mfit$beta, 1)
## List of 10
##  $ y1 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y2 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y3 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y4 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y5 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y6 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y7 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y8 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y9 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y10:Formal class 'dgCMatrix' [package "Matrix"] with 6 slots

# Tuning parameter selection by 10-fold cross-validation
set.seed(12345)
kcvLassoM <- cv.glmnet(x = X, y = Y, family = "mgaussian", alpha = 1)
kcvLassoM$lambda.min
## [1] 0.135243
```

```
kcvLassoM$lambda.1se
## [1] 0.497475

# Location of both optimal lambdas in the CV loss function
indMin <- which.min(kcvLassoM$cvm)
plot(kcvLassoM)
abline(h = kcvLassoM$cvm[indMin] + c(0, kcvLassoM$cvsd[indMin]))
```



```
# Extract the coefficients associated with some fits
coefs <- predict(kcvLassoM, type = "coefficients",
                 s = c(kcvLassoM$lambda.min, kcvLassoM$lambda.1se))
str(coefs, 1)
## List of 10
##  $ y1 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y2 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y3 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y4 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y5 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y6 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y7 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y8 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y9 :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##  $ y10:Formal class 'dgCMatrix' [package "Matrix"] with 6 slots

# Predictions for the first two observations
preds <- predict(kcvLassoM, type = "response",
                 s = c(kcvLassoM$lambda.min, kcvLassoM$lambda.1se),
                 newx = X[1:2, ])
preds
## , , 1
##
##            y1       y2       y3       y4       y5       y6       y7       y8       y9      y10
## [1,] 1607.532 1137.740 928.9553 804.5352 719.4861 656.1383 607.9023 568.9489 536.3892 508.3132
## [2,] 1598.747 1130.989 923.1741 799.3845 715.3698 652.8120 604.2244 565.1894 531.9679 504.8820
##
## , , 2
##
##            y1       y2       y3       y4       y5       y6       y7       y8       y9      y10
## [1,] 1606.685 1137.092 928.4553 804.0957 719.0616 655.7909 607.5870 568.5401 536.0135 508.0311
## [2,] 1600.321 1132.040 924.0621 800.1538 715.9371 653.3917 604.7983 565.6777 532.6244 505.4317
```

Finally, the next animation helps visualizing how the zeroing of the lasso happens for the estimator of **B** with overall low absolute values on the previous simulated model.

```
manipulate::manipulate({

  # Color
  col <- viridisLite::viridis(20)

  # Common zlim
  zlim <- range(B) + c(-0.25, 0.25)

  # Plot true B
  par(mfrow = c(1, 2))
  image(1:q, 1:p, t(B), col = col, xlab = "q", ylab = "p", zlim = zlim,
        main = "B")

  # Extract B_hat from the lasso fit, a p x q matrix
  B_hat <- sapply(seq_along(mfit$beta), function(i) mfit$beta[i][[1]][, j])

  # Put as black rows the predictors included
  not_zero <- abs(B_hat) > 0
  image(1:q, 1:p, t(not_zero), breaks = c(0.5, 1),
        col = rgb(1, 1, 1, alpha = 0.1), add = TRUE)
  # For B_hat
  image(1:q, 1:p, t(B_hat), col = col, xlab = "q", ylab = "p", zlim = zlim,
        main = "Bhat")
  image(1:q, 1:p, t(not_zero), breaks = c(0.5, 1),
        col = rgb(1, 1, 1, alpha = 0.1), add = TRUE)

}, j = manipulate::slider(min = 1, max = ncol(mfit$beta$y1), step = 1,
                          label = "j in lambda(j)"))
```

## 4.4 Big data considerations

The computation of the least squares estimator

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y} \tag{4.24}$$

involves inverting the $(p+1)\times(p+1)$ matrix $\mathbf{X}^\top\mathbf{X}$, where $\mathbf{X}$ is an $n\times(p+1)$ matrix. The vector to be obtained, $\hat{\boldsymbol{\beta}}$, is of size $p+1$. However, computing it directly from (4.24) 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 $\hat{\boldsymbol{\beta}}$ with big data are required.

An approach for computing (4.24) in a memory-friendly way is to split the computation of $(\mathbf{X}^\top\mathbf{X})^{-1}$ and $\mathbf{X}^\top\mathbf{Y}$ by *blocks* that are storable 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 $\hat{\boldsymbol{\beta}}$ with $\hat{\boldsymbol{\beta}}_{-i}$, the vector of estimated coefficients *without* the $i$-th datum:

$$\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}_{-i} + (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{x}_i\left(Y_i - \mathbf{x}_i^\top\hat{\boldsymbol{\beta}}_{-i}\right). \tag{4.25}$$

In (4.25) above, $\mathbf{x}_i^\top$ is the $i$-th row of the design matrix $\mathbf{X}$. The expression follows from the Sherman–Morrison formula for an invertible matrix $\mathbf{A}$ and a vector $\mathbf{b}$,

$$(\mathbf{A}+\mathbf{b}\mathbf{b}^\top)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{b}\mathbf{b}^\top\mathbf{A}^{-1}}{1+\mathbf{b}^\top\mathbf{A}^{-1}\mathbf{b}},$$

and from the equalities

$$\begin{aligned}\mathbf{X}^\top\mathbf{X} &= \mathbf{X}_{-i}^\top\mathbf{X}_{-i} + \mathbf{x}_i\mathbf{x}_i^\top,\\ \mathbf{X}^\top\mathbf{Y} &= \mathbf{X}_{-i}^\top\mathbf{Y}_{-i} + \mathbf{x}_i Y_i^\top,\end{aligned}$$

where $\mathbf{X}_{-i}$ is the $(n-1)\times(p+1)$ matrix obtained by removing the $i$-th row of $\mathbf{X}$. In (4.25), using again the Sherman–Morrison formula, we can update $(\mathbf{X}^\top\mathbf{X})^{-1}$ easily from $\left(\mathbf{X}_{-i}^\top\mathbf{X}_{-i}\right)^{-1}$:

$$(\mathbf{X}^\top\mathbf{X})^{-1} = \left(\mathbf{X}_{-i}^\top\mathbf{X}_{-i}\right)^{-1} - \frac{\left(\mathbf{X}_{-i}^\top\mathbf{X}_{-i}\right)^{-1}\mathbf{x}_i\mathbf{x}_i^\top\left(\mathbf{X}_{-i}^\top\mathbf{X}_{-i}\right)^{-1}}{1+\mathbf{x}_i^\top\left(\mathbf{X}_{-i}^\top\mathbf{X}_{-i}\right)^{-1}\mathbf{x}_i}. \tag{4.26}$$

This has the advantage of not requiring to compute $\mathbf{X}^\top\mathbf{X}$ and then to invert it. Instead of that, we work directly with $\left(\mathbf{X}_{-i}^\top\mathbf{X}_{-i}\right)^{-1}$, which was *already* computed and has size $(p+1)\times(p+1)$.

This idea can be iterated and we can compute $\hat{\boldsymbol{\beta}}$ 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 $\hat{\boldsymbol{\beta}}_{\text{old}} \equiv \hat{\boldsymbol{\beta}}_{-i}$.
2. Add one of the remaining data points to get $\hat{\boldsymbol{\beta}}_{\text{new}} \equiv \hat{\boldsymbol{\beta}}$ from (4.25) and (4.26).
3. Set $\hat{\boldsymbol{\beta}}_{\text{new}} \leftarrow \hat{\boldsymbol{\beta}}_{\text{old}}$ and $\mathbf{X}_{\text{new}} \leftarrow \mathbf{X}_{\text{old}}$.
4. Repeat Steps 2–3 until there are no remaining data points left.
5. Return $\hat{\boldsymbol{\beta}} \leftarrow \hat{\boldsymbol{\beta}}_{\text{new}}$.

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 (4.24), `biglm::biglm` performs a QR decomposition[38] of $\mathbf{X}$ that is computed iteratively. Then, instead of computing (4.24), it solves the triangular system

$$\mathbf{R}\hat{\boldsymbol{\beta}} = \mathbf{Q}^\top\mathbf{Y}.$$

Let's see how `biglm::biglm` works in practice.

```
# 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 <- 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
# biglm::biglm(formula = resp ~ ., data = bigData1) # Does not work
# biglm::biglm(formula = y ~ x) # Does not work
# biglm::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::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(biglmMod), units = "KB")
## 13.1 Kb
print(object.size(lmMod), units = "MB")
## 381.5 Mb

# Summaries
s1 <- summary(biglmMod)
s2 <- summary(lmMod)
s1
```

[38] 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.

```
## Large data regression model: biglm::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$mat # Coefficients and their inferences
##                       Coef          (95%          CI)           SE            p
## (Intercept)  1.0021454430  0.9939053491  1.010385537 0.0041200470 0.000000e+00
## pred.1      -0.9732674585 -1.0132653005 -0.933269616 0.0199989210 0.000000e+00
## pred.2      -0.2866314070 -0.2911021089 -0.282160705 0.0022353509 0.000000e+00
## pred.3      -0.0534833941 -0.0554827653 -0.051484023 0.0009996856 0.000000e+00
## pred.4      -0.0040771777 -0.0060739907 -0.002080365 0.0009984065 4.432709e-05
## pred.5      -0.0002051218 -0.0022030377  0.001792794 0.0009989579 8.373098e-01
## pred.6       0.0002828388 -0.0017149118  0.002280589 0.0009988753 7.770563e-01
## pred.7       0.0026085425  0.0006093153  0.004607770 0.0009996136 9.066118e-03
## pred.8       0.0520743791  0.0500755376  0.054073221 0.0009994208 0.000000e+00
## pred.9       0.2840358104  0.2800374345  0.288034186 0.0019991879 0.000000e+00
## pred.10      0.9866850849  0.9667099026  1.006660267 0.0099875911 0.000000e+00
s1$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        pred.5        pred.6        pred.7
##  1.0021454430 -0.9732674585 -0.2866314070 -0.0534833941 -0.0040771777 -0.0002051218  0.0002828388  0.0026085425
##        pred.8        pred.9       pred.10
##  0.0520743791  0.2840358104  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::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 `MASS::stepAIC`, but with the more advanced `leaps` package. This is achieved by the `leaps::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[39] 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
reg <- leaps::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 decipher)
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
```

[39] 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.

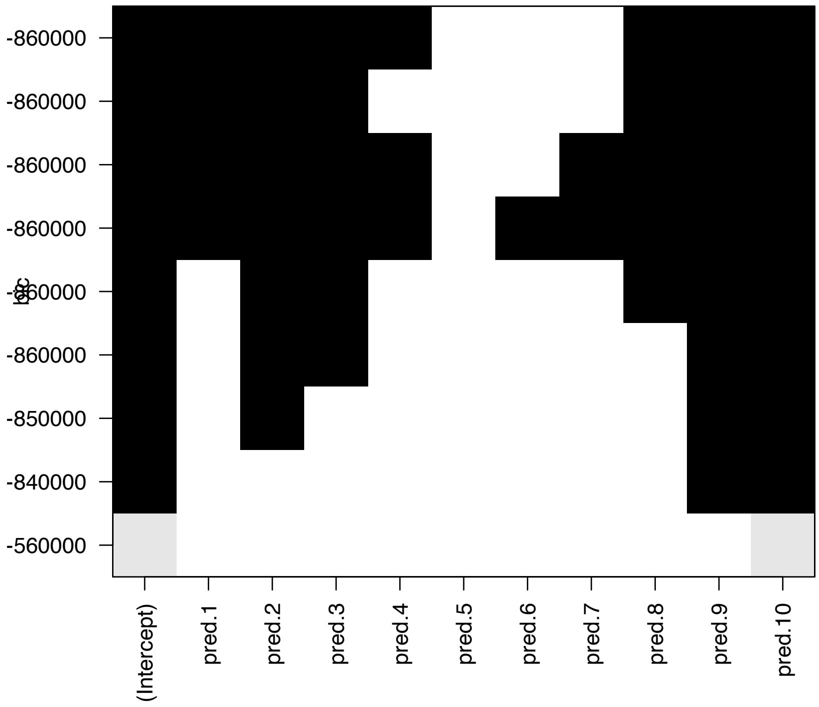


Figure 4.6: Best subsets for $p = 10$ predictors returned by `leaps::regsubsets`. The vertical axis indicates the sorting in terms of the BIC (the top positions contain the best models in terms of the BIC). White color indicates that the predictor is not included in the model and black that it is included. The $p$ models obtained with the best subsets of $1 \le r \le p$ out of $p$ predictors are displayed. Note that the vertical ordering does not necessarily coincide with $r = 1, \ldots, p$.

```
## 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 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 24444 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$which
##   (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5 pred.6 pred.7 pred.8 pred.9 pred.10
## 1        TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE    TRUE
## 2        TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE    TRUE
## 3        TRUE  FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE    TRUE
## 4        TRUE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE    TRUE
## 5        TRUE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE    TRUE
## 6        TRUE   TRUE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE    TRUE
## 7        TRUE   TRUE   TRUE   TRUE   TRUE  FALSE  FALSE  FALSE   TRUE   TRUE    TRUE
## 8        TRUE   TRUE   TRUE   TRUE   TRUE  FALSE  FALSE   TRUE   TRUE   TRUE    TRUE
## 9        TRUE   TRUE   TRUE   TRUE   TRUE  FALSE   TRUE   TRUE   TRUE   TRUE    TRUE
subs$bic
## [1] -558603.7 -837859.9 -854062.3 -856893.8 -859585.3 -861936.5 -861939.3 -861932.3 -861918.6
subs$which[which.min(subs$bic), ]
## (Intercept)      pred.1      pred.2      pred.3      pred.4      pred.5      pred.6      pred.7      pred.8      pred.9
##        TRUE        TRUE        TRUE        TRUE        TRUE       FALSE       FALSE       FALSE        TRUE        TRUE
##     pred.10
##        TRUE

# Show the display in Figure 4.6
subs$which[order(subs$bic), ]
##   (Intercept) pred.1 pred.2 pred.3 pred.4 pred.5 pred.6 pred.7 pred.8 pred.9 pred.10
## 7        TRUE   TRUE   TRUE   TRUE   TRUE  FALSE  FALSE  FALSE   TRUE   TRUE    TRUE
## 6        TRUE   TRUE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE    TRUE
## 8        TRUE   TRUE   TRUE   TRUE   TRUE  FALSE  FALSE   TRUE   TRUE   TRUE    TRUE
## 9        TRUE   TRUE   TRUE   TRUE   TRUE  FALSE   TRUE   TRUE   TRUE   TRUE    TRUE
## 5        TRUE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE    TRUE
## 4        TRUE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE    TRUE
## 3        TRUE  FALSE   TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE    TRUE
## 2        TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE   TRUE    TRUE
## 1        TRUE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE    TRUE

# It also works with ordinary linear models and it is much faster and
# informative than stepAIC
reg <- leaps::regsubsets(resp ~ ., data = bigData1, nvmax = p,
                         method = "backward")
subs <- summary(reg)
subs$bic
##  [1] -558603.7 -837859.9 -854062.3 -856893.8 -859585.3 -861936.5 -861939.3 -861932.3 -861918.6 -861904.8
subs$which[which.min(subs$bic), ]
## (Intercept)      pred.1      pred.2      pred.3      pred.4      pred.5      pred.6      pred.7      pred.8      pred.9
##        TRUE        TRUE        TRUE        TRUE        TRUE       FALSE       FALSE       FALSE        TRUE        TRUE
##     pred.10
##        TRUE

# Compare it with stepAIC
MASS::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       pred.8       pred.9      pred.10
##    1.002141    -0.973201    -0.286626    -0.053487    -0.004074     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 $8.2 \times 2 = 16.4$ GB more). The final `lm` object will thus be at the very least, of size 18.5 GB. Clearly, this is not a very memory-friendly procedure.

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::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::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")
## 7.8 Kb
```

The `summary` of a `biglm` object yields slightly different significances for the coefficients than those of `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.

# 5
# Generalized linear models

As we saw in Chapter 2, linear regression assumes that the response variable $Y$ is such that

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathcal{N}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \sigma^2)$$

and hence

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

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 $Y \mid (X_1 = x_1, \ldots, X_p = x_p)$, particularly with *discrete* responses, by modeling 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 the sample $\{(X_i, Y_i)\}_{i=1}^n$ in Figure 5.1.

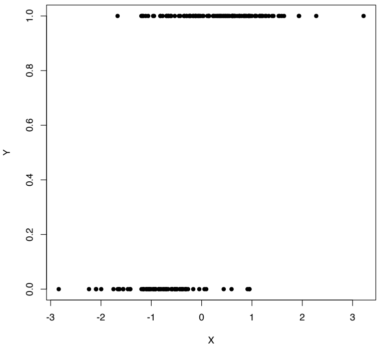


Figure 5.1: Scatterplot of a sample $\{(X_i, Y_i)\}_{i=1}^n$ sampled from a logistic regression.

## 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 NASA's 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,[1] among others) was created in order to investigate the causes of the disaster.

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 booster due to the unusually cold temperature ($-0.6$ Celsius degrees; 30.92 Fahrenheit degrees) during the launch. This failure produced a breach of burning gas through the solid rocket booster that compromised the whole shuttle structure, resulting in its disintegration due to the extreme aerodynamic forces.

Figure 5.2: Challenger launch and posterior explosion, as broadcasted live by NBC in 28/01/1986. Video also available here.

[1] His Appendix F in the Rogers Commission is particularly critical on the increasing sequence of risks that NASA took previously to the Challenger disaster. His closing sentence: "For a successful technology, reality must take precedence over public relations, for nature cannot be fooled".

The problem with O-rings was something known. The night before the launch, there was a three-hour teleconference between rocket engineers at Thiokol, the manufacturer company of the solid rocket boosters, and NASA. In the teleconference it was discussed the effect on the O-rings performance of the low temperature forecasted for the launch, and eventually a launch decision was attained.[2] Figure 5.3a influenced the data analysis conclusion that sustained the launch decision:

> "Temperature data [is] not conclusive on predicting primary O-ring blowby."

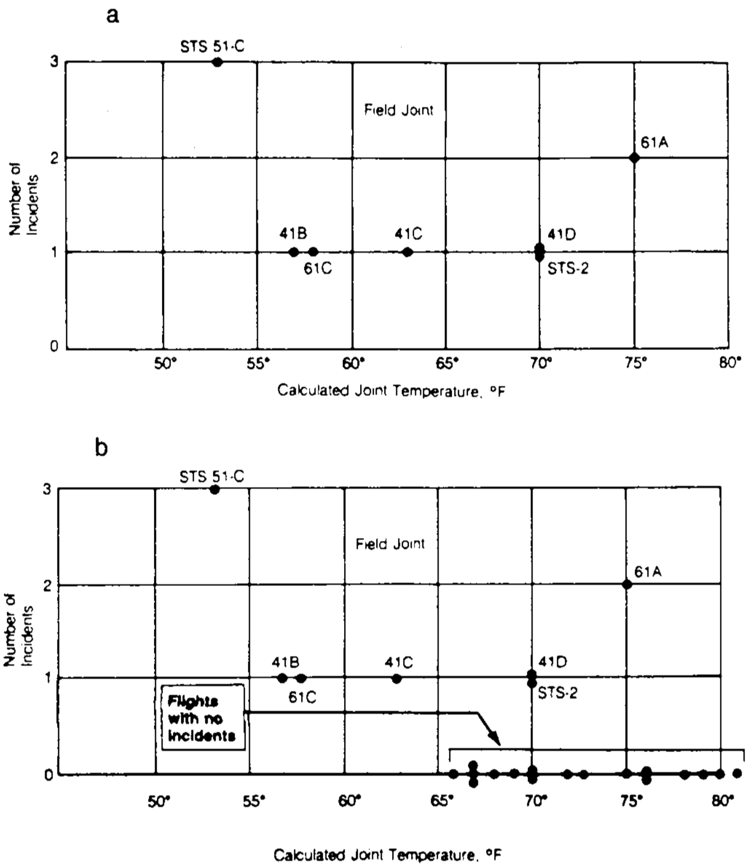


Figure 5.3: Number of incidents in the O-rings (filed joints) versus temperatures. Panel *a* includes *only flights with incidents*. Panel *b* contains all flights (with and *without* incidents).

The Rogers Commission noted a major flaw in Figure 5.3a, the one presented by the solid rocket booster engineers at the teleconference: the **flights with zero incidents were excluded** from the plot by the engineers because *it was felt* that **these flights did not contribute any information about the temperature effect** (Figure 5.3b). The Rogers Commission therefore concluded:

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

The purpose of this case study, inspired by Siddhartha et al. (1989), is to quantify what was the influence of the temperature

[2] Episode S1:E3 "A Major Malfunction" in Netflix's series "Challenger: The Final Flight" reproduces the events during this teleconference between Thiokol (Promontory, Utah) and NASA (Cape Canaveral, Florida), with interviews to the relevant actors. The coverage of the teleconference spans from the time marks 17:40 to 23:55. Relevant highlights are: the engineers are told to give an assessment of concerns regarding launching in cold weather (17:40–18:56); the engineers present the report and their *unquantified* concerns about higher risk with cold temperatures (20:38–21:20); Larry Mulloy, NASA manager in 1986, says "The data [the engineers presented] did not support the recommendation that the engineers were making. The data was totally inconclusive." (21:47–21:58); the vice president of engineers of Thiokol, Robert Lund, gives the specific recommendation to not launch below 53 degrees Fahrenheit (11.67 degrees Celsius) and Larry Mulloy replies (22:33–22:53); Robert Lund, pressured, changes his assessment and agrees with Larry Mulloy's: data is inconclusive (24:12–25:33). Recall that the data presented during the teleconference is the one summarized in Figure 5.3a.

on 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?*
- Q4. *What was the predicted maximum probability of an incident in an O-ring if the launch were postponed until the temperature was above* 11.67 *degrees Celsius, as the vice president of engineers of Thiokol recommended?*

To try to answer these questions, we analyze the `challenger` dataset (download), partially collected in Table 5.1. The dataset contains information regarding the state of the solid rocket boosters after launch[3] for 23 flights prior to the Challenger launch. 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. For the analysis, we focus on the O-rings of the field joint as those were the most determinant for the accident.
- `nfail.field`, `nfail.nozzle`: number of incidents with the O-rings in the field joints and in the nozzles.
- `temp`: temperature in the day of launch, measured in Celsius degrees.
- `pres.field`, `pres.nozzle`: leak-check pressure tests of the O-rings. These tests assured that the rings would seal the joint.

Table 5.1: The `challenger` dataset.

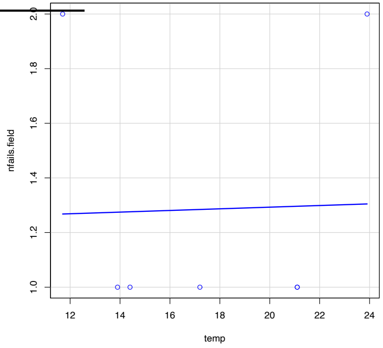
| flight | date | nfails.field | nfails.nozzle | fail.field | fail.nozzle | temp |
|---|---|---|---|---|---|---|
| 1 | 12/04/81 | 0 | 0 | 0 | 0 | 18.9 |
| 2 | 12/11/81 | 1 | 0 | 1 | 0 | 21.1 |
| 3 | 22/03/82 | 0 | 0 | 0 | 0 | 20.6 |
| 5 | 11/11/82 | 0 | 0 | 0 | 0 | 20.0 |
| 6 | 04/04/83 | 0 | 2 | 0 | 1 | 19.4 |
| 7 | 18/06/83 | 0 | 0 | 0 | 0 | 22.2 |
| 8 | 30/08/83 | 0 | 0 | 0 | 0 | 22.8 |
| 9 | 28/11/83 | 0 | 0 | 0 | 0 | 21.1 |
| 41-B | 03/02/84 | 1 | 1 | 1 | 1 | 13.9 |
| 41-C | 06/04/84 | 1 | 1 | 1 | 1 | 17.2 |
| 41-D | 30/08/84 | 1 | 1 | 1 | 1 | 21.1 |
| 41-G | 05/10/84 | 0 | 0 | 0 | 0 | 25.6 |
| 51-A | 08/11/84 | 0 | 0 | 0 | 0 | 19.4 |
| 51-C | 24/01/85 | 2 | 2 | 1 | 1 | 11.7 |

[3] After the shuttle exits the atmosphere, the solid rocket boosters separate and descend to land using a parachute. After recovery, they are carefully analyzed.

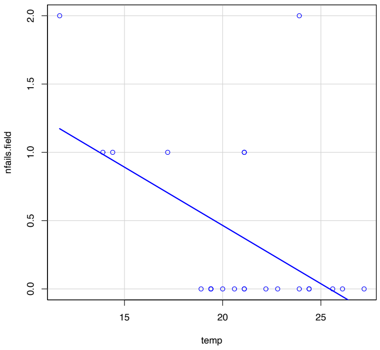
| | | | | | | |
|---|---|---|---|---|---|---|
| 51-D | 12/04/85 | 0 | 2 | 0 | 1 | 19.4 |
| 51-B | 29/04/85 | 0 | 2 | 0 | 1 | 23.9 |
| 51-G | 17/06/85 | 0 | 2 | 0 | 1 | 21.1 |
| 51-F | 29/07/85 | 0 | 0 | 0 | 0 | 27.2 |
| 51-I | 27/08/85 | 0 | 0 | 0 | 0 | 24.4 |
| 51-J | 03/10/85 | 0 | 0 | 0 | 0 | 26.1 |
| 61-A | 30/10/85 | 2 | 0 | 1 | 0 | 23.9 |
| 61-B | 26/11/85 | 0 | 2 | 0 | 1 | 24.4 |
| 61-C | 12/01/86 | 1 | 2 | 1 | 1 | 14.4 |

Let's begin the analysis by replicating Figures 5.3a and 5.3b, and by checking that linear regression is not the right tool for addressing Q1–Q4.

```
# Read data
challenger <- read.table(file = "challenger.txt", header = TRUE, sep = "\t")

# Figures 5.3a and 5.3b
car::scatterplot(nfails.field ~ temp, smooth = FALSE, boxplots = FALSE,
                 data = challenger, subset = nfails.field > 0)
```

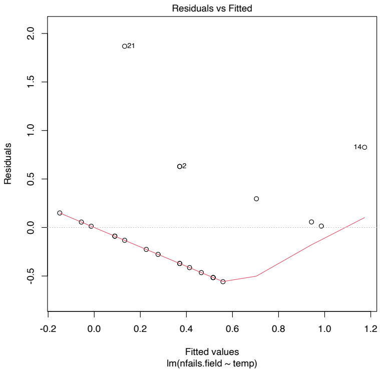


```
car::scatterplot(nfails.field ~ temp, smooth = FALSE, boxplots = FALSE,
                 data = challenger)
```

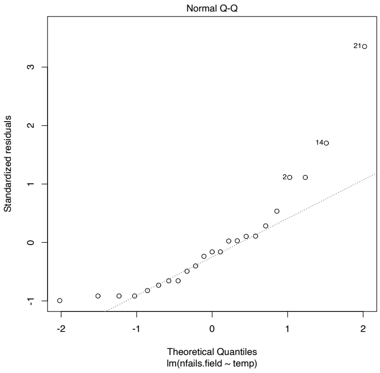


There is a fundamental problem in using linear regression for this data: the response is not continuous. As a consequence, there is no linearity and the errors around the mean are not normal (indeed, they are strongly non-normal). Let's check this with the corresponding diagnostic plots:

```
# Fit linear model, and run linearity and normality diagnostics
mod <- lm(nfails.field ~ temp, data = challenger)
plot(mod, 1)
```



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



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

1. The **trend for launches with incidents is flat**, hence suggesting there is **no correlation** on the temperature (Figure 5.3a). This was one of the arguments behind NASA's decision to launch the rocket at a temperature of $-0.6$ Celsius degrees.
2. However, as Figure 5.3b reveals, the **trend for *all* launches** indicates a **clear negative dependence** between temperature and number of incidents! 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.

Along this chapter we will see the required tools for answering precisely Q1–Q4.

## 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 $X_1, \ldots, X_p$ and $Y$ by assuming a linear relation in the conditional expectation:

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p. \tag{5.1}$$

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:

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathcal{N}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \sigma^2).$$

Recall that a necessary condition for the linear model to hold is that $Y$ is 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 different 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 to as a *Bernoulli variable*:[4] $Y \sim \mathrm{Ber}(p)$, $0 \leq p \leq 1$ [5], if

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

or, equivalently, if

$$\mathbb{P}[Y = y] = p^y (1 - p)^{1-y}, \quad y = 0, 1. \tag{5.2}$$

Recall that a Bernoulli variable is completely determined by the probability $p$. Therefore, so are its mean and variance:

$$\mathbb{E}[Y] = \mathbb{P}[Y = 1] = p \quad \text{and} \quad \mathbb{V}\mathrm{ar}[Y] = p(1 - p).$$

Assume then that $Y$ is a Bernoulli variable and that $X_1, \ldots, X_p$ are predictors associated with $Y$. The purpose in logistic regression is to model

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \mathbb{P}[Y = 1 \mid X_1 = x_1, \ldots, X_p = x_p], \tag{5.3}$$

that is, to model 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

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p =: \eta.$$

[4] Recall that a *binomial variable with size $n$ and probability $p$*, $\mathrm{B}(n, p)$, is obtained by summing $n$ independent $\mathrm{Ber}(p)$, so $\mathrm{Ber}(p)$ is the same distribution as $\mathrm{B}(1, p)$.

[5] Do not confuse this $p$ with the number of predictors in the model, represented by $p$. The context should make unambiguous the use of $p$.

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 $\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p]$ 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 $\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p]$. There are several link functions $g$ with associated $g^{-1}$. Each link generates a different model:

- **Uniform link**. Based on the truncation $g^{-1}(\eta) = \eta \mathbb{1}_{\{0<\eta<1\}} + \mathbb{1}_{\{\eta \geq 1\}}$.
- **Probit link**. Based on the *normal* cdf, this is, $g^{-1}(\eta) = \Phi(\eta)$.
- **Logit link**. Based on the **logistic cdf**:[6]

$$g^{-1}(\eta) = \mathrm{logistic}(\eta) := \frac{e^\eta}{1 + e^\eta} = \frac{1}{1 + e^{-\eta}}.$$

The logistic transformation is the most employed due to its tractability, interpretability, and smoothness.[7] Its inverse, $g : [0, 1] \longrightarrow \mathbb{R}$, is known as the **logit function**:

$$\mathrm{logit}(p) := \mathrm{logistic}^{-1}(p) = \log\left(\frac{p}{1 - p}\right).$$

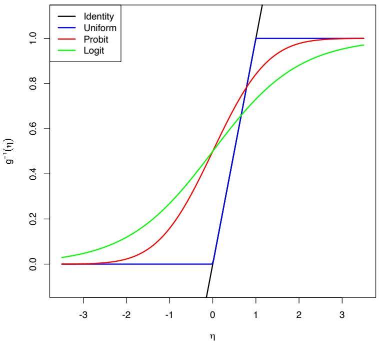


Figure 5.4: Transformations $g^{-1}$ associated with different link functions. The transformations $g^{-1}$ map the response of a linear regression $\eta = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$ to $[0, 1]$.

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

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \mathrm{logistic}(\eta) = \frac{1}{1 + e^{-\eta}}, \tag{5.4}$$

or as

$$\mathrm{logit}(\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p]) = \eta \tag{5.5}$$

where recall that

$$\eta = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

There is a clear interpretation of the role of the linear predictor $\eta$ in (5.4) when we come back to (5.3):

- If $\eta = 0$, then $\mathbb{P}[Y = 1 \mid X_1 = x_1, \ldots, X_p = x_p] = \frac{1}{2}$ ($Y = 1$ and $Y = 0$ are equally likely).
- If $\eta < 0$, then $\mathbb{P}[Y = 1 \mid X_1 = x_1, \ldots, X_p = x_p] < \frac{1}{2}$ ($Y = 1$ is less likely).
- If $\eta > 0$, then $\mathbb{P}[Y = 1 \mid X_1 = x_1, \ldots, X_p = x_p] > \frac{1}{2}$ ($Y = 1$ is 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

$$\mathrm{odds}(Y) := \frac{p}{1 - p} = \frac{\mathbb{P}[Y = 1]}{\mathbb{P}[Y = 0]}. \tag{5.6}$$

[6] The fact that the logistic function is a cdf allows remembering that the logistic is to be applied to map $\mathbb{R}$ into $[0, 1]$, as opposed to the logit function.

[7] And also, as we will see later, because it is the *canonical link function*.

[8] Consequently, the name "odds" used in this context is singular, as it refers to a single ratio.

[9] Recall that (traditionally) the result of a bet is binary: one either wins or loses it.

The odds is thus the *ratio between the probability of success and the probability of failure*.[8] It is extensively used in betting.[9] due to its better interpretability[10] Conversely, if the odds of $Y$ is given, we can easily know what is the probability of success $p$, using the inverse of (5.6):[11]

$$p = \mathbb{P}[Y = 1] = \frac{\text{odds}(Y)}{1 + \text{odds}(Y)}.$$

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) in terms of the odds (5.6)[12] so we get:

$$\text{odds}(Y \mid X_1 = x_1, \ldots, X_p = x_p) = e^{\eta} = e^{\beta_0} e^{\beta_1 x_1} \ldots e^{\beta_p x_p}. \tag{5.7}$$

Alternatively, taking logarithms, we have the *log-odds* (or logit)

$$\log(\text{odds}(Y \mid X_1 = x_1, \ldots, X_p = x_p)) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p. \tag{5.8}$$

The conditional log-odds (5.8) 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 $X_1 = \cdots = X_p = 0$.
- $\beta_j$, $1 \leq j \leq p$: is the **additive** increment of the **log-odds** for an increment of one unit in $X_j = x_j$, provided that the remaining variables $X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_p$ *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 (5.7), is:

- $e^{\beta_0}$: is the odds when $X_1 = \cdots = X_p = 0$.
- $e^{\beta_j}$, $1 \leq j \leq p$: is the **multiplicative** increment of the **odds** for an increment of one unit in $X_j = x_j$, provided that the remaining variables $X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_p$ *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 (5.7). 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 $\mathbb{P}[Y = 1 \mid X_1 = x_1, \ldots, X_p = x_p]$.

[10] For example, if a horse $Y$ has 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").

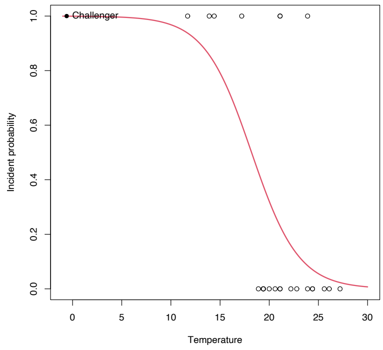
[11] For the previous example: if the odds of the horse was 5, then the probability of winning would be $p = 5/6$.

[12] To do so, apply (5.6) to (5.4) and use (5.3).

### 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.

```
# Logistic regression: computed with glm and family = "binomial"
nasa <- glm(fail.field ~ temp, family = "binomial", data = challenger)
nasa
##
## Call:  glm(formula = fail.field ~ temp, family = "binomial", data = challenger)
##
## Coefficients:
## (Intercept)         temp
##      7.5837      -0.4166
##
## Degrees of Freedom: 22 Total (i.e. Null);  21 Residual
## Null Deviance:       28.27
## Residual Deviance: 20.33     AIC: 24.33

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

# Draw the fitted logistic curve
x <- seq(-1, 30, l = 200)
eta <- nasa$coefficients[1] + nasa$coefficients[2] * x
y <- 1 / (1 + exp(-eta))
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 (estimated) probability of having an incident ($Y = 1$) is 1965.974 times larger than the probability of not having an incident ($Y = 0$). Or, 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 on the temperature multiplies the fitted odds by a factor of $0.659 \approx \frac{2}{3}$, hence reducing it.

However, for the moment we cannot say whether these findings are significant or are just an artifact of the randomness of the data, since we do not have information on the variability of the estimates of $\boldsymbol{\beta}$. We will need inference for that.

### Estimation by maximum likelihood

The estimation of $\boldsymbol{\beta}$ from a sample $\{(\mathbf{x}_i, Y_i)\}_{i=1}^n$ [13] is done by *Maximum Likelihood Estimation* (MLE). As it can be seen in Appendix B, in the linear model, under the assumptions mentioned in Section 2.3, MLE is equivalent to least squares estimation. In the logistic model, we assume that[14]

$$Y_i \mid (X_1 = x_{i1}, \ldots, X_p = x_{ip}) \sim \mathrm{Ber}(\mathrm{logistic}(\eta_i)), \quad i = 1, \ldots, n,$$

where $\eta_i := \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$. Denoting $p_i(\boldsymbol{\beta}) := \mathrm{logistic}(\eta_i)$, the log-likelihood of $\boldsymbol{\beta}$ is

$$\begin{aligned}\ell(\boldsymbol{\beta}) &= \log\left(\prod_{i=1}^n p_i(\boldsymbol{\beta})^{Y_i}(1 - p_i(\boldsymbol{\beta}))^{1-Y_i}\right)\\ &= \sum_{i=1}^n \left[Y_i \log(p_i(\boldsymbol{\beta})) + (1 - Y_i)\log(1 - p_i(\boldsymbol{\beta}))\right]. \qquad (5.9)\end{aligned}$$

The ML estimate of $\boldsymbol{\beta}$ is

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

Unfortunately, due to the nonlinearity of (5.9), there is no explicit expression for $\hat{\boldsymbol{\beta}}$ and it has to be obtained numerically by means of an iterative procedure. We will see that 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.

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 available here.

Figure 5.2.1 shows how the log-likelihood changes with respect to the values for $(\beta_0, \beta_1)$ in three data patterns. The data of the illustration has been generated with the next chunk of 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 a binomial variable (since it is a Bernoulli). Let’s check that indeed

[13] As in the linear model, we assume the randomness comes from the error present in $Y$ once $\mathbf{X}$ is given, not from the $\mathbf{X}$, and we therefore denote $\mathbf{x}_i$ to the $i$-th observation of $\mathbf{X}$.

[14] Section 5.7 discusses in detail the assumptions of generalized linear models.

the coefficients given by glm are the ones that maximize the likelihood given in the animation of Figure 5.2.1. We do so for y1 ~ x.

```
# 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 log-likelihood surface
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]))
  }
}
filled.contour(beta0, beta1, -L, color.palette = viridis::viridis,
               xlab = expression(beta[0]), ylab = expression(beta[1]),
               plot.axes = {
                 axis(1); axis(2)
                 points(mod$coefficients[1], mod$coefficients[2],
                        col = 2, pch = 16)
                 points(opt$par[1], opt$par[2], col = 4)
               })


# The plot.axes argument is a hack to add graphical information within the
# coordinates of the main panel (behind filled.contour there is a layout()...)
```

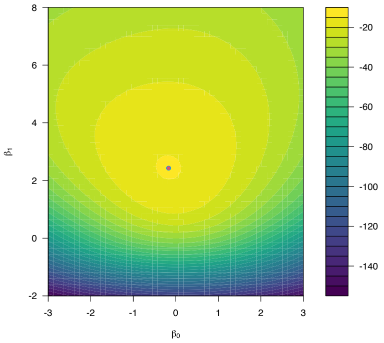


Figure 5.5: Log-likelihood surface $\ell(\beta_0, \beta_1)$ and its global maximum $(\hat{\beta}_0, \hat{\beta}_1)$.

**Exercise 5.1.** For the regressions y2 ~ x and y3 ~ x, do the following:

a. Check that the true $\boldsymbol{\beta}$ is close to maximizing the likelihood computed in Figure 5.2.1.
b. Plot the fitted logistic curve and compare it with the one in Figure 5.2.1.

The extension of the logistic model to the case of a *categorical response with more than two levels* is sketched in Appendix C.

### 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 extends 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*[15] to be modeled by a linear combination of the predictors:

$$g\left(\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p]\right) = \eta$$

or, equivalently,

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = g^{-1}(\eta),$$

where

$$\eta := \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

is the *linear predictor*.

The second ingredient of generalized linear models is a distribution for $Y \mid (X_1, \ldots, X_p)$, just as the linear model assumes normality or the logistic model assumes a Bernoulli random variable. Thus, we have **two linked 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 $Y \mid (X_1, \ldots, X_p)$ may be different from the normal.

Generalized linear models are intimately related with the **exponential family**,[16] [17] which is the family of distributions with pdf expressible as

$$f(y;\theta,\phi) = \exp\left\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\right\}, \tag{5.10}$$

where $a(\cdot)$, $b(\cdot)$, and $c(\cdot,\cdot)$ are specific functions. If $Y$ has the pdf (5.10), then we write $Y \sim \mathrm{E}(\theta, \phi, a, b, c)$. 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).

[15] Notice that this approach is *very* different from directly transforming the response as $g(Y)$, as outlined in Section 3.5.1. Indeed, in generalized linear models one transforms $\mathbb{E}[Y \mid X_1, \ldots, X_p]$, not $Y$. Of course, $g(\mathbb{E}[Y \mid X_1, \ldots, X_p]) \neq \mathbb{E}[g(Y) \mid X_1, \ldots, X_p]$.

[16] Not to be confused with the exponential *distribution* $\mathrm{Exp}(\lambda)$, which is a *member* of the exponential family.

[17] This is the so-called *canonical form* of the exponential family. Generalizations of the family are possible, though we do not consider them.

Distributions from the exponential family have some nice properties. Importantly, if $Y \sim \mathrm{E}(\theta, \phi, a, b, c)$, then

$$\mu := \mathbb{E}[Y] = b'(\theta), \quad \sigma^2 := \mathbb{V}\mathrm{ar}[Y] = b''(\theta)a(\phi). \tag{5.11}$$

The **canonical link function** is the function $g$ that **transforms** $\mu = b'(\theta)$ **into the canonical parameter** $\theta$. For $\mathrm{E}(\theta, \phi, a, b, c)$, this happens if

$$\theta = g(\mu) \tag{5.12}$$

or, more explicitly due to (5.11), if

$$g(\mu) = (b')^{-1}(\mu). \tag{5.13}$$

In the case of canonical link function, the one-line summary of the generalized linear model is (independence is implicit)

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{E}(\eta, \phi, a, b, c). \tag{5.14}$$

Expression (5.14) gives insight on what a generalized linear model does:

1. Select a member of the exponential family in (5.10) for modeling $Y$.
2. The canonical link function $g$ is $g(\mu) = (b')^{-1}(\mu)$. In this case, $\theta = g(\mu)$.
3. The generalized linear model associated with the member of the exponential family and $g$ models the *conditional* $\theta$, given $X_1, \ldots, X_n$, by means of the linear predictor $\eta$. This is equivalent to modeling the conditional $\mu$ by means of $g^{-1}(\eta)$.

The **linear model** arises as a particular case of (5.14) with

$$a(\phi) = \phi, \quad b(\theta) = \frac{\theta^2}{2}, \quad c(y, \phi) = -\frac{1}{2}\left\{\frac{y^2}{\phi} + \log(2\pi\phi)\right\},$$

and scale parameter $\phi = \sigma^2$. In this case, $\mu = \theta$ and the canonical link function $g$ is the identity.

**Exercise 5.2.** Show that the normal, Bernoulli, exponential, and Poisson distributions are members of the exponential family. For that, express their pdfs in terms of (5.10) and identify who is $\theta$ and $\phi$.

**Exercise 5.3.** Show that the binomial and gamma (which includes exponential and chi-squared) distributions are members of the exponential family. For that, express their pdfs in terms of (5.10) and identify who is $\theta$ and $\phi$.

[18] The pdf of a $\Gamma(a, \nu)$ is $f(x; a, p) = \frac{a^p}{\Gamma(\nu)} x^{\nu-1} e^{-ax}$, for $a, \nu > 0$ and $x \in (0, \infty)$ (the pdf is zero otherwise). The expectation is $\mathbb{E}[\Gamma(a, \nu)] = \frac{\nu}{a}$.

[19] If the argument $-\eta\nu$ is not positive, then the probability assigned by $\Gamma(-\eta\nu, \nu)$ is zero. This delicate case may complicate the estimation of the model. Valid starting values for $\boldsymbol{\beta}$ are required.

The following table lists some useful generalized linear models. Recall that the linear and logistic models of Sections 2.2.3 and 5.2.1 are obtained from the first and second rows, respectively.

| Support of $Y$ | Generating distribution | Link $g(\mu)$ | Expectation $g^{-1}(\eta)$ | Scale $\phi$ | Distribution of $Y \mid \mathbf{X} = \mathbf{x}$ |
|---|---|---|---|---|---|
| $\mathbb{R}$ | $\mathcal{N}(\mu, \sigma^2)$ | $\mu$ | $\eta$ | $\sigma^2$ | $\mathcal{N}(\eta, \sigma^2)$ |
| $\{0, 1\}$ | $\mathrm{Ber}(p)$ | $\mathrm{logit}(\mu)$ | $\mathrm{logistic}(\eta)$ | $1$ | $\mathrm{Ber}\left(\mathrm{logistic}(\eta)\right)$ |
| $\{0, \ldots, N\}$ | $\mathrm{B}(N, p)$ | $\log\left(\frac{\mu}{N-\mu}\right)$ | $N \cdot \mathrm{logistic}(\eta)$ | $1$ | $\mathrm{B}(N, \mathrm{logistic}(\eta))$ |
| $\{0, 1, \ldots\}$ | $\mathrm{Pois}(\lambda)$ | $\log(\mu)$ | $e^\eta$ | $1$ | $\mathrm{Pois}(e^\eta)$ |
| $(0, \infty)$ | $\Gamma(a, \nu)$ [18] | $-\frac{1}{\mu}$ | $-\frac{1}{\eta}$ | $\frac{1}{\nu}$ | $\Gamma(-\eta\nu, \nu)$ [19] |

**Exercise 5.4.** Obtain the canonical link function for the exponential distribution $\mathrm{Exp}(\lambda)$. What is the scale parameter? What is the distribution of $Y \mid (X_1 = x_1, \ldots, X_p = x_p)$ in such model?

### Poisson regression

Poisson regression is usually employed for modeling **count data** that arises from the recording of the frequencies of a certain phenomenon. It considers that

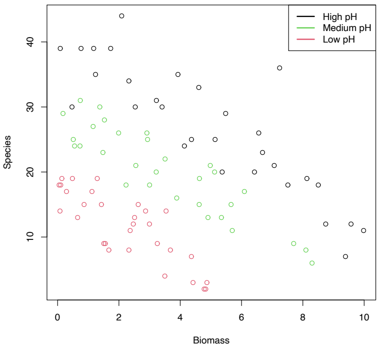
$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{Pois}(e^\eta),$$

this is,

$$\begin{aligned}\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] &= \lambda(Y \mid X_1 = x_1, \ldots, X_p = x_p) \\ &= e^{\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p}. \end{aligned} \tag{5.15}$$

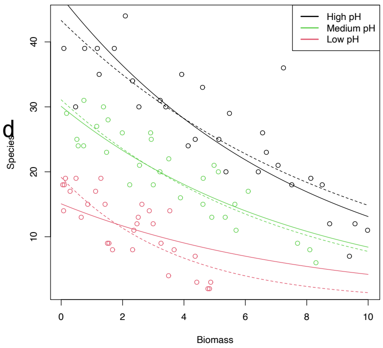
Let's see how to apply a Poisson regression. For that aim we consider the `species` (download) dataset. The goal is to analyze whether the `Biomass` and the `pH` (a factor) of the terrain are influential on the number of `Species`. Incidentally, it will serve to illustrate that the use of factors within `glm` is completely analogous to what we did with `lm`.

```
# Read data
species <- read.table("species.txt", header = TRUE)
species$pH <- as.factor(species$pH)

# Plot data
plot(Species ~ Biomass, data = species, col = as.numeric(pH))
legend("topright", legend = c("High pH", "Medium pH", "Low pH"),
       col = c(1, 3, 2), lwd = 2) # colors according to as.numeric(pH)
```



```
# 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 Biomass:pHmed 
##    43.2987424     0.8984091     0.4423865     0.7178730     0.8563910     0.9686112 
# - If pH decreases to med (low), then the effect of the biomass in the number
#   of species decreases by a factor of 0.9686 (0.8564). The higher the pH, the
#   stronger the effect of the Biomass in Species
# Draw fits
plot(Species ~ Biomass, data = species, col = as.numeric(pH))
legend("topright", legend = c("High pH", "Medium pH", "Low pH"),
       col = c(1, 3, 2), lwd = 2) # colors according to as.numeric(pH)

# Without interactions
bio <- seq(0, 10, l = 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, l = 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)
```



**Exercise 5.5.** For the `challenger` dataset, do the following:

a. Do a Poisson regression of the total number of incidents, `nfails.field + nfails.nozzle`, on `temp`.
b. Plot the data and the fitted Poisson regression curve.
c. Predict the expected number of incidents at temperatures $-0.6$ and 11.67.

### Binomial regression

Binomial regression is an extension of logistic regression that allows us to model discrete responses $Y$ in $\{0, 1, \ldots, N\}$, where $N$ is fixed. In its most vanilla version, it considers the model

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{B}(N, \mathrm{logistic}(\eta)), \tag{5.16}$$

this is,

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = N \cdot \mathrm{logistic}(\eta). \tag{5.17}$$

Comparing (5.17) with (5.4), it is clear that the logistic regression is a particular case with $N = 1$. The interpretation of the coefficients is therefore clear from the interpretation of (5.4), given that logistic$(\eta)$ models the probability of success of each of the $N$ experiments of the binomial $\mathrm{B}(N, \mathrm{logistic}(\eta))$.

The extra flexibility that binomial regression has offers interesting applications. First, we can use (5.16) as an approach to model *proportions*[20] $Y/N \in [0, 1]$. In this case, (5.17) becomes[21]

$$\mathbb{E}[Y/N \mid X_1 = x_1, \ldots, X_p = x_p] = \mathrm{logistic}(\eta).$$

Second, we can **let $N$ be dependent on the predictors** to accommodate group structures, perhaps the most common usage of binomial regression:

$$Y \mid \mathbf{X} = \mathbf{x} \sim \mathrm{B}(N_{\mathbf{x}}, \mathrm{logistic}(\eta)), \tag{5.18}$$

[20] Note that this situation is very different from logistic regression, for which we either have observations with the values 0 or 1. In binomial regression, we can naturally have proportions.

[21] Clearly, $\mathbb{E}[\mathrm{B}(N, p)/N] = p$ because $\mathbb{E}[\mathrm{B}(N, p)] = Np$.

where the size of the binomial distribution, $N_{\mathbf{x}}$, depends on the values of the predictors. For example, imagine that the predictors are two quantitative variables and two dummy variables encoding three categories. Then $\mathbf{X} = (X_1, X_2, D_1, D_2)^\top$ and $\mathbf{x} = (x_1, x_2, d_1, d_2)^\top$. In this case, $N_{\mathbf{x}}$ could for example take the form

$$N_{\mathbf{x}} = \begin{cases} 30, & d_1 = 0, d_2 = 0, \\ 25, & d_1 = 1, d_2 = 0, \\ 50, & d_1 = 0, d_2 = 1, \end{cases}$$

that is, we have a *different* number of experiments on each category, and we want to model the number (or, equivalently, the proportion) of successes for each one, also taking into account the effects of other qualitative variables. This is a very common situation in practice, when one encounters the sample version of (5.18):

$$Y_i \mid \mathbf{X}_i = \mathbf{x}_i \sim \mathrm{B}(N_i, \mathrm{logistic}(\eta_i)), \quad i = 1, \ldots, n. \tag{5.19}$$

Let's see an example of binomial regression that illustrates the particular usage of `glm()` in this case. The example is a data application from Wood (2006) featuring different binomial sizes. It employs the `heart` (download) dataset. The goal is to investigate whether the level of creatinine kinase level present in the blood, `ck`, is a good diagnostic for determining if a patient is likely to have a future heart attack. The number of patients that did not have a heart attack (`ok`) and that had a heart attack (`ha`) was established after `ck` was measured. In total, there are 226 patients that have been aggregated into $n = 12$ [22] categories of different sizes that have been created according to the average level of `ck`. Table 5.3 shows the data.

```
# Read data
heart <- read.table("heart.txt", header = TRUE)

# Sizes for each observation (Ni's)
heart$Ni <- heart$ok + heart$ha

# Proportions of patients with heart attacks
heart$prop <- heart$ha / (heart$ha + heart$ok)
```

Table 5.3: The `heart` dataset with $N_i$ (`Ni`) and $Y_i/N_i$ (`prop`) added.

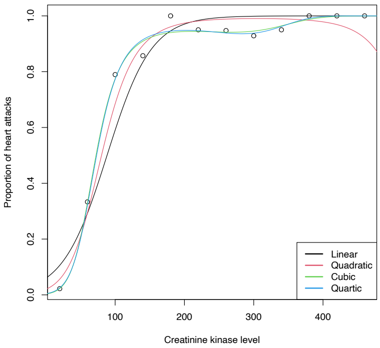
| ck | ha | ok | Ni | prop |
|---:|---:|---:|---:|---:|
| 20 | 2 | 88 | 90 | 0.022 |
| 60 | 13 | 26 | 39 | 0.333 |
| 100 | 30 | 8 | 38 | 0.789 |
| 140 | 30 | 5 | 35 | 0.857 |
| 180 | 21 | 0 | 21 | 1.000 |
| 220 | 19 | 1 | 20 | 0.950 |
| 260 | 18 | 1 | 19 | 0.947 |

[22] The sample size here is $n = 12$, not 226. There are $N_1, \ldots, N_{12}$ binomial sizes corresponding to each observation, and $\sum_{i=1}^n N_i = 226$.

| | | | | |
|---|---|---|---|---|
| 300 | 13 | 1 | 14 | 0.929 |
| 340 | 19 | 1 | 20 | 0.950 |
| 380 | 15 | 0 | 15 | 1.000 |
| 420 | 7 | 0 | 7 | 1.000 |
| 460 | 8 | 0 | 8 | 1.000 |

```
# Plot of proportions versus ck: twelve observations, each requiring
# Ni patients to determine the proportion
plot(heart$ck, heart$prop, xlab = "Creatinine kinase level",
     ylab = "Proportion of heart attacks")

# Fit binomial regression: recall the cbind() to pass the number of successes
# and failures
heart1 <- glm(cbind(ha, ok) ~ ck, family = binomial, data = heart)
summary(heart1)
##
## Call:
## glm(formula = cbind(ha, ok) ~ ck, family = binomial, data = heart)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.08184  -1.93008   0.01652   0.41772   2.60362
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.758358   0.336696  -8.192 2.56e-16 ***
## ck           0.031244   0.003619   8.633  < 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: 271.712  on 11  degrees of freedom
## Residual deviance:  36.929  on 10  degrees of freedom
## AIC: 62.334
##
## Number of Fisher Scoring iterations: 6

# Alternatively: put proportions as responses, but then it is required to
# inform about the binomial size of each observation
heart1 <- glm(prop ~ ck, family = binomial, data = heart, weights = Ni)
summary(heart1)
##
## Call:
## glm(formula = prop ~ ck, family = binomial, data = heart, weights = Ni)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.08184  -1.93008   0.01652   0.41772   2.60362
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.758358   0.336696  -8.192 2.56e-16 ***
## ck           0.031244   0.003619   8.633  < 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: 271.712  on 11  degrees of freedom
## Residual deviance:  36.929  on 10  degrees of freedom
## AIC: 62.334
##
## Number of Fisher Scoring iterations: 6

# Add fitted line
```

```
ck <- 0:500
newdata <- data.frame(ck = ck)
logistic <- function(eta) 1 / (1 + exp(-eta))
lines(ck, logistic(cbind(1, ck) %*% heart1$coefficients))

# It seems that a polynomial fit could better capture the "wiggly" pattern
# of the data
heart2 <- glm(prop ~ poly(ck, 2, raw = TRUE), family = binomial, data = heart,
              weights = Ni)
heart3 <- glm(prop ~ poly(ck, 3, raw = TRUE), family = binomial, data = heart,
              weights = Ni)
heart4 <- glm(prop ~ poly(ck, 4, raw = TRUE), family = binomial, data = heart,
              weights = Ni)

# Best fit given by heart3
BIC(heart1, heart2, heart3, heart4)
##        df      BIC
## heart1  2 63.30371
## heart2  3 44.27018
## heart3  4 35.59736
## heart4  5 37.96360
summary(heart3)
##
## Call:
## glm(formula = prop ~ poly(ck, 3, raw = TRUE), family = binomial,
##     data = heart, weights = Ni)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.99572  -0.08966   0.07468   0.17815   1.61096
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -5.786e+00  9.268e-01  -6.243 4.30e-10 ***
## poly(ck, 3, raw = TRUE)1  1.102e-01  2.139e-02   5.153 2.57e-07 ***
## poly(ck, 3, raw = TRUE)2 -4.649e-04  1.381e-04  -3.367  0.00076 ***
## poly(ck, 3, raw = TRUE)3  6.448e-07  2.544e-07   2.535  0.01125 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 271.7124  on 11  degrees of freedom
## Residual deviance:   4.2525  on  8  degrees of freedom
## AIC: 33.658
##
## Number of Fisher Scoring iterations: 6

# All fits together
lines(ck, logistic(cbind(1, poly(ck, 2, raw = TRUE)) %*% heart2$coefficients),
      col = 2)
lines(ck, logistic(cbind(1, poly(ck, 3, raw = TRUE)) %*% heart3$coefficients),
      col = 3)
lines(ck, logistic(cbind(1, poly(ck, 4, raw = TRUE)) %*% heart4$coefficients),
      col = 4)
legend("bottomright", legend = c("Linear", "Quadratic", "Cubic", "Quartic"),
       col = 1:4, lwd = 2)
```



### Estimation by maximum likelihood

The estimation of $\boldsymbol{\beta}$ by MLE can be done in a unified framework, for all generalized linear models, thanks to the exponential family (5.10). Given $\{(\mathbf{x}_i, Y_i)\}_{i=1}^n$,[23] and employing a canonical link function (5.13), we have that

$$Y_i \mid (X_1 = x_{i1}, \ldots, X_p = x_{ip}) \sim \mathrm{E}(\theta_i, \phi, a, b, c), \quad i = 1, \ldots, n,$$

where

$$\begin{aligned}\theta_i &:= \eta_i := \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip},\\ \mu_i &:= \mathbb{E}[Y_i \mid X_1 = x_{i1}, \ldots, X_p = x_{ip}] = g^{-1}(\eta_i).\end{aligned}$$

Then, the log-likelihood is

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^n \left( \frac{Y_i\theta_i - b(\theta_i)}{a(\phi)} + c(Y_i, \phi) \right). \tag{5.20}$$

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

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^n \frac{(Y_i - \mu_i)}{g'(\mu_i)V_i} \mathbf{x}_i, \tag{5.21}$$

where $\mathbf{x}_i$ now represents the $i$-th row of the design matrix $\mathbf{X}$ and $V_i := \mathbb{V}\mathrm{ar}[Y_i] = a(\phi)b''(\theta_i)$. 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[24]

$$\left.\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}} (\boldsymbol{\beta}_{\text{new}} - \boldsymbol{\beta}_{\text{old}}) = -\left.\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}}. \tag{5.22}$$

A simplifying trick is to consider the *expectation* of $\left.\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}}$ in (5.22), rather than its actual value. By doing so, we can arrive to a neat iterative algorithm called *Iterative Reweighted Least Squares* (IRLS). We use the following well-known property of the *Fisher information matrix* of the MLE theory:

$$\mathbb{E}\left[\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top}\right] = -\mathbb{E}\left[\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \left(\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right)^\top\right].$$

Then, it can be seen that[25]

$$\mathbb{E}\left[\left.\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\text{old}}}\right] = -\sum_{i=1}^n w_i \mathbf{x}_i \mathbf{x}_i^\top = -\mathbf{X}^\top \mathbf{W} \mathbf{X}, \tag{5.23}$$

[23] We assume the randomness comes from the error present in $Y$ once $\mathbf{X}$ is given, not from the $\mathbf{X}$. This is implicit in the considered expectations.

[24] The system stems from a first-order Taylor expansion of the function $\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} : \mathbb{R}^{p+1} \to \mathbb{R}^{p+1}$ about the root $\hat{\boldsymbol{\beta}}$, where $\left.\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}} = \mathbf{0}$.

[25] Recall that $\mathbb{E}[(Y_i - \mu_i)(Y_j - \mu_j)] = \mathrm{Cov}[Y_i, Y_j] = \begin{cases} V_i, & i = j, \\ 0, & i \neq j, \end{cases}$ because of independence.

where $w_i := \frac{1}{V_i(g'(\mu_i))^2}$ and $\mathbf{W} := \mathrm{diag}(w_1, \ldots, w_n)$. Using this notation and from (5.21),

$$\left.\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\mathrm{old}}} = \mathbf{X}^\top \mathbf{W}(\mathbf{Y} - \boldsymbol{\mu}_{\mathrm{old}})\mathbf{g}^\top(\boldsymbol{\mu}_{\mathrm{old}}), \tag{5.24}$$

Substituting (5.23) and (5.24) in (5.22), we have:

$$\begin{aligned}\boldsymbol{\beta}_{\mathrm{new}} &= \boldsymbol{\beta}_{\mathrm{old}} - \mathbb{E}\left[\left.\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\mathrm{old}}}\right]^{-1} \left.\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\mathrm{old}}} \\ &= \boldsymbol{\beta}_{\mathrm{old}} + (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}(\mathbf{Y} - \boldsymbol{\mu}_{\mathrm{old}})\mathbf{g}^\top(\boldsymbol{\mu}_{\mathrm{old}}) \\ &= (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}\mathbf{z}, \end{aligned} \tag{5.25}$$

where $\mathbf{z} := \mathbf{X}\boldsymbol{\beta}_{\mathrm{old}} + (\mathbf{Y} - \boldsymbol{\mu}_{\mathrm{old}})\mathbf{g}^\top(\boldsymbol{\mu}_{\mathrm{old}})$ 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 $\boldsymbol{\beta}_{\mathrm{old}}$ with some initial estimation.
2. Compute $\boldsymbol{\mu}_{\mathrm{old}}$, $\mathbf{W}$, and $\mathbf{z}$.
3. Compute $\boldsymbol{\beta}_{\mathrm{new}}$ using (5.25).
4. Set $\boldsymbol{\beta}_{\mathrm{old}}$ as $\boldsymbol{\beta}_{\mathrm{new}}$.
5. Iterate Steps 2–4 until convergence, then set $\hat{\boldsymbol{\beta}} = \boldsymbol{\beta}_{\mathrm{new}}$.

In general, $\mathbb{E}\left[\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top}\right] \neq \frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top}$. Therefore, IRLS in general departures from the standard Newton–Raphson. However, if the **canonical link** is used, it can be seen that **the equality of $\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top}$ with its expectation is guaranteed** and IRLS is *exactly* the same as Newton–Raphson. In that case, $w_i = \frac{1}{g'(\mu_i)}$ (which simplifies the computation of $\mathbf{W}$ in the algorithm above).

## 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 $\hat{\boldsymbol{\beta}}$ through the theory of MLE. Again, the distribution is derived conditionally on the predictors' sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$. In other words, we assume that the randomness of $Y$ comes only from $Y \mid (X_1 = x_1, \ldots, X_p = x_p)$ and not from the predictors.

For the ease of exposition, we will **focus only 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 the intuition side 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*.[26] The reason is the higher complexity of the model in terms of nonlinearity. This is the **usual situation** for the majority of regression models.[27]

### 5.3.1 *Distributions of the fitted coefficients*

The distribution of $\hat{\boldsymbol{\beta}}$ is given by the asymptotic theory of MLE:

$$\hat{\boldsymbol{\beta}} \overset{a}{\sim} \mathcal{N}_{p+1}\left(\boldsymbol{\beta}, \mathcal{I}(\boldsymbol{\beta})^{-1}\right) \tag{5.26}$$

where $\overset{a}{\sim}$ $[\ldots]$ means "asymptotically distributed as $[\ldots]$ when $n \to \infty$" and

$$\mathcal{I}(\boldsymbol{\beta}) := -\mathbb{E}\left[\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top}\right]$$

is the *Fisher information matrix*. The name comes from the fact that *it measures the information available in the sample for estimating* $\boldsymbol{\beta}$. The "larger" the matrix (larger eigenvalues) is, the more precise the estimation of $\boldsymbol{\beta}$ is, because that results in smaller variances in (5.26).

The inverse of the Fisher information matrix is[28]

$$\mathcal{I}(\boldsymbol{\beta})^{-1} = (\mathbf{X}^\top \mathbf{V} \mathbf{X})^{-1}, \tag{5.27}$$

where $\mathbf{V} = \mathrm{diag}(V_1, \ldots, V_n)$ and $V_i = \mathrm{logistic}(\eta_i)(1 - \mathrm{logistic}(\eta_i))$, with $\eta_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$. In the case of the multiple linear regression, $\mathcal{I}(\boldsymbol{\beta})^{-1} = \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}$ (see (2.11)), so the presence of $\mathbf{V}$ here is a consequence of the heteroscedasticity of the logistic model.

The interpretation of (5.26) and (5.27) 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 $\mathbf{X}^\top \mathbf{V} \mathbf{X}$. As $n$ grows, the precision of the estimators increases.
  - *Weighted predictor sparsity* $(\mathbf{X}^\top \mathbf{V} \mathbf{X})^{-1}$. The more "disperse"[29] the predictors are, the more precise $\hat{\boldsymbol{\beta}}$ is. When $p = 1$, $\mathbf{X}^\top \mathbf{V} \mathbf{X}$ is a weighted version of $s_x^2$.

Figure 5.3.1 aids visualizing these insights.

[26] That work quite well in practice and deliver many valuable insights.

[27] The linear model is an exception in terms of exact and simple inference, not the rule.

[28] Recall expression (5.23) for the general case of $\mathcal{I}(\boldsymbol{\beta})$.

[29] Undestood as small $|(\mathbf{X}^\top \mathbf{V} \mathbf{X})^{-1}|$.

**The precision of $\hat{\boldsymbol{\beta}}$ is affected by the value of $\boldsymbol{\beta}$**, 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 $\boldsymbol{\beta}$ (see (2.11)). The reason is partially due to the **heteroscedasticity** of logistic regression, which implies a dependence of the variance of $Y$ in the logistic curve, hence in $\boldsymbol{\beta}$.

Similar to linear regression, the problem with (5.26) and (5.27) is that $\mathbf{V}$ *is unknown* in practice because it depends on $\boldsymbol{\beta}$. Plugging-in the estimate $\hat{\boldsymbol{\beta}}$ into $\boldsymbol{\beta}$ in $\mathbf{V}$ gives the estimator $\hat{\mathbf{V}}$. Now we can use $\hat{\mathbf{V}}$ to get

$$\frac{\hat\beta_j-\beta_j}{\hat{\mathrm{SE}}(\hat\beta_j)} \overset{a}{\sim} \mathcal{N}(0,1), \quad \hat{\mathrm{SE}}(\hat\beta_j)^2 := v_j \tag{5.28}$$

where

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

The LHS of (5.28) is the **Wald statistic** for $\beta_j$, $j=0,\ldots,p$. They are employed for building marginal confidence intervals and hypothesis tests, in a completely analogous way to how the $t$-statistics in linear regression operate.

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 predictors' sample $x_1,\ldots,x_n$ are fixed and new responses $Y_1,\ldots,Y_n$ are generated each time from a simple logistic model $Y \mid X=x \sim \mathrm{Ber}(\mathrm{logistic}(\beta_0+\beta_1 x))$. Application available here.

### 5.3.2 *Confidence intervals for the coefficients*

Thanks to (5.28), we can have the $100(1-\alpha)\%$ CI for the coefficient $\beta_j$, $j=0,\ldots,p$:

$$\left(\hat\beta_j \pm \hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2}\right) \tag{5.29}$$

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.[30] So the $100(1-\alpha)\%$ CI for $e^{\beta_j}$, $j=0,\ldots,p$, is

$$e^{\left(\hat\beta_j\pm\hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2}\right)}.$$

Of course, this CI is **not** the same as $\left(e^{\hat\beta_j} \pm e^{\hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2}}\right)$, which is *not* a valid CI for $e^{\beta_j}$!

### 5.3.3 *Testing on the coefficients*

The distributions in (5.28) also allow us to conduct a formal hypothesis test on the coefficients $\beta_j$, $j=0,\ldots,p$. For example, the test for significance:

$$H_0 : \beta_j = 0$$

[30] Because $e^{\hat\beta_j-\hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2}} \leq e^{\beta_j} \leq e^{\hat\beta_j+\hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2}} \iff \hat\beta_j-\hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2} \leq \beta_j \leq \hat\beta_j+\hat{\mathrm{SE}}(\hat\beta_j)z_{\alpha/2}$ since the exponential is a monotone increasing function.

for $j = 0, \ldots, p$. 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

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

which is asymptotically distributed as a $\mathcal{N}(0,1)$ *under the (veracity of) the null hypothesis*. $H_0$ is tested *against* the *two-sided* alternative hypothesis $H_1 : \beta_j \neq 0$.

*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**.

The tests for significance are built-in in the `summary` function. However, due to discrepancies between `summary` and `confint`, a note of caution is required when applying the previous rule of thumb for rejecting $H_0$ in terms of the CI.

The significances given in `summary` and the output of `MASS::confint` are *slightly* incoherent and the previous rule of thumb **does not apply**. The reason is because `MASS::confint` is using a more sophisticated method (profile likelihood) to estimate the standard error of $\hat{\beta}_j$, $\hat{\mathrm{SE}}(\hat{\beta}_j)$, and not the asymptotic distribution behind the 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 will still be completely valid. For the contents of this course we prefer `confint.default` due to its better interpretability. This point is exemplified in the next section.

### 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 $4.2233 \times 10^6$ more likely than `fail.field`=0 (using the CI for $e^{\beta_0}$).
- **`temp` has a significantly negative effect on 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 0.4507 and 0.9643 (using the CI for $e^{\beta_1}$).

This completes the answers to Q1 and Q2.

We conclude by 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
##                  2.5 %     97.5 %
## (Intercept)  1.3364047 17.7834329
## temp        -0.9237721 -0.1089953
confint(nasa, level = 0.99) # temp still significant
##                  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**

$$\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = g^{-1}(\eta) = g^{-1}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)$$

by means of $\hat{\eta} := \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$ 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.[31]

For the logistic model, the prediction of the conditional response follows immediately from $\mathrm{logistic}(\hat{\eta})$:

$$\hat{Y} \mid (X_1 = x_1, \ldots, X_p = x_p) = \begin{cases} 1, & \text{with probability } \mathrm{logistic}(\hat{\eta}), \\ 0, & \text{with probability } 1 - \mathrm{logistic}(\hat{\eta}). \end{cases}$$

As a consequence, we can predict $Y$ as $1$ if $\mathrm{logistic}(\hat{\eta}) > \frac{1}{2}$ 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`:

[31] For example, the CI for the conditional response in the logistic model is not be very informative, as it can either be $\{0\}$, $\{1\}$ or $\{0, 1\}$. Predictions and CIs for the conditional response are carried out on a model-by-model basis.

- 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 behavior than 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 the computation of CIs for prediction is not implemented and has to be done manually from the standard errors returned when se.fit = TRUE (see Section 5.4.1).

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 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 evidence raised by the simple analysis 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 the function to our model:

```
# 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)
```

Finally, let's answer Q4 and see what was the probability of having at least one incident with the O-rings if the launch was postponed until the temperature was above 11.67 degrees Celsius.

```
# Estimated probability for launching at 53 degrees Fahrenheit
predictCIsLogistic(nasa, newdata = data.frame(temp = 11.67))
##         fit       lwr       upr
## 1 0.9382822 0.3504908 0.9976707
```

The maximum predicted probability is 0.94. Notice that this is the *maximum* in accordance to the suggestion of launching *above* 11.67 degrees Celsius. The probability of having at least one incident[32] with the O-rings is still very high.

**Exercise 5.6.** For the `challenger` dataset, do the following:

a. Regress `fail.nozzle` on `temp` and `pres.nozzle`.
b. Compute the predicted probability of `fail.nozzle=1` for `temp` $= 15$ and `pres.nozzle` $= 200$. What is the predicted probability for `fail.nozzle=0`?
c. 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 the sample $\{(\mathbf{x}_i, Y_i)\}_{i=1}^n$. This perfect model, known as the *saturated model*, is the model that perfectly fits the data, in the sense that the fitted responses $(\hat{Y}_i)$ equal the observed responses $(Y_i)$. For example, in logistic regression this would be the model such that

$$\hat{\mathbb{P}}[Y = 1 \mid X_1 = X_{i1}, \ldots, X_p = X_{ip}] = Y_i, \quad i = 1, \ldots, n.$$

Figure 5.6 shows $a^{33}$ saturated model and a fitted logistic model.

Formally, the deviance is defined through the difference of the log-likelihoods between the fitted model, $\ell(\hat{\boldsymbol{\beta}})$, and the saturated

[32] Whether at this temperature it would have been a *fatal* incident or not is left to speculation.

[33] Several are possible depending on the interpolation between points.

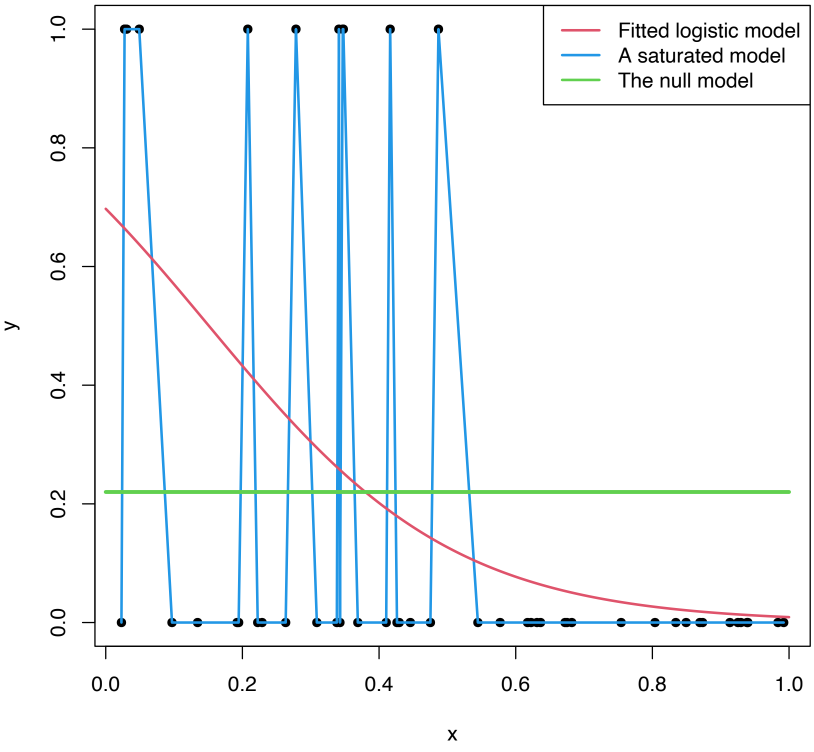


Figure 5.6: Fitted logistic regression versus a saturated model and the null model.

model, $\ell_s$. Computing $\ell_s$ amounts to substitute $\mu_i$ by $Y_i$ in (5.20). If the canonical link function is used, this corresponds to setting $\theta_i = g(Y_i)$ (recall (5.12)). The deviance is then defined as:

$$D := -2\left[\ell(\hat{\boldsymbol{\beta}}) - \ell_s\right]\phi.$$

The log-likelihood $\ell(\hat{\boldsymbol{\beta}})$ is always smaller than $\ell_s$ (the saturated model is more likely given the sample, since it is the sample itself). As a consequence, the *deviance is always greater than or equal to 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

$$\begin{aligned} D &= -\frac{2}{a(\phi)}\sum_{i=1}^n \left(Y_i\hat{\theta}_i - b(\hat{\theta}_i) - Y_i g(Y_i) + b(g(Y_i))\right)\phi \\ &= \frac{2\phi}{a(\phi)}\sum_{i=1}^n \left(Y_i(g(Y_i) - \hat{\theta}_i) - b(g(Y_i)) + b(\hat{\theta}_i)\right). \end{aligned} \tag{5.30}$$

In most of the cases, $a(\phi) \propto \phi$, so the deviance *does not depend on* $\phi$. Expression (5.30) is interesting, since it delivers the following key insight:

The **deviance generalizes 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 this insight, let's consider the linear model in (5.30) by setting $\phi = \sigma^2$, $a(\phi) = \phi$, $b(\theta) = \frac{\theta^2}{2}$, $c(y, \phi) = -\frac{1}{2}\{\frac{y^2}{\phi} + \log(2\pi\phi)\}$, and $\theta = \mu = \eta$.[34] Then:

$$\begin{aligned} D &= \frac{2\sigma^2}{\sigma^2}\sum_{i=1}^n \left(Y_i(Y_i - \hat{\eta}_i) - \frac{Y_i^2}{2} + \frac{\hat{\eta}_i^2}{2}\right) \\ &= \sum_{i=1}^n \left(2Y_i^2 - 2Y_i\hat{\eta}_i - Y_i^2 + \hat{\eta}_i^2\right) \\ &= \sum_{i=1}^n (Y_i - \hat{\eta}_i)^2 \\ &= \mathrm{RSS}(\hat{\boldsymbol{\beta}}), \end{aligned} \tag{5.31}$$

since $\hat{\eta}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$. Remember that $\mathrm{RSS}(\hat{\boldsymbol{\beta}})$ is just another name for the SSE.

A benchmark for evaluating the scale 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 model without predictors**, the one featuring only an intercept, to the perfect model. In logistic regression, this model is

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{Ber}(\mathrm{logistic}(\beta_0))$$

and, in this case, $\hat{\beta}_0 = \mathrm{logit}(\frac{m}{n}) = \log\left(\frac{\frac{m}{n}}{1-\frac{m}{n}}\right)$ where $m$ is the number of 1's in $Y_1, \ldots, Y_n$ (see Figure 5.6).

Using again (5.30), we can see that the **null deviance** is a generalization of the **total sum of squares of the linear model** (see Section 2.6):

$$D_0 = \sum_{i=1}^n (Y_i - \hat{\eta}_i)^2 = \sum_{i=1}^n \left(Y_i - \hat{\beta}_0\right)^2 = \mathrm{SST},$$

since $\hat{\beta}_0 = \bar{Y}$ because there are no predictors.

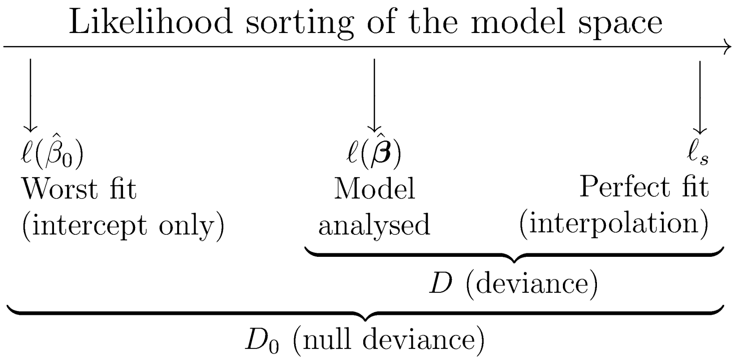


Figure 5.7: Pictorial representation of the deviance ($D$) and the null deviance ($D_0$).

Using the deviance and the null deviance, we can compare how much the model has improved by adding the predictors $X_1, \ldots, X_p$

[34] The canonical link function $g$ is the identity, check (5.12) and (5.13).

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:

$$R^2 := 1 - \frac{D}{D_0} \overset{\substack{\text{linear}\\ \text{model}}}{=} 1 - \frac{\text{SSE}}{\text{SST}}.$$

The $R^2$ for generalized linear models is a measure that shares the same philosophy with the determination coefficient in linear regression: it is a proportion of how good the model fit is. If perfect, $D = 0$ and $R^2 = 1$. If the predictors are not model-related with $Y$, then $D = D_0$ and $R^2 = 0$.

However, this $R^2$ has a different interpretation than 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.

The deviance is returned by `summary`. It is important to recall that R refers to the deviance as the `'Residual deviance'` and the null deviance is referred to as `'Null deviance'`.

```
# 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
## -0.852  -0.852  -0.852   1.542   1.542
##
## 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

# Compute 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] -4.440892e-16
```

A quantity related with the deviance is the **scaled deviance**:

$$D^* := \frac{D}{\phi} = -2\left[\ell(\hat{\boldsymbol{\beta}}) - \ell_s\right].$$

If $\phi = 1$, such as in the binomial or Poisson regression models, then both the deviance and the scaled deviance agree. The scaled deviance has asymptotic distribution

$$D^* \overset{a}{\sim} \chi^2_{n-p-1}, \tag{5.32}$$

where $\chi^2_k$ is the *Chi-squared distribution with k degrees of freedom*. In the case of the linear model, $D^* = \frac{1}{\sigma^2}\mathrm{RSS}$ is *exactly* distributed as a $\chi^2_{n-p-1}$.

The result (5.32) provides a way of estimating $\phi$ when it is unknown: match $D^*$ with the expectation $\mathbb{E}\left[\chi^2_{n-p-1}\right] = n - p - 1$. This provides

$$\hat{\phi}_D := \frac{-2(\ell(\hat{\boldsymbol{\beta}}) - \ell_s)}{n-p-1},$$

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 $M_2$, with $p_2$ predictors and another model, say $M_1$, with $p_1 < p_2$ predictors that are contained in the set of predictors of the M2. In other words, assume $M_1$ is **nested** within $M_2$. Then we can test the null hypothesis that the extra coefficients of $M_2$ are simultaneously zero. For example, if $M_1$ has the coefficients $\{\beta_0, \beta_1, \ldots, \beta_{p_1}\}$ and M2 has coefficients $\{\beta_0, \beta_1, \ldots, \beta_{p_1}, \beta_{p_1+1}, \ldots, \beta_{p_2}\}$, we can test

$$H_0 : \beta_{p_1+1} = \cdots = \beta_{p_2} = 0 \quad \text{vs.} \quad H_1 : \beta_j \neq 0 \text{ for any } p_1 < j \leq p_2.$$

This can be done by means of the statistic[35]

$$D^*_{p_1} - D^*_{p_2} \overset{a,H_0}{\sim} \chi^2_{p_2-p_1}. \tag{5.33}$$

If $H_0$ is true,[36] 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 $\chi^2_{p_2-p_1}$*, denoted as $\chi^2_{\alpha;p_2-p_1}$.

$D^*$ apparently removes the effects of $\phi$, but it is still dependent on $\phi$, since this is hidden in the likelihood (see (5.30)). Therefore, $D^*$ cannot be computed unless $\phi$ is known, which forbids using (5.33). Hopefully, this dependence is removed by employing (5.32) and (5.33) and assuming that they are asymptotically independent. This gives the *F-test* for $H_0$:

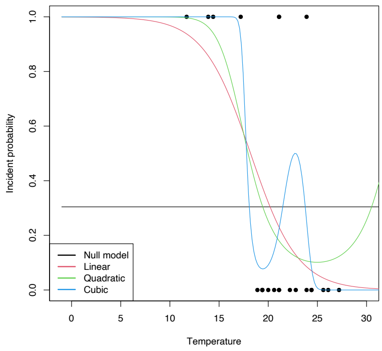
$$F = \frac{(D^*_{p_1} - D^*_{p_2})/(p_2 - p_1)}{D^*_{p_2}/(n - p_2 - 1)} = \frac{(D_{p_1} - D_{p_2})/(p_2 - p_1)}{D_{p_2}/(n - p_2 - 1)} \overset{a,H_0}{\sim} F_{p_2-p_1,n-p_2-1}.$$

Note that $F$ 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** seen in Section 2.6: take $p_1 = 0$ and $p_2 = p$ and then one tests 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***. This is illustrated in the following code, which coincidentally also illustrates the inclusion of nonlinear transformations on the predictors.

```r
# 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)

# 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)
```



[35] Note that $D^*_{p_1} > D^*_{p_2}$ because $\ell(\hat{\boldsymbol{\beta}}_{p_1}) < \ell(\hat{\boldsymbol{\beta}}_{p_2})$.

[36] In other words, $M_2$ does not add significant predictors when compared with the *submodel* $M_1$.

```
# Rˆ2's
r2glm(nasa0)
## [1] -4.440892e-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")
## 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 significant

# 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** (increases) 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.
4. Roughly speaking, the **BIC is consistent** in performing model-selection, and penalizes more the complex models when compared to AIC and LOOCV.

In addition, variable selection can also be done with the lasso for generalized linear models, as seen in Section 5.8.

Stepwise selection can be done through `MASS::stepAIC` as in linear models.[37] 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
##
```

[37] 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.

```
## 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
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
```

MASS::stepAIC works analogously as in linear regression. An illustration is given next for a predicting binary variable that measures whether a Boston suburb (Boston dataset from Section 3.1) is wealthy 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.[38] The cutoff 25000$ corresponds to the 25% richest suburbs.

```
# Boston dataset
data(Boston, package = "MASS")

# 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
```

[38] This is a common way of generating a binary variable from a quantitative variable.

```
## -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 <- MASS::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 | $\hat{Y} = 0$ | $\hat{Y} = 1$ |
|---|---|---|
| $Y = 0$ | $\text{Correct}_0$ | $\text{Incorrect}_{01}$ |
| $Y = 1$ | $\text{Incorrect}_{10}$ | $\text{Correct}_1$ |

The *hit ratio*, $\frac{\text{Correct}_0 + \text{Correct}_1}{n}$, is the percentage of correct classifications. The hit matrix is easily computed with the `table` function which, whenever called with two vectors, computes the cross-table between those two vectors.

```
# Fitted probabilities for Y = 1
nasa$fitted.values
##          1          2          3          4          5          6          7          8          9         10
## 0.42778935 0.23014393 0.26910358 0.32099837 0.37772880 0.15898364 0.12833090 0.23014393 0.85721594 0.60286639
##         11         12         13         14         15         16         17         18         19         20
## 0.23014393 0.04383877 0.37772880 0.93755439 0.37772880 0.08516844 0.23014393 0.02299887 0.07027765 0.03589053
##         21         22         23
## 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 classifications)
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 data-splitting/cross-validation is therefore needed to estimate unbiasedly the hit matrix.

**Exercise 5.7.** For the `Boston` dataset, do the following:

a. Compute the hit matrix and hit ratio for the regression `I(medv > 25) ~ .`.
b. Fit `I(medv > 25) ~ .` but now using only the first 300 observations of `Boston`, the training dataset.
c. 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.
d. 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 $\boldsymbol{\beta}$ and $\phi$.

In general, if we employ the canonical link function, we assume that the data has been generated from (independence is implicit)

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{E}(\eta(x_1, \ldots, x_p), \phi, a, b, c), \tag{5.34}$$

in such a way that

$$\mu = \mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = g^{-1}(\eta),$$

and $\eta(x_1, \ldots, x_p) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$.

In the case of the logistic and Poisson regressions, both with canonical link functions, the general model takes the form (independence is implicit)

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{Ber}\left(\mathrm{logistic}(\eta)\right), \tag{5.35}$$

$$Y \mid (X_1 = x_1, \ldots, X_p = x_p) \sim \mathrm{Pois}\left(e^{\eta}\right). \tag{5.36}$$

The assumptions behind (5.34), (5.35), and (5.36) are:

i. **Linearity** in the transformed expectation: $\mathbb{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = g^{-1}\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right)$.
ii. **Response distribution**: $Y \mid \mathbf{X} = \mathbf{x} \sim \mathrm{E}(\eta(\mathbf{x}), \phi, a, b, c)$ (the scale $\phi$ and the functions $a, b, c$ are *constant* for $\mathbf{x}$).
iii. **Independence**: $Y_1, \ldots, Y_n$ are independent, conditionally on $\mathbf{X}_1, \ldots, \mathbf{X}_n$.

There are two important points of the linear model assumptions "missing" here:

- *Where is homoscedasticity?* Homoscedasticity is specific to certain exponential family distributions in which $\theta$ does not affect the variance. This is not the case for binomial or Poisson distributions variables, which result in heteroscedastic models. Also, homoscedasticity is the consequence 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" directly using (2.8).

Recall that:

- Nothing is said about the distribution of $X_1, \ldots, X_p$. They could be deterministic or random. They could be discrete or continuous.
- $X_1, \ldots, X_p$ are not required to be independent between them.

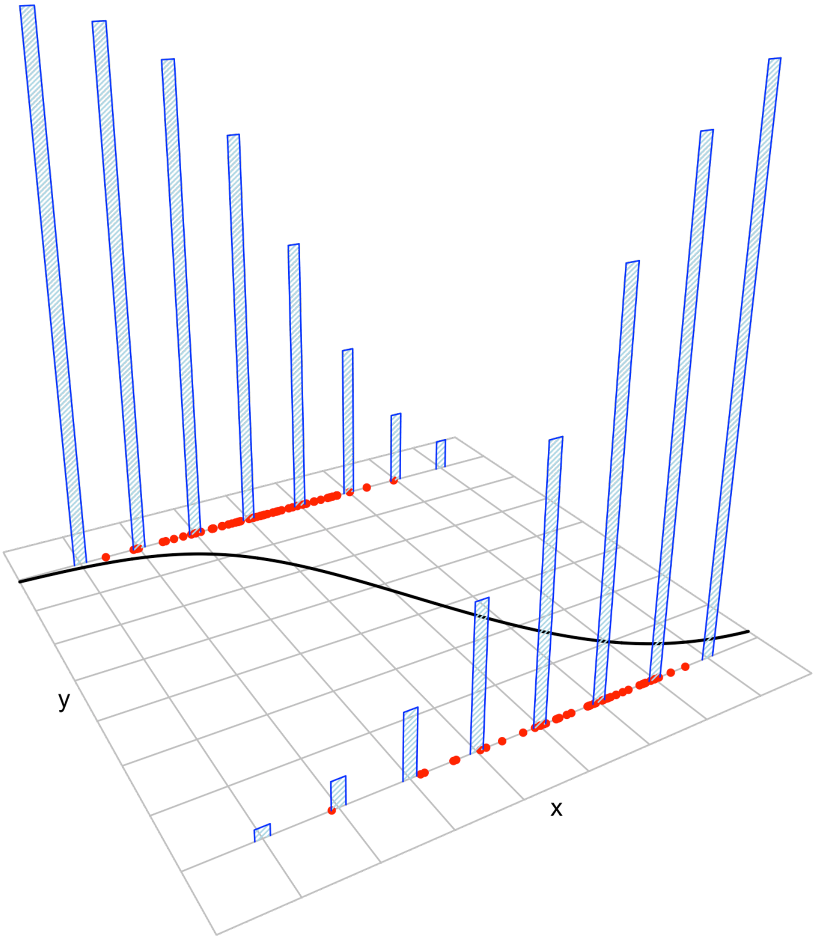


Figure 5.8: The key concepts of the logistic model. The red points represent a sample with population logistic curve $y = \text{logistic}(\beta_0 + \beta_1 x)$, shown in black. The blue bars represent the conditional probability mass functions of $Y$ given $X = x$, whose means lie in the logistic curve.

Checking the assumptions of a generalized linear model is more complicated than what we did in Section 3.5. The reason is the **heterogeneity and heteroscedasticity of the responses**, which makes the inspection of the residuals $Y_i - \hat{Y}_i$ complicated.[39] 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 seen in (5.31). The deviance can be expressed into a sum of terms associated with each datum (recall, e.g., (5.30)):

$$D = \sum_{i=1}^{n} d_i.$$

For the linear model, $d_i = \hat{\varepsilon}_i^2$, since $D = \mathrm{RSS}(\hat{\boldsymbol{\beta}})$. Based on this, we can define the deviance residuals as

$$\hat{\varepsilon}_i^D := \mathrm{sign}(Y_i - \hat{\mu}_i)\sqrt{d_i}, \quad i = 1, \ldots, n$$

and have a generalization of $\hat{\varepsilon}_i$ for generalized linear models. This definition has interesting distributional consequences. From (5.32), we know that $D^* \overset{a}{\sim} \chi^2_{n-p-1}$. This *suggests*[40] that

$$\hat{\varepsilon}_i^D \text{ are approximately normal if the model holds.} \tag{5.37}$$

The previous statement is of *variable accuracy*, depending on the model, sample size, and distribution of the predictors.[41] In the linear model, it is exact and $(\hat{\varepsilon}_1^D, \ldots, \hat{\varepsilon}_n^D)$ are distributed *exactly* as a $\mathcal{N}_n(\mathbf{0}, \sigma^2\mathbf{X}^\top(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X})$.

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 (since several definitions of residuals are possible). They are also the residuals returned in R, either by `glm$residuals` or by `residuals(glm)`.

The definition of the deviance residuals has interesting connections:

- If the canonical function is employed, then $\sum_{i=1}^n \hat{\varepsilon}_i^D = 0$, as in the linear model.
- 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 $\hat{\sigma}^2$ in the linear model.
- Therefore, $\hat{\phi}_D$ is the sample variance of $\hat{\varepsilon}_1^D, \ldots, \hat{\varepsilon}_n^D$, which suggets that $\phi$ is the asymptotic variance of the population deviance residuals, in other words, $\mathbb{V}\mathrm{ar}[\varepsilon^D] \approx \phi$.
- The deviance residuals equal the standard residuals in the linear model.

The script used for generating the following (perhaps surprising) Figures 5.9–5.20 is available here.

[39] Also, due to the variety of ranges the response $Y$ is allowed to take in generalized linear models.

[40] The logic behind the statement is that a $\chi^2_k$ random variable has the same distribution as the sum of $k$ independent squared $\mathcal{N}(0, 1)$ variables.

[41] Practical counterexamples to the statement are possible, see Figures 5.13 and 5.15.

### 5.7.1 *Linearity*

Linearity between the *transformed expectation* of $Y$ and the predictors $X_1, \ldots, X_p$ 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**.

**How to check it**

We use the *residuals vs. fitted values plot*, which for generalized linear models is the scatterplot of $\{(\hat{\eta}_i, \hat{\varepsilon}_i^D)\}_{i=1}^n$. Recall that it is **not** the scatterplot of $\{(\hat{Y}_i, \hat{\varepsilon}_i)\}_{i=1}^n$. Under linearity, we expect that there is **no trend in the residuals** $\hat{\varepsilon}_i^D$ with respect to $\hat{\eta}_i$, **in addition to the patterns inherent to the nature of the response**. If nonlinearities are observed, it is worth plotting the regression terms of the model via `termplot`.

**What to do it fails**

Using an adequate nonlinear transformation for the problematic predictors or adding interaction terms might be helpful. Alternatively, considering a nonlinear transformation $f$ for the response $Y$ might also be helpful, at expenses of the same comments given in Section 3.5.1.

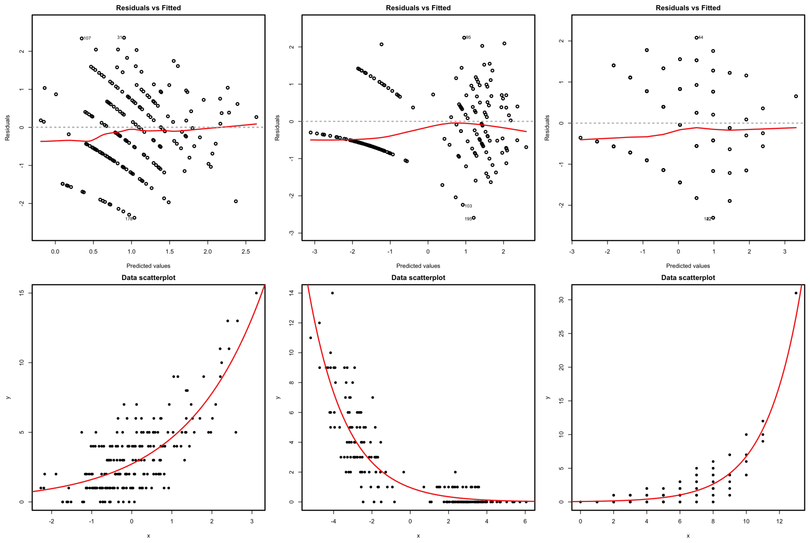


Figure 5.9: Residuals vs. fitted values plots (first row) for datasets (second row) ***respecting*** the linearity assumption in **Poisson** regression.

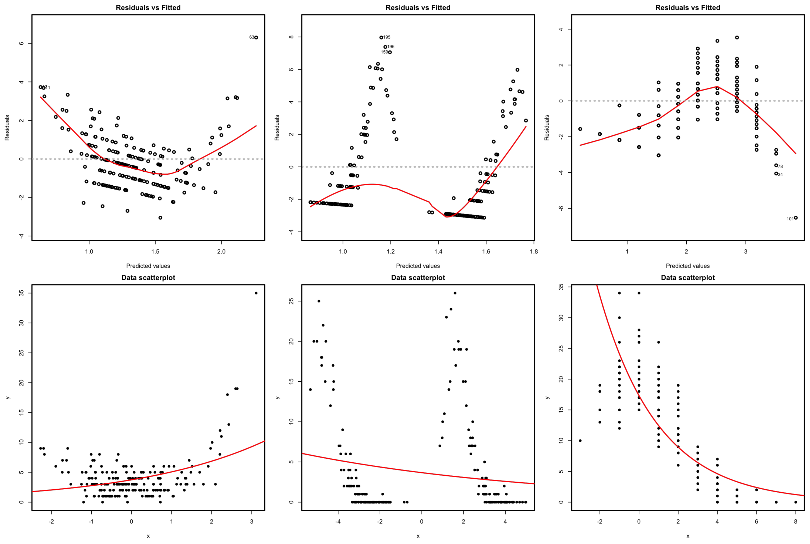


Figure 5.10: Residuals vs. fitted values plots (first row) for datasets (second row) ***violating*** the linearity assumption in **Poisson** regression.

## *5.7.2 Response distribution*

The *approximate* normality in the deviance residuals allows us 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**.[42] The bad news is that we have to pay an important price in terms of **inexactness**, since we employ an **asymptotic distribution**. The speed of this asymptotic convergence and the effective validity of (5.37) largely depends on several aspects: distribution of the response, sample size, and 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

[42] This would be the rigorous approach, but it is notably more complex.

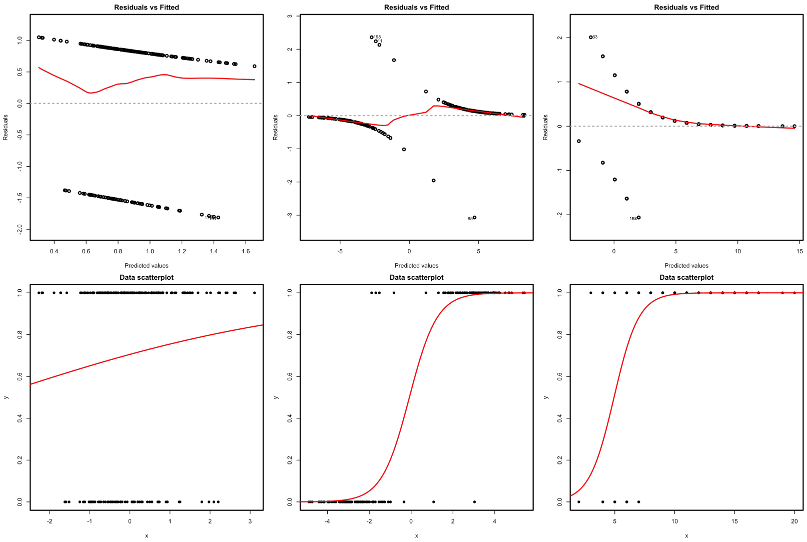


Figure 5.11: Residuals vs. fitted values plots (first row) for datasets (second row) ***respecting*** the linearity assumption in **logistic** regression.

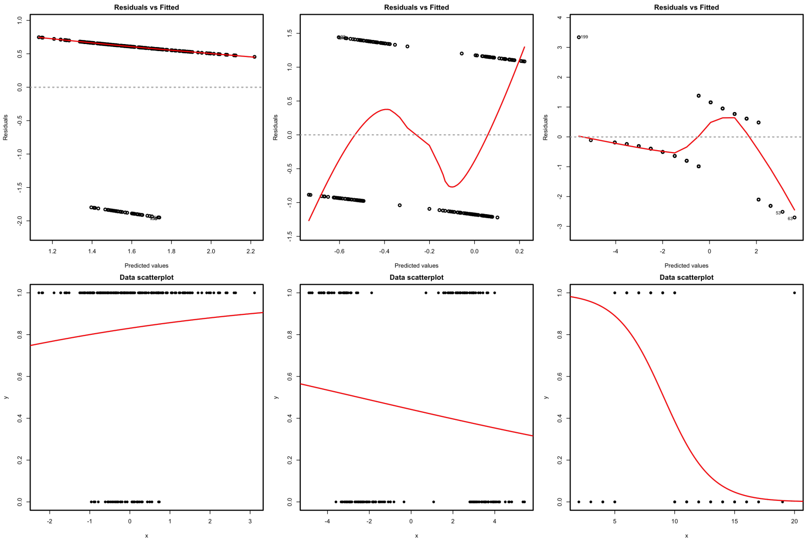


Figure 5.12: Residuals vs. fitted values plots (first row) for datasets (second row) ***violating*** the linearity assumption in **logistic** regression.

the **points to align with the diagonal line**. It is usual to 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 perfectly correct, see below. The reason is simply that the deviance residuals are significantly non-normal, which happens often in logistic regression.

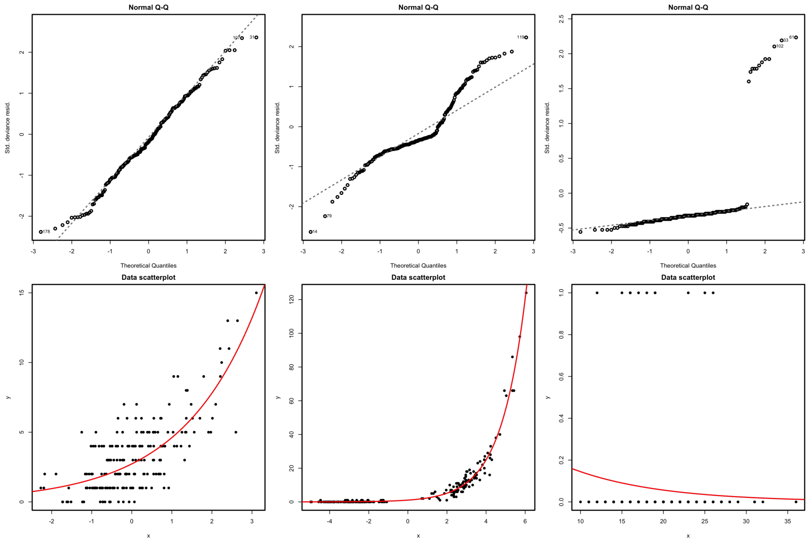


Figure 5.13: QQ-plots for the deviance residuals (first row) for datasets (second row) ***respecting*** the response distribution assumption for **Poisson** regression.

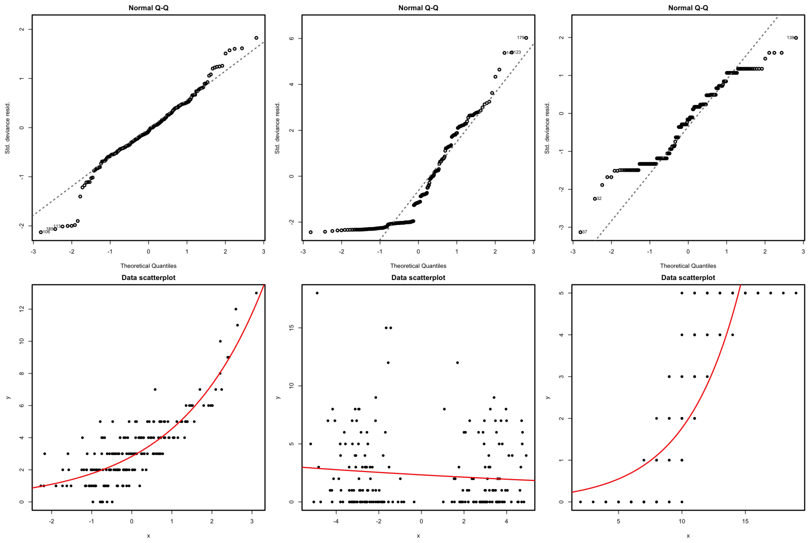


Figure 5.14: QQ-plots for the deviance residuals (first row) for datasets (second row) ***violating*** the response distribution assumption for **Poisson** regression.

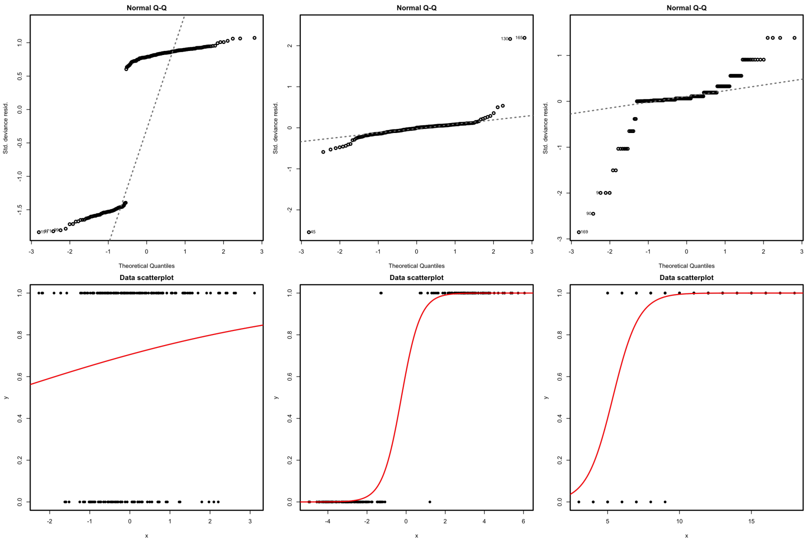


Figure 5.15: QQ-plots for the deviance residuals (first row) for datasets (second row) ***respecting*** the response distribution assumption for **logistic** regression.

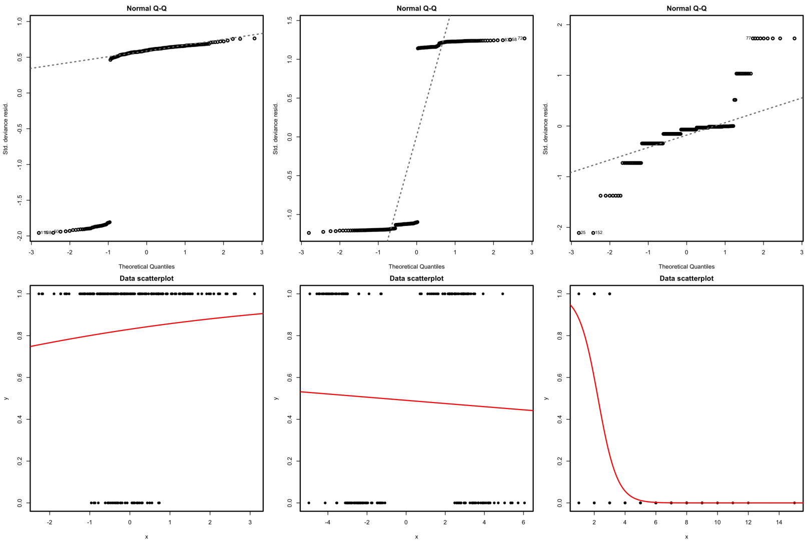


Figure 5.16: QQ-plots for the deviance residuals (first row) for datasets (second row) ***violating*** the response distribution assumption for **logistic** regression.

### What to do if it 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 one of the transformations discussed in Section 3.5.2, of course at the price of modeling 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 $X_1, \ldots, X_p$ 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 seen in Section 3.5.4.

### What to do if it 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.

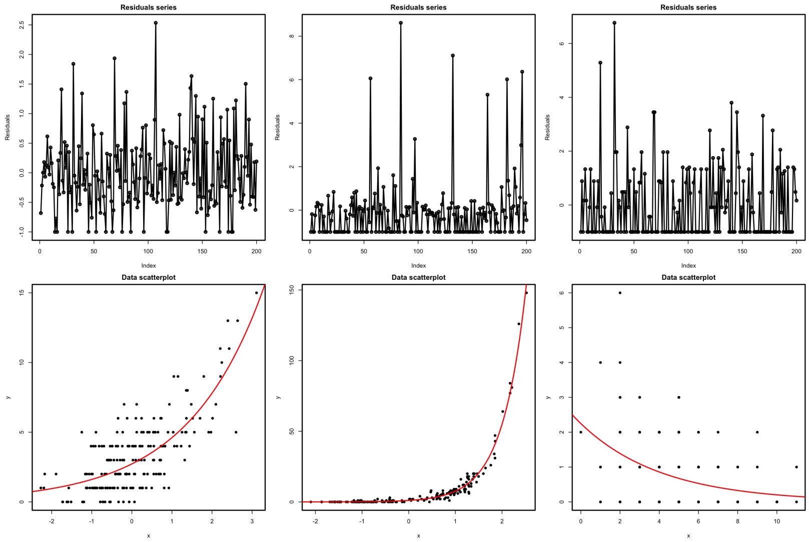


Figure 5.17: Serial plots of the residuals (first row) for datasets (second row) ***respecting*** the independence assumption for **Poisson** regression.

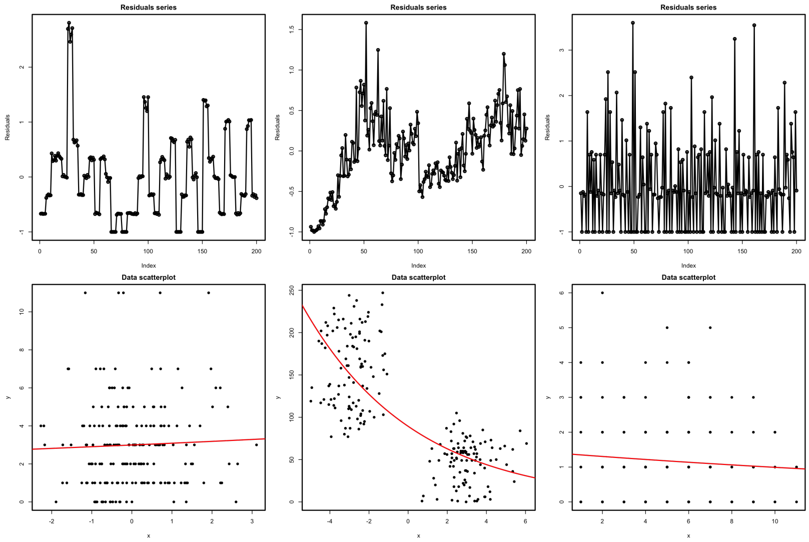


Figure 5.18: Serial plots of the residuals (first row) for datasets (second row) ***violating*** the independence assumption for **Poisson** regression.

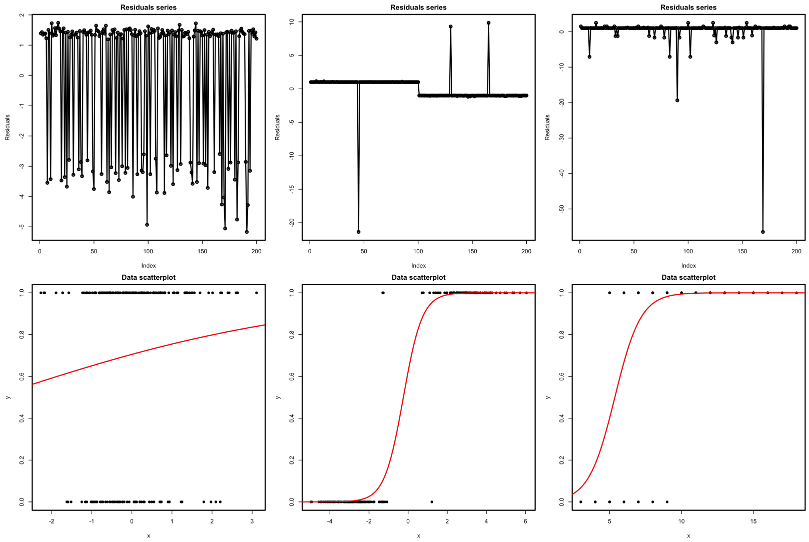


Figure 5.19: Serial plots of the residuals (first row) for datasets (second row) ***respecting*** the independence assumption for **logistic** regression.

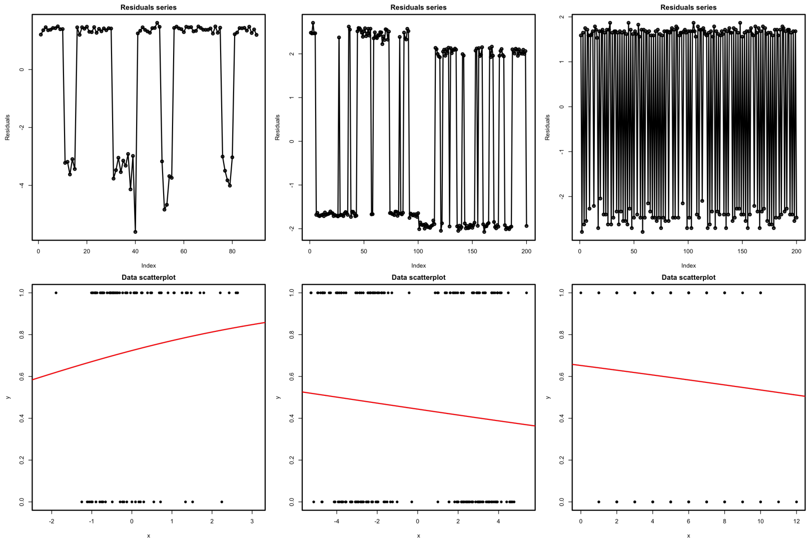


Figure 5.20: Serial plots of the residuals (first row) for datasets (second row) ***violating*** the independence assumption for **logistic** regression.

### 5.7.4 *Multicollinearity*

Multicollinearity can also be present in generalized linear models. Despite the nonlinear effect of the predictors on the response, **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 will be hard to separate from the rest of correlated predictors.

Then, a useful way of detecting multicollinearity is to inspect the VIF of each coefficient. 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)

# 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 0.38254782  0.2142011 -0.5261464  0.20198825
## x2  0.3825478 1.00000000  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")
car::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)
##
## 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
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

# Generalized variance inflation factors normal
car::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 straighforwardly in generalized linear models. The situation 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 how it was done in 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 $\boldsymbol{\beta}$ 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 $\boldsymbol{\beta}$:[43]

$$-\ell(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} (\alpha|\beta_j| + (1-\alpha)|\beta_j|^2). \tag{5.38}$$

As in Section 4.1, **ridge** regression corresponds to $\alpha = 0$ (quadratic penalty) and **lasso** to $\alpha = 1$ (absolute value penalty). Obviously, if $\lambda = 0$, we are back to the generalized linear models theory. The optimization of (5.38) gives

$$\hat{\boldsymbol{\beta}}_{\lambda,\alpha} := \arg \min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} \left\{ -\ell(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} (\alpha|\beta_j| + (1-\alpha)|\beta_j|^2) \right\}. \tag{5.39}$$

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.

[43] We are now *minimizing* $-\ell(\boldsymbol{\beta})$, the *negative* log-likelihood.

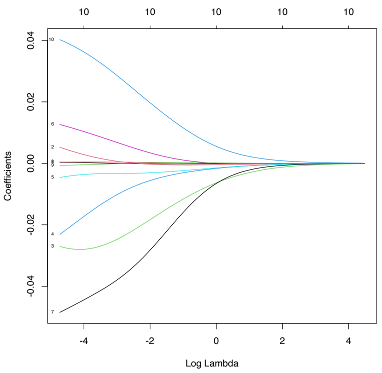
As in linear models, the predictors need to be standardized if they have a different nature.

We illustrate the shrinkage in generalized linear models with the `ISLR::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.

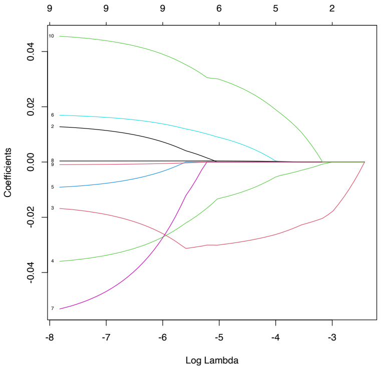
```r
# Load data
data(Hitters, package = "ISLR")

# 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.

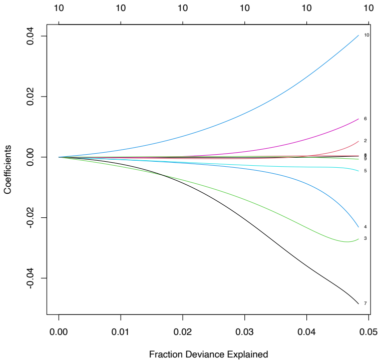
```r
# Ridge and lasso regressions
library(glmnet)
ridgeMod <- glmnet(x = x, y = y, alpha = 0, family = "binomial")
lassoMod <- glmnet(x = x, y = y, alpha = 1, family = "binomial")

# Solution paths versus lambda
plot(ridgeMod, label = TRUE, xvar = "lambda")
```



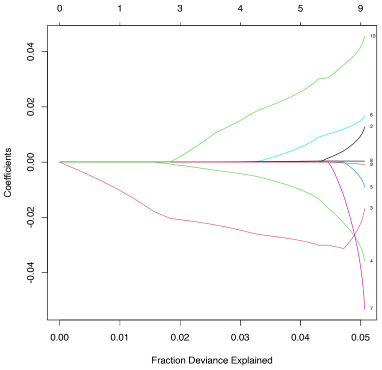
```r
plot(lassoMod, label = TRUE, xvar = "lambda")
```



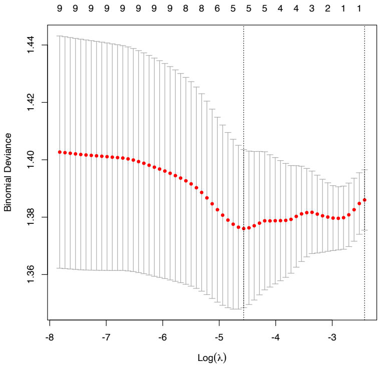
```r
# Versus the percentage of deviance explained
plot(ridgeMod, label = TRUE, xvar = "dev")
```

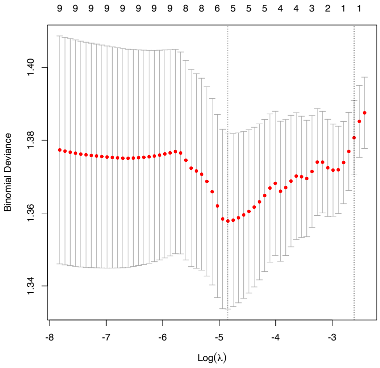


```r
plot(lassoMod, label = TRUE, xvar = "dev")
```



```r
# 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 minimizes the CV error and "one standard error rule"'s lambda
kcvLasso$lambda.min
## [1] 0.01039048
kcvLasso$lambda.1se
## [1] 0.08829343

# Leave-one-out cross-validation -- similar result
ncvLasso <- cv.glmnet(x = x, y = y, alpha = 1, nfolds = nrow(Hitters),
                      family = "binomial")
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"
##                       s1
## (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 Appendix E to avoid spurious findings.

```
# Analyze 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 Appendix A.5)

# 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(12345678)
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")
predict(ncvLasso, type = "coefficients", s = ncvLasso$lambda.1se)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.27240020
## AtBat        .
## Hits         .
## HmRun       -0.01255322
## 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
## -1.1170  -1.0177  -0.8517   1.3173   1.6896
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.14389    0.25754  -0.559   0.5764
## HmRun       -0.03255    0.01955  -1.665   0.0958 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 216.54  on 162  degrees of freedom
## Residual deviance: 213.64  on 161  degrees of freedom
## AIC: 217.64
##
## 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 and hit ratio
H <- table(pred > 0.5, y[!train] == "A") # ("A" was the reference level)
H
##
##          FALSE TRUE
##   FALSE     5   18
##   TRUE     57   83
sum(diag(H)) / sum(H) # Almost like tossing a coin!
## [1] 0.5398773
```

From the above analysis, we can conclude that there are no significant differences between the players of both leagues in terms of the variables analyzed.

**Exercise 5.8.** Perform an adequate statistical analysis based on shrinkage of a generalized linear model to reply the following questions:

a. 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?

b. 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 4.4 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 4.4: `biglm::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”.[44] The function that we will employ is `ffbase::bigglm.ffdf`, and requires an object of the class `ffdf` (`ff`’s data frames).

> The latest version of `ffbase` has a bug in `ffbase::bigglm.ffdf` that is reported in this GitHub issue. Until that issue is solved, you will not be able to apply `ffbase::bigglm.ffdf` with the latest package versions. A temporary workaround is to downgrade the packages `bit`, `ff`, and `ffbase` to the respective versions `1.1-15.2`, `2.2-14.2`, and `0.12.8`. The next two chunk of code provides this fix.

```r
# To install specific versions of packages
install.packages("versions")
library(versions)

# Install specific package versions. It may take a while to do so, be patient
install.versions(pkgs = c("bit", "ff", "ffbase"),
                 versions = c("1.1-15.2", "2.2-14.2", "0.12.8"))
# After bit's version 1.1-15.2, something is off in the integration with
# ffbase; see issue in https://github.com/edwindj/ffbase/issues/61

# Alternatively, if the binaries for your OS are not available (e.g., for
# Apple M1's processors), then you will need to compile the packages from
# source... and cross your fingers!
```

[44] The `ff` package implements the `ff` vectors and `ffdf` data frames classes. The package `ffbase` provides convenience functions for working with these non-standard classes in a more transparent way.

```r
urls <- c(
  "https://cran.r-project.org/src/contrib/Archive/bit/bit_1.1-15.2.tar.gz",
  "https://cran.r-project.org/src/contrib/Archive/ff/ff_2.2-14.2.tar.gz",
  "https://cran.r-project.org/src/contrib/Archive/ffbase/ffbase_0.12.8.tar.gz"
  )
install.packages(pkgs = urls, repos = NULL, type = "source")
```

After the packages `bit`, `ff`, and `ffbase` have been downgraded (or if the GitHub issue has been fixed), then you will be able to run the following code:

```r
# 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.ffdf(file = "bigData1.csv", header = TRUE, sep = ",")
bigData2ff <- read.table.ffdf(file = "bigData2.csv", header = TRUE, sep = ",")

# Recall: bigData1.csv is not copied into RAM
print(object.size(bigData1), units = "MB")
print(object.size(bigData1ff), units = "KB")

# Logistic regression
# Same comments for the formula framework -- this is the hack for automatic
# inclusion of all the predictors
library(biglm)
f <- formula(paste("resp ~", paste(names(bigData1)[-1], collapse = " + ")))
bigglmMod <- bigglm.ffdf(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")
print(object.size(glmMod), units = "MB")

# Summaries
s1 <- summary(bigglmMod)
s2 <- summary(glmMod)
s1
s2

# Further information
s1$mat # Coefficients and their inferences
s1$rsq # R^2
s1$nullrss # Null deviance

# Extract coefficients
coef(bigglmMod)

# Prediction works as usual
predict(bigglmMod, newdata = bigData2[1:5, ], type = "response")
```

```
# predict(bigglmMod, newdata = bigData2[1:5, -1]) # Error

# Update the model with training data
update(bigglmMod, moredata = bigData2)

# AIC and BIC
AIC(bigglmMod, k = 2)
AIC(bigglmMod, k = log(n))

# Delete the files in disk
file.remove(c("bigData1.csv", "bigData2.csv"))
```

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 `biglm::bigglm` models is not so straightforward. The trick that `leaps::regsubsets` employs for simplifying the model search in linear models (see Section 4.4) does not apply for generalized linear models because of the nonlinearity of the likelihood. However, there is a simple and useful hack: we can do **best subset selection** in the **linear model associated with the last iteration of the IRLS algorithm** and then refine the search by computing the exact BIC/AIC from a set of candidate models.[45] If we do so, we translate the model selection problem back to the linear case, plus an extra overhead of fitting several generalized linear models. Keep in mind that, albeit useful, this approach is a hacky approximation to the task of finding the best subset of predictors.

[45] Without actually expanding that list, as coming up with this list of candidate models is the most expensive part in best subset selection.

```
# Model selection adapted to big data generalized linear models
reg <- leaps::regsubsets(bigglmMod, nvmax = p + 1, method = "exhaustive")
# This takes the QR decomposition, which encodes the linear model associated with
# 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$which
subs$bic
subs$which[which.min(subs$bic), ]

# 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")

# Pearson correlation
cor(subs$bic, exactBICs, method = "pearson")
```

```r
# Order correlation
cor(subs$bic, exactBICs, method = "spearman")

# Both give the same model selection and same order
subs$which[which.min(subs$bic), ] # Approximate
subs$which[which.min(exactBICs), ] # Exact
```

# 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*.[1] *If* this assumption truly holds, then parametric methods are the best approach for estimating $m$. But 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 on a case-by-case basis. Avoiding this nuisance is the strongest point of **nonparametric** methods: they do not assume major hard-to-satisfy hypotheses on the regression function, but just minimal assumptions, which makes them directly employable. Their weak points are that they usually are more computationally demanding and are harder to interpret.

We consider first the simplest situation:[2] a single *continuous* predictor $X$ for predicting a response $Y$. In this case, recall that 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 $Y \mid X = x$, it is also a challenging task to estimate it: for each $x$ we have to estimate a *different curve*! A simpler approach, yet still challenging, is to estimate the conditional mean (a scalar) for each $x$ through the regression function

$$m(x) = \mathbb{E}[Y \mid X = x] = \int y f_{Y|X=x}(y)\,\mathrm{d}y.$$

As we will see, this density-based view of the regression function is useful in order to motivate estimators.

## 6.1 Nonparametric density estimation

In order to introduce a nonparametric estimator for the regression function $m$, we need to introduce first a nonparametric estimator for the *density* of the predictor $X$. This estimator is aimed to estimate $f$, the density of $X$, from a sample $X_1, \ldots, X_n$ without assuming any specific form for $f$. This is, without assuming, e.g., that the data is normally distributed.

[1] For example, linear models assume that $m$ is of the form $m(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$ for some unknown coefficients $\boldsymbol{\beta}$.

[2] For the sake of introducing the main concepts, on Section 6.3 we will see the full general situation.

### 6.1.1 Histogram and moving histogram

The simplest method to estimate a density $f$ from an iid sample $X_1, \ldots, X_n$ is the **histogram**. From an analytical point of view, the idea is to aggregate the data in intervals of the form $[x_0, x_0 + h)$ and then use their relative frequency to approximate the density at $x \in [x_0, x_0 + h)$, $f(x)$, by the estimate of

$$\begin{aligned} f(x_0) &= F'(x_0) \\ &= \lim_{h \to 0^+} \frac{F(x_0+h) - F(x_0)}{h} \\ &= \lim_{h \to 0^+} \frac{\mathbb{P}[x_0 < X < x_0 + h]}{h}. \end{aligned}$$

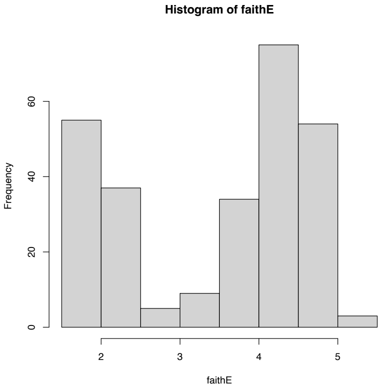
More precisely, given an *origin* $t_0$ and a *bandwidth* $h > 0$, the histogram builds a piecewise constant function in the intervals $\{B_k := [t_k, t_{k+1}) : t_k = t_0 + hk, k \in \mathbb{Z}\}$ by counting the number of sample points inside each of them. These constant-length intervals are also called *bins*. The fact they have constant length $h$ is important, since it allows us to standardize by $h$ in order to have relative frequencies *per length*[3] in the bins. The histogram at a point $x$ is defined as

$$\hat{f}_\mathrm{H}(x; t_0, h) := \frac{1}{nh} \sum_{i=1}^n 1_{\{X_i \in B_k : x \in B_k\}}. \qquad (6.1)$$

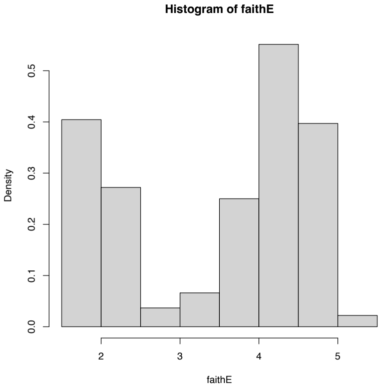
Equivalently, if we denote the number of points in $B_k$ as $v_k$, then the histogram is $\hat{f}_\mathrm{H}(x; t_0, h) = \frac{v_k}{nh}$ if $x \in B_k$ for $k \in \mathbb{Z}$.

The computation of histograms is straightforward in R. As an example, we consider the `faithful` dataset, which contains the duration of the eruption and the waiting time between eruptions for the Old Faithful geyser in Yellowstone National Park (USA).

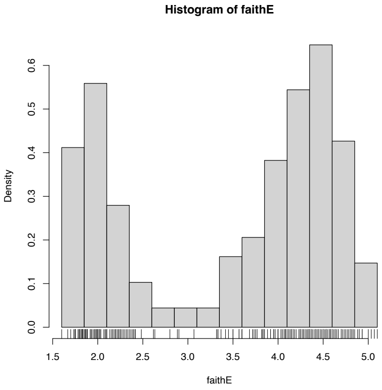
```r
# Duration of eruption
faithE <- faithful$eruptions

# Default histogram: automatically chooses bins and uses absolute frequencies
histo <- hist(faithE)
```



```r
# Bins and bin counts
histo$breaks # Bk's
## [1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
histo$counts # vk's
## [1] 55 37  5  9 34 75 54  3

# With relative frequencies
hist(faithE, probability = TRUE)
```



```r
# Choosing the breaks
t0 <- min(faithE)
h <- 0.25
Bk <- seq(t0, max(faithE), by = h)
hist(faithE, probability = TRUE, breaks = Bk)
rug(faithE) # The sample
```



Recall that the shape of the histogram depends on:

[3] Recall that with this standardization we approach to the probability *density* concept.

- $t_0$, since the separation between bins happens at $t_0k$, $k \in \mathbb{Z}$;
- $h$, which controls the bin size and the effective number of bins for aggregating the sample.

We focus first on exploring the dependence on $t_0$ with the next example, as it serves for motivating the next density estimator.

```
# Uniform sample
set.seed(1234567)
u <- runif(n = 100)

# t0 = 0, h = 0.2
Bk1 <- seq(0, 1, by = 0.2)

# t0 = -0.1, h = 0.2
Bk2 <- seq(-0.1, 1.1, by = 0.2)

# Comparison
par(mfrow = 1:2)
hist(u, probability = TRUE, breaks = Bk1, ylim = c(0, 1.5),
     main = "t0 = 0, h = 0.2")
rug(u)
abline(h = 1, col = 2)
hist(u, probability = TRUE, breaks = Bk2, ylim = c(0, 1.5),
     main = "t0 = -0.1, h = 0.2")
rug(u)
abline(h = 1, col = 2)
```

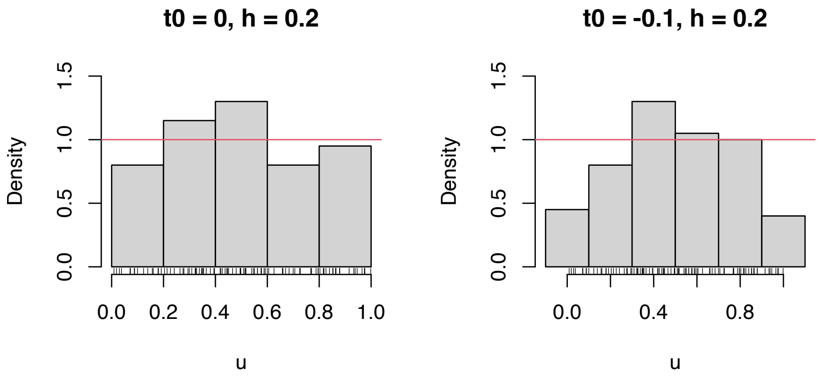


Figure 6.1: The dependence of the histogram on the origin $t_0$.

Clearly, this dependence is undesirable, as it is prone to change notably the estimation of $f$ using the same data. An alternative to avoid the dependence on $t_0$ is the **moving histogram** or **naive density estimator**. The idea is to aggregate the sample $X_1, \ldots, X_n$ in intervals of the form $(x-h, x+h)$ and then use its relative frequency in $(x-h, x+h)$ to approximate the density at $x$, which can be written as

$$\begin{aligned} f(x) &= F'(x) \\ &= \lim_{h \to 0^+} \frac{F(x+h) - F(x-h)}{2h} \\ &= \lim_{h \to 0^+} \frac{\mathbb{P}[x-h < X < x+h]}{2h}. \end{aligned}$$

Recall the differences with the histogram: the intervals **depend on the evaluation point** $x$ and are centered about it. That allows to directly estimate $f(x)$ (without the proxy $f(x_0)$) by an estimate of the symmetric derivative.

Given a bandwidth $h > 0$, the naive density estimator builds a piecewise constant function by considering the relative frequency of $X_1, \ldots, X_n$ inside $(x-h, x+h)$:[4]

$$\hat{f}_{\mathrm{N}}(x;h) := \frac{1}{2nh}\sum_{i=1}^n 1_{\{x-h<X_i<x+h\}}. \tag{6.2}$$

The analysis of $\hat{f}_{\mathrm{N}}(x;h)$ as a random variable follows from realizing that

$$\sum_{i=1}^n 1_{\{x-h<X_i<x+h\}} \sim \mathrm{B}(n, p_{x,h})$$

where

$$p_{x,h} := \mathbb{P}[x-h < X < x+h] = F(x+h) - F(x-h).$$

Therefore, employing the bias and variance expressions of a binomial,[5] it follows:

$$\begin{aligned}\mathbb{E}[\hat{f}_{\mathrm{N}}(x;h)] &= \frac{F(x+h)-F(x-h)}{2h},\\ \mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{N}}(x;h)] &= \frac{F(x+h)-F(x-h)}{4nh^2}\\ &\quad - \frac{(F(x+h)-F(x-h))^2}{4nh^2}.\end{aligned}$$

These two results provide very interesting insights on the effect of $h$ on the moving histogram:

1. If $h \to 0$, then $\mathbb{E}[\hat{f}_{\mathrm{N}}(x;h)] \to f(x)$ and (6.2) is an *asymptotically unbiased* estimator of $f(x)$. However, if $h \to 0$, the variance explodes: $\mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{N}}(x;h)] \approx \frac{f(x)}{2nh} - \frac{f(x)^2}{n} \to \infty$.
2. If $h \to \infty$, then both $\mathbb{E}[\hat{f}_{\mathrm{N}}(x;h)] \to 0$ and $\mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{N}}(x;h)] \to 0$. Therefore, the variance shrinks to zero but the bias grows.
3. If $nh \to \infty$,[6] then the variance shrinks to zero. If, in addition, $h \to 0$, the bias also shrinks to zero. So *both* the bias and the variance are reduced if $n \to \infty$, $h \to 0$, and $nh \to \infty$, **simultaneously**.

The animation in Figure 6.1.1 illustrates the previous points and gives insight on how the performance of (6.2) varies with $h$.

Bias and variance for the moving histogram. The animation shows how for small bandwidths the bias of $\hat{f}_{\mathrm{N}}(x;h)$ on estimating $f(x)$ is small, but the variance is high, and how for large bandwidths the bias is large and the variance is small. The variance is represented by the asymptotic 95% confidence intervals for $\hat{f}_{\mathrm{N}}(x;h)$. Recall how the variance of $\hat{f}_{\mathrm{N}}(x;h)$ is (almost) proportional to $f(x)$. Application also available here.

The estimator (6.2) poses an interesting question:

[4] Note that the function has $2n$ discontinuities that are located at $X_i \pm h$.

[5] $\mathbb{E}[\mathrm{B}(N,p)] = Np$ and $\mathbb{V}\mathrm{ar}[\mathrm{B}(N,p)] = Np(1-p)$.

[6] Or, in other words, if $h^{-1}$ grows slower than $n$.

Why giving the same weight to all $X_1, \ldots, X_n$ in $(x-h, x+h)$ for estimating $f(x)$?

We are estimating $f(x) = F'(x)$ by estimating $\frac{F(x+h)-F(x-h)}{2h}$ through the relative frequency of $X_1, \ldots, X_n$ in the interval $(x-h, x+h)$. Should not be the **data points closer to $x$ more important** than the ones further away? The answer to this question shows that (6.2) is indeed a particular case of a wider class of density estimators.

### 6.1.2 Kernel density estimation

The moving histogram (6.2) can be equivalently written as

$$\begin{aligned}\hat{f}_{\mathrm{N}}(x;h) &= \frac{1}{nh}\sum_{i=1}^{n}\frac{1}{2}1_{\left\{-1<\frac{x-X_i}{h}<1\right\}} \\ &= \frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{x-X_i}{h}\right), \qquad (6.3)\end{aligned}$$

with $K(z) = \frac{1}{2}1_{\{-1<z<1\}}$. Interestingly, $K$ is a uniform *density* in $(-1,1)$. This means that, when approximating

$$\mathbb{P}[x-h<X<x+h] = \mathbb{P}\left[-1<\frac{x-X}{h}<1\right]$$

by (6.3), we give equal weight to all the points $X_1, \ldots, X_n$. The generalization of (6.3) is now obvious: replace $K$ by an arbitrary density. Then $K$ is known as a *kernel*: a density with certain regularity that is (typically) symmetric and unimodal at 0. This generalization provides the definition of **kernel density estimator**[7] (kde):

$$\hat{f}(x;h) := \frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{x-X_i}{h}\right). \qquad (6.4)$$

A common notation is $K_h(z) := \frac{1}{h}K\left(\frac{z}{h}\right)$, so the kde can be compactly written as $\hat{f}(x;h) = \frac{1}{n}\sum_{i=1}^{n}K_h(x-X_i)$.

It is useful to recall (6.4) with the normal kernel. If that is the case, then $K_h(x-X_i) = \phi(x;X_i,h^2) = \phi(x-X_i;0,h^2)$ (denoted simply as $\phi(x-X_i;h^2)$) and the kernel is the density of a $\mathcal{N}(X_i,h^2)$. Thus the bandwidth $h$ can be thought of as the *standard deviation* of a normal density whose mean is $X_i$ and the kde (6.4) as a data-driven mixture of those densities. Figure 6.1.2 illustrates the construction of the kde, 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 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.[8] The *rectangular kernel* $K(z) = \frac{1}{2}1_{\{|z|<1\}}$ yields the moving histogram as a particular case. The kde inherits the smoothness properties of the kernel. That means,

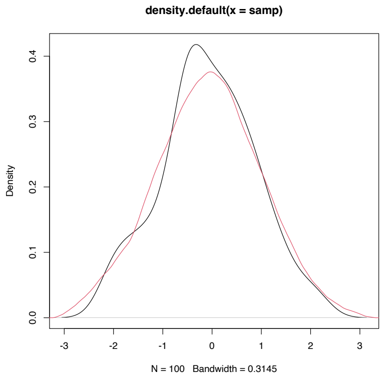
[7] Also known as the *Parzen–Rosenblatt estimator* to honor the proposals by Parzen (1962) and Rosenblatt (1956).

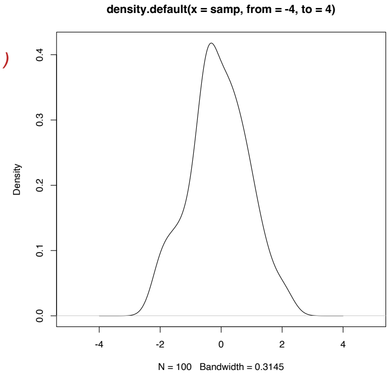
[8] Although the efficiency of the normal kernel, with respect to the Epanechnikov kernel, is roughly 0.95.

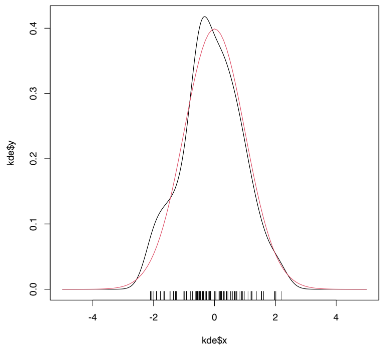
e.g., that (6.4) with a normal kernel is infinitely differentiable. But, with an Epanechnikov kernel, (6.4) is not differentiable, and, with a rectangular kernel, the kde is not even continuous. However, if a certain smoothness is guaranteed (continuity at least), then the *choice of the kernel has little importance in practice* (at least compared with the choice of the bandwidth $h$).

The computation of the kde in R is done through the `density` function. The function automatically chooses the bandwidth $h$ using a data-driven criterion.[9]

```r
# 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() 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))


# 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] 4.91e-17 2.97e-17 1.86e-17 2.53e-17 3.43e-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). This is because, internally,
# kde employs an efficient Fast Fourier Transform on grids of size 2^m

# Plotting the returned values of the kde
plot(kde$x, kde$y, type = "l")
curve(dnorm(x), col = 2, add = TRUE) # True density
rug(samp)
```







**Exercise 6.1.** Load the dataset `faithful`. Then:

a. Estimate and plot the density of `faithful$eruptions`.
b. Create a new plot and superimpose different density estimations with bandwidths equal to 0.1, 0.5, and 1.
c. Get the density estimate at *exactly* the point $x = 3.1$ using $h = 0.15$ and the Epanechnikov kernel.

### 6.1.3 *Bandwidth selection*

The kde critically depends on the employed bandwidth; hence objective and automatic bandwidth selectors that attempt to minimize

[9] Precisely, the *rule-of-thumb* given by `bw.nrd`.

the estimation error of the target density $f$ are required to properly apply a kde in practice.

A global, rather than local, error criterion for the kde is the *Integrated Squared Error* (ISE):

$$\mathrm{ISE}[\hat{f}(\cdot;h)] := \int (\hat{f}(x;h) - f(x))^2 \,\mathrm{d}x.$$

The ISE is a random quantity, since it depends directly on the sample $X_1, \ldots, X_n$. As a consequence, looking for an optimal-ISE bandwidth is a hard task, since the optimality is dependent on the sample itself (not only on $f$ and $n$). To avoid this problematic, it is usual to compute the *Mean Integrated Squared Error* (MISE):

$$\begin{aligned}\mathrm{MISE}[\hat{f}(\cdot;h)] &:= \mathbb{E}\left[\mathrm{ISE}[\hat{f}(\cdot;h)]\right]\\ &= \int \mathbb{E}\left[(\hat{f}(x;h) - f(x))^2\right] \mathrm{d}x\\ &= \int \mathrm{MSE}[\hat{f}(x;h)]\,\mathrm{d}x.\end{aligned}$$

Once the MISE is set as the error criterion to be minimized, our aim is to find

$$h_{\mathrm{MISE}} := \arg\min_{h>0} \mathrm{MISE}[\hat{f}(\cdot;h)].$$

For that purpose, we need an explicit expression of the MISE that we can attempt to minimize. An asymptotic expansion can be derived when $h \to 0$ and $nh \to \infty$, resulting in

$$\mathrm{MISE}[\hat{f}(\cdot;h)] \approx \mathrm{AMISE}[\hat{f}(\cdot;h)] := \frac{1}{4}\mu_2^2(K)R(f'')h^4 + \frac{R(K)}{nh}, \quad (6.5)$$

where $\mu_2(K) := \int z^2 K(z)\,\mathrm{d}z$ and $R(g) := \int g(x)^2\,\mathrm{d}x$. The AMISE stands for *Asymptotic MISE* and, due to its closed expression, it allows to obtain a bandwidth that minimizes it:[10]

$$h_{\mathrm{AMISE}} = \left[\frac{R(K)}{\mu_2^2(K)R(f'')n}\right]^{1/5}. \quad (6.6)$$

Unfortunately, the AMISE bandwidth depends on $R(f'') = \int (f''(x))^2\,\mathrm{d}x$, which measures the *curvature* of the **unknown** density $f$. As a consequence, it cannot be readily applied in practice!

**Plug-in selectors**

A simple solution to turn (6.6) into something computable is to estimate $R(f'')$ by *assuming* that $f$ is the density of a $\mathcal{N}(\mu,\sigma^2)$, and then plug-in the form of the curvature for such density:

$$R(\phi''(\cdot;\mu,\sigma^2)) = \frac{3}{8\pi^{1/2}\sigma^5}.$$

While doing so, we approximate the curvature of an arbitrary density by means of the curvature of a normal and we have that

$$h_{\mathrm{AMISE}} = \left[\frac{8\pi^{1/2}R(K)}{3\mu_2^2(K)n}\right]^{1/5}\sigma.$$

[10] By solving $\frac{\mathrm{d}}{\mathrm{d}h}\mathrm{AMISE}[\hat{f}(\cdot;h)] = 0$, i.e., $\mu_2^2(K)R(f'')h^3 - R(K)n^{-1}h^{-2} = 0$, yields $h_{\mathrm{AMISE}}$.

Interestingly, the bandwidth is directly proportional to the standard deviation of the target density. Replacing $\sigma$ by an estimate yields the **normal scale bandwidth selector**, which we denote by $\hat{h}_{\mathrm{NS}}$ to emphasize its randomness:

$$\hat{h}_{\mathrm{NS}} = \left[\frac{8\pi^{1/2}R(K)}{3\mu_2^2(K)n}\right]^{1/5}\hat{\sigma}.$$

The estimate $\hat{\sigma}$ can be chosen as the standard deviation $s$, or, in order to avoid the effects of potential outliers, as the standardized interquantile range

$$\hat{\sigma}_{\mathrm{IQR}} := \frac{X_{([0.75n])} - X_{([0.25n])}}{\Phi^{-1}(0.75) - \Phi^{-1}(0.25)}$$

or as

$$\hat{\sigma} = \min(s, \hat{\sigma}_{\mathrm{IQR}}). \tag{6.7}$$

When combined with a normal kernel, for which $\mu_2(K) = 1$ and $R(K) = \frac{1}{2\sqrt{\pi}}$, this particularization of $\hat{h}_{\mathrm{NS}}$ gives the famous **rule-of-thumb** for bandwidth selection:

$$\hat{h}_{\mathrm{RT}} = \left(\frac{4}{3}\right)^{1/5} n^{-1/5}\hat{\sigma} \approx 1.06n^{-1/5}\hat{\sigma}.$$

$\hat{h}_{\mathrm{RT}}$ is implemented in R through the function `bw.nrd`.[11]

```r
# Data
set.seed(667478)
n <- 100
x <- rnorm(n)

# Rule-of-thumb
bw.nrd(x = x)
## [1] 0.4040319
# bwd.nrd employs 1.34 as an approximation for diff(qnorm(c(0.25, 0.75)))

# Same as
iqr <- diff(quantile(x, c(0.25, 0.75))) / diff(qnorm(c(0.25, 0.75)))
1.06 * n^(-1/5) * min(sd(x), iqr)
## [1] 0.4040319
```

The rule-of-thumb is an example of a **zero-stage plug-in** selector, a terminology which lies on the fact that $R(f'')$ was estimated by plugging-in a parametric estimation at *"the very first moment a quantity that depends on $f$ appears"*. We could have opted to estimate $R(f'')$ nonparametrically, in a certain optimal way, and then plug-in the estimate into $h_{\mathrm{AMISE}}$. The important catch lies on the optimal estimation of $R(f'')$ : it requires the knowledge of $R\big(f^{(4)}\big)$! What $\ell$-stage plug-in selectors do is to iterate these steps $\ell$ times and finally plug-in a normal estimate of the unknown $R\big(f^{(2\ell)}\big)$.[12]

Typically, **two stages** are considered a good trade-off between bias (mitigated when $\ell$ increases) and variance (augments with $\ell$) of the plug-in selector. This is the method proposed by Sheather and Jones (1991), yielding what we call the **Direct Plug-In** (DPI). The DPI selector is implemented in R through the function `bw.SJ`

[11] Not to confuse with `bw.nrd0`!

[12] The motivation for doing so is to try to add the parametric assumption at a later, less important, step.

(use `method = "dpi"`). An alternative and faster implementation is `ks::hpi`, which also provides more flexibility and has a somewhat more detailed documentation.

```
# Data
set.seed(672641)
x <- rnorm(100)

# DPI selector
bw.SJ(x = x, method = "dpi")
## [1] 0.5006905

# Similar to
ks::hpi(x) # Default is two-stages
## [1] 0.4999456
```

### Cross-validation

We turn now our attention to a different philosophy of bandwidth estimation. Instead of trying to minimize the AMISE by plugging-in estimates for the unknown curvature term, we directly attempt to minimize the MISE by using the sample twice: one for computing the kde and other for *evaluating* its performance on estimating $f$. To avoid the clear dependence on the sample, we do the evaluation in a *cross-validatory* way: the data used for computing the kde is *not* used for its evaluation.

We begin by expanding the square in the MISE expression:

$$\begin{aligned}\mathrm{MISE}[\hat{f}(\cdot;h)] =& \mathbb{E}\left[\int (\hat{f}(x;h)-f(x))^2\,\mathrm{d}x\right]\\ =& \mathbb{E}\left[\int \hat{f}(x;h)^2\,\mathrm{d}x\right] - 2\mathbb{E}\left[\int \hat{f}(x;h)f(x)\,\mathrm{d}x\right]\\ &+\int f(x)^2\,\mathrm{d}x.\end{aligned}$$

Since the last term does not depend on $h$, minimizing $\mathrm{MISE}[\hat{f}(\cdot;h)]$ is equivalent to minimizing

$$\mathbb{E}\left[\int \hat{f}(x;h)^2\,\mathrm{d}x\right] - 2\mathbb{E}\left[\int \hat{f}(x;h)f(x)\,\mathrm{d}x\right].$$

This quantity is unknown, but it can be estimated unbiasedly by

$$\mathrm{LSCV}(h) := \int \hat{f}(x;h)^2\,\mathrm{d}x - 2n^{-1}\sum_{i=1}^n \hat{f}_{-i}(X_i;h), \tag{6.8}$$

where $\hat{f}_{-i}(\cdot;h)$ is the *leave-one-out* kde and is based on the sample with the $X_i$ removed:

$$\hat{f}_{-i}(x;h) = \frac{1}{n-1}\sum_{\substack{j=1\\ j\neq i}}^n K_h(x-X_j).$$

The **Least Squares Cross-Validation** (LSCV) selector, also denoted **Unbiased Cross-Validation** (UCV) selector, is defined as

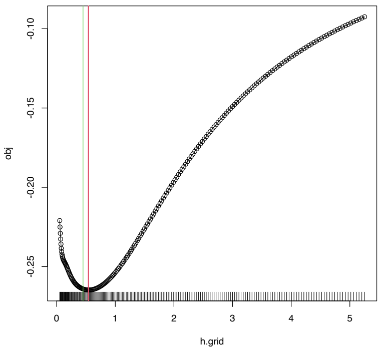
$$\hat{h}_{\mathrm{LSCV}} := \arg\min_{h>0} \mathrm{LSCV}(h).$$

Numerical optimization is required for obtaining $\hat{h}_{\mathrm{LSCV}}$, contrary to the previous plug-in selectors, and there is little control on the shape of the objective function.

⚠ Numerical optimization of the (6.8) can be challenging. In practice, several local minima are possible, and the roughness of the objective function can vary notably depending on $n$ and $f$. As a consequence, **optimization routines may get trapped in spurious solutions**. To be on the safe side, it is advisable to check the solution by plotting $\mathrm{LSCV}(h)$ for a range of $h$, or to perform an exhaustive search in a bandwidth grid: $\hat{h}_{\mathrm{LSCV}} \approx \arg\min_{h_1,\ldots,h_G} \mathrm{LSCV}(h)$.

$\hat{h}_{\mathrm{LSCV}}$ is implemented in R through the function `bw.ucv`. `bw.ucv` uses `optimize`, which is quite sensible to the selection of the search interval.[13] Therefore, some care is needed and that is why the `bw.ucv.mod` function is presented.

```
# Data
set.seed(123456)
x <- rnorm(100)

# UCV gives a warning
bw.ucv(x = x)
## [1] 0.4499177

# Extend search interval
bw.ucv(x = x, lower = 0.01, upper = 1)
## [1] 0.5482419

# bw.ucv.mod replaces the optimization routine of bw.ucv by an exhaustive
# search on "h.grid" (chosen adaptatively from the sample) and optionally
# plots the LSCV curve with "plot.cv"
bw.ucv.mod <- function(x, nb = 1000L,
                       h.grid = diff(range(x)) * (seq(0.1, 1, l = 200))^2,
                       plot.cv = FALSE) {
  if ((n <- length(x)) < 2L)
    stop("need at least 2 data points")
  n <- as.integer(n)
  if (is.na(n))
    stop("invalid length(x)")
  if (!is.numeric(x))
    stop("invalid 'x'")
  nb <- as.integer(nb)
  if (is.na(nb) || nb <= 0L)
    stop("invalid 'nb'")
  storage.mode(x) <- "double"
  hmax <- 1.144 * sqrt(var(x)) * n^(-1/5)
  Z <- .Call(stats:::C_bw_den, nb, x)
  d <- Z[[1L]]
  cnt <- Z[[2L]]
  fucv <- function(h) .Call(stats:::C_bw_ucv, n, d, cnt, h)
  ## Original
  # h <- optimize(fucv, c(lower, upper), tol = tol)$minimum
  # if (h < lower + tol | h > upper - tol)
  #   warning("minimum occurred at one end of the range")
  ## Modification
  obj <- sapply(h.grid, function(h) fucv(h))
  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
}

# Compute the bandwidth and plot the LSCV curve
bw.ucv.mod(x = x, plot.cv = TRUE)
## [1] 0.5431732

# We can compare with the default bw.ucv output
abline(v = bw.ucv(x = x), col = 3)
```



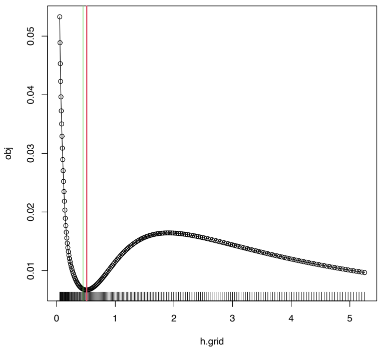
The next cross-validation selector is based on **Biased Cross-Validation** (BCV). The BCV selector presents a hybrid strategy that combines plug-in and cross-validation ideas. It starts by considering the AMISE expression in (6.5) and then plugs-in an estimate for $R(f'')$ based on a modification of $R(\hat{f}''(\cdot;h))$. The appealing

[13] Long intervals containing the solution may lead to unsatisfactory termination of the search; short intervals might not contain the minimum.

property of $\hat{h}_{\mathrm{BCV}}$ is that it has a considerably smaller variance compared to $\hat{h}_{\mathrm{LSCV}}$. This reduction in variance comes at the price of an increased bias, which tends to make $\hat{h}_{\mathrm{BCV}}$ larger than $h_{\mathrm{MISE}}$.

$\hat{h}_{\mathrm{BCV}}$ is implemented in R through the function `bw.bcv`. Again, `bw.bcv` uses `optimize` so the `bw.bcv.mod` function is presented to have better guarantees on finding the adequate minimum.[14]

```
# Data
set.seed(123456)
x <- rnorm(100)

# BCV gives a warning
bw.bcv(x = x)
## [1] 0.4500924

# Extend search interval
args(bw.bcv)
## function (x, nb = 1000L, lower = 0.1 * hmax, upper = hmax, tol = 0.1 *
##     lower)
## NULL
bw.bcv(x = x, lower = 0.01, upper = 1)
## [1] 0.5070129

# bw.bcv.mod replaces the optimization routine of bw.bcv by an exhaustive
# search on "h.grid" (chosen adaptatively from the sample) and optionally
# plots the BCV curve with "plot.cv"
bw.bcv.mod <- function(x, nb = 1000L,
                       h.grid = diff(range(x)) * (seq(0.1, 1, l = 200))^2,
                       plot.cv = FALSE) {
  if ((n <- length(x)) < 2L)
    stop("need at least 2 data points")
  n <- as.integer(n)
  if (is.na(n))
    stop("invalid length(x)")
  if (!is.numeric(x))
    stop("invalid 'x'")
  nb <- as.integer(nb)
  if (is.na(nb) || nb <= 0L)
    stop("invalid 'nb'")
  storage.mode(x) <- "double"
  hmax <- 1.144 * sqrt(var(x)) * n^(-1/5)
  Z <- .Call(stats:::C_bw_den, nb, x)
  d <- Z[[1L]]
  cnt <- Z[[2L]]
  fbcv <- function(h) .Call(stats:::C_bw_bcv, n, d, cnt, h)
  ## Original code
  # h <- optimize(fbcv, c(lower, upper), tol = tol)$minimum
  # if (h < lower + tol | h > upper - tol)
  #   warning("minimum occurred at one end of the range")
  ## Modification
  obj <- sapply(h.grid, function(h) fbcv(h))
  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
}

# Compute the bandwidth and plot the BCV curve
bw.bcv.mod(x = x, plot.cv = TRUE)
## [1] 0.5130493

# We can compare with the default bw.bcv output
abline(v = bw.bcv(x = x), col = 3)
```



[14] But be careful not expanding the upper limit of the search interval too much!

### Comparison of bandwidth selectors

Despite it is possible to compare theoretically the performance of bandwidth selectors by investigating the convergence of $n^\nu(\hat{h}/h_{\mathrm{MISE}} - 1)$, comparisons are usually done by simulation and investigation of the averaged ISE error. A popular collection of simulation scenarios was given by Marron and Wand (1992) and are conveniently available through the package `nor1mix`. They form a collection of normal $r$-mixtures of the form

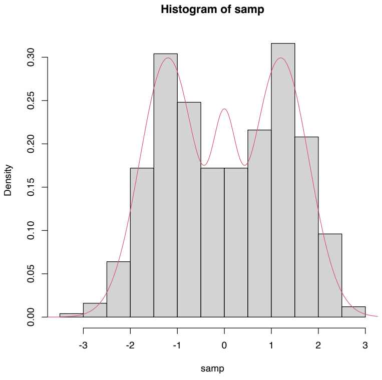
$$f(x; \boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{w}) := \sum_{j=1}^{r} w_j \phi(x; \mu_j, \sigma_j^2),$$

where $w_j \geq 0$, $j = 1, \ldots, r$ and $\sum_{j=1}^{r} w_j = 1$. Densities of this form are specially attractive since they allow for arbitrarily flexibility and, if the normal kernel is employed, they allow for **explicit and exact** MISE expressions, directly computable as

$$\begin{aligned}\mathrm{MISE}_r[\hat{f}(\cdot; h)] &= (2\sqrt{\pi} nh)^{-1} + \mathbf{w}^\top \{(1 - n^{-1})\boldsymbol{\Omega}_2 - 2\boldsymbol{\Omega}_1 + \boldsymbol{\Omega}_0\}\mathbf{w},\\ (\boldsymbol{\Omega}_a)_{ij} &:= \phi(\mu_i - \mu_j; ah^2 + \sigma_i^2 + \sigma_j^2), \quad i, j = 1, \ldots, r.\end{aligned}$$

This expression is especially useful for benchmarking bandwidth selectors, as the MISE optimal bandwidth can be computed by $h_{\mathrm{MISE}} = \arg\min_{h>0} \mathrm{MISE}_r[\hat{f}(\cdot; h)]$.

```r
# Available models
?nor1mix::MarronWand

# Simulating -- specify density with MW object
samp <- nor1mix::rnorMix(n = 500, obj = nor1mix::MW.nm9)
hist(samp, freq = FALSE)

# Density evaluation
x <- seq(-4, 4, length.out = 400)
lines(x, nor1mix::dnorMix(x = x, obj = nor1mix::MW.nm9), col = 2)
```



```r
# Plot a MW object directly
# A normal with the same mean and variance is plotted in dashed lines
par(mfrow = c(2, 2))
plot(nor1mix::MW.nm5)
plot(nor1mix::MW.nm7)
plot(nor1mix::MW.nm10)
plot(nor1mix::MW.nm12)
lines(nor1mix::MW.nm1, col = 2:3) # Also possible
```

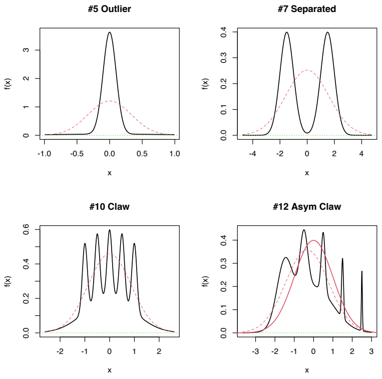


Figure 6.1.3 presents a visualization of the performance of the kde with different bandwidth selectors, carried out in the family of mixtures of Marron and Wand (1992).

Performance comparison of bandwidth selectors. The RT, DPI, LSCV, and BCV are computed for each sample for a normal mixture density. For each sample, computes the ISEs of the selectors and sorts them from best to worst. Changing the scenarios gives insight on the adequacy of each selector to hard- and simple-to-estimate densities. Application also available here.

**Which bandwidth selector is the most adequate for a given dataset?**

There is no simple and universal answer to this question. There are, however, a series of useful facts and suggestions:

- Trying several selectors and inspecting the results may help on determining which one is estimating the density better.
- The **DPI selector has a convergence rate much faster than the cross-validation selectors**. Therefore, in theory, it is expected to perform better than LSCV and BCV. For this reason, it tends to be amongst the **preferred bandwidth selectors** in the literature.
- Cross-validatory selectors may be better suited for highly non-normal and rough densities, in which plug-in selectors may end up oversmoothing.
- LSCV tends to be considerably more variable than BCV.
- The RT is a quick, simple, and inexpensive selector. However, it tends to give bandwidths that are too large for non-normal data.

### 6.1.4 Multivariate extension

Kernel density estimation can be extended to estimate multivariate densities $f$ in $\mathbb{R}^p$. For a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ in $\mathbb{R}^p$, the kde of $f$ evaluated at $\mathbf{x} \in \mathbb{R}^p$ is

$$\hat{f}(\mathbf{x}; \mathbf{H}) := \frac{1}{n|\mathbf{H}|^{1/2}} \sum_{i=1}^{n} K\left(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{X}_i)\right), \tag{6.9}$$

where $K$ is *multivariate kernel*, a $p$-variate density that is (typically) symmetric and unimodal at $\mathbf{0}$, and that depends on the *bandwidth matrix*[15] $\mathbf{H}$, a $p \times p$ symmetric and positive definite matrix. A common notation is $K_{\mathbf{H}}(\mathbf{z}) := |\mathbf{H}|^{-1/2} K\big(\mathbf{H}^{-1/2}\mathbf{z}\big)$, so the kde can be compactly written as $\hat{f}(\mathbf{x}; \mathbf{H}) := \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i)$. The most employed multivariate kernel is the normal kernel $K(\mathbf{z}) = \phi(\mathbf{z}; \mathbf{0}, \mathbf{I}_p)$.

The interpretation of (6.9) is analogous to the one of (6.4): build a mixture of densities with each density centered at each data point. As a consequence, and roughly speaking, most of the concepts and ideas seen in univariate kernel density estimation extend to the multivariate situation, although some of them with considerable technical complications. For example, bandwidth selection inherits the same cross-validatory ideas (LSCV and BCV selectors) and plug-in methods (NS and DPI) seen before, but with increased complexity for the BCV and DPI selectors. The interested reader is referred to Chacón and Duong (2018) for a rigorous and comprehensive treatment.

We briefly discuss next the NS and LSCV selectors, denoted by $\hat{\mathbf{H}}_{\mathrm{NS}}$ and $\hat{\mathbf{H}}_{\mathrm{LSCV}}$, respectively. The **normal scale bandwidth selector** follows, as in the univariate case, by minimizing the asymptotic MISE of the kde, which now takes the form

$$\mathrm{MISE}[\hat{f}(\cdot; \mathbf{H})] = \mathbb{E}\left[\int (\hat{f}(\mathbf{x}; \mathbf{H}) - f(\mathbf{x}))^2 \, \mathrm{d}\mathbf{x}\right],$$

and then assuming that $f$ is the pdf of a $\mathcal{N}_p\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$. With the normal kernel, this results in

$$\mathbf{H}_{\mathrm{NS}} = (4(p+2))^{2/(p+4)} n^{-2/(p+4)} \boldsymbol{\Sigma}. \tag{6.10}$$

Replacing $\boldsymbol{\Sigma}$ by the sample covariance matrix $\mathbf{S}$ in (6.10) gives $\hat{\mathbf{H}}_{\mathrm{NS}}$.

The **unbiased cross-validation** selector neatly extends from the univariate case and attempts to minimize $\mathrm{MISE}[\hat{f}(\cdot; \mathbf{H})]$ by estimating it unbiasedly with

$$\mathrm{LSCV}(\mathbf{H}) := \int \hat{f}(\mathbf{x}; \mathbf{H})^2 \, \mathrm{d}\mathbf{x} - 2n^{-1} \sum_{i=1}^{n} \hat{f}_{-i}(\mathbf{X}_i; \mathbf{H})$$

and then minimizing it:

$$\hat{\mathbf{H}}_{\mathrm{LSCV}} := \arg \min_{\mathbf{H} \in \mathrm{SPD}_p} \mathrm{LSCV}(\mathbf{H}), \tag{6.11}$$

where $\mathrm{SPD}_p$ is the set of positive definite matrices[16] of size $p$.

Considering a *full* bandwidth matrix $\mathbf{H}$ gives more flexibility to the kde, but also increases notably the amount of bandwidth

[15] Observe that, if $p = 1$, then $\mathbf{H}$ will equal the *square* of the bandwidth $h$, that is, $\mathbf{H} = h^2$.

[16] Observe that implementing the optimization of (6.11) is not trivial, since it is required to enforce the constraint $\mathbf{H} \in \mathrm{SPD}_p$. A neat way of parametrizing $\mathbf{H}$ that induces the positive definiteness constrain is through the (unique) Cholesky decomposition of $\mathbf{H} \in \mathrm{SPD}_p$: $\mathbf{H} = \mathbf{R}^\top \mathbf{R}$, where $\mathbf{R}$ is a *triangular* matrix with *positive* entries on the diagonal (but the remaining entries unconstrained). Therefore, optimization of (6.11) can be done through the $\frac{p(p+1)}{2}$ entries of $\mathbf{R}$.

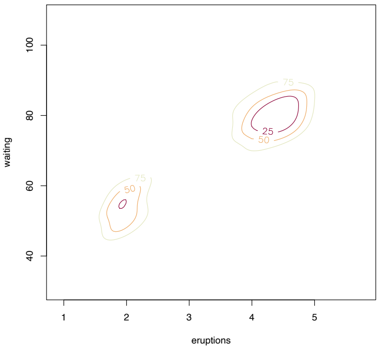
parameters that need to be chosen – precisely $\frac{p(p+1)}{2}$ – which significantly complicates bandwidth selection as the dimension $p$ grows. A common simplification is to consider a *diagonal* bandwidth matrix $\mathbf{H} = \mathrm{diag}(h_1^2, \ldots, h_p^2)$, which yields the kde employing *product kernels*:

$$\hat{f}(\mathbf{x}; \mathbf{h}) = \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x_1 - X_{i,1}) \times \overset{p}{\cdots} \times K_{h_p}(x_p - X_{i,p}), \qquad (6.12)$$

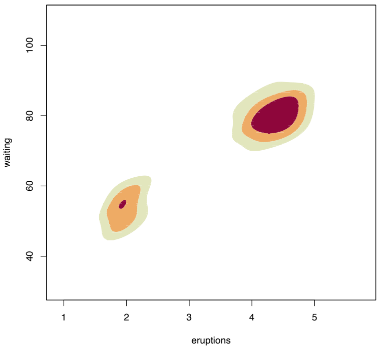
where $\mathbf{h} = (h_1, \ldots, h_p)^\top$ is the vector of bandwidths. If the variables $X_1, \ldots, X_p$ have been standardized (so that they have the same scale), then a simple choice is to consider $h = h_1 = \cdots = h_p$.

Multivariate kernel density estimation and bandwidth selection is not supported in base R, but the ks package implements the kde by `ks::kde` for $p \leq 6$. Bandwidth selectors, allowing for full or diagonal bandwidth matrices are implemented by: `ks::Hns` (NS), `ks::Hpi` and `ks::Hpi.diag` (DPI), `ks::Hlscv` and `ks::Hlscv.diag` (LSCV), and `ks::Hbcv` and `ks::Hbcv.diag` (BCV). The next chunk of code illustrates their usage with the `faithful` dataset.

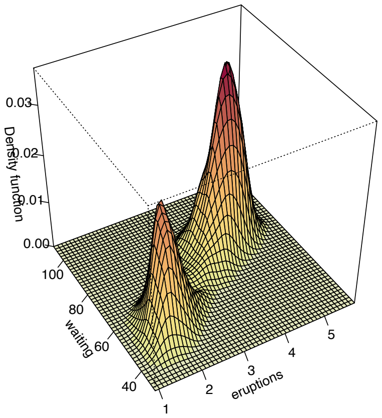
```r
# DPI selectors
Hpi1 <- ks::Hpi(x = faithful)
Hpi1
##            [,1]       [,2]
## [1,] 0.06326802  0.6041862
## [2,] 0.60418624 11.1917775

# Compute kde (if H is missing, ks::Hpi is called)
kdeHpi1 <- ks::kde(x = faithful, H = Hpi1)

# Different representations
plot(kdeHpi1, display = "slice", cont = c(25, 50, 75))
```



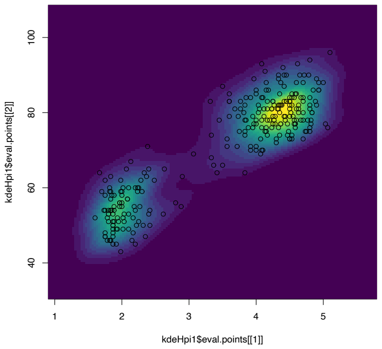
```r
# "cont" specifies the density contours, which are upper percentages of highest
# density regions. The default contours are at 25%, 50%, and 75%
plot(kdeHpi1, display = "filled.contour2", cont = c(25, 50, 75))
```



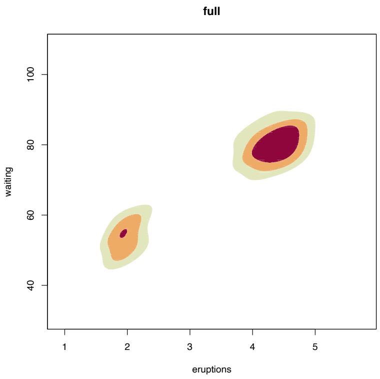
```r
plot(kdeHpi1, display = "persp")
```

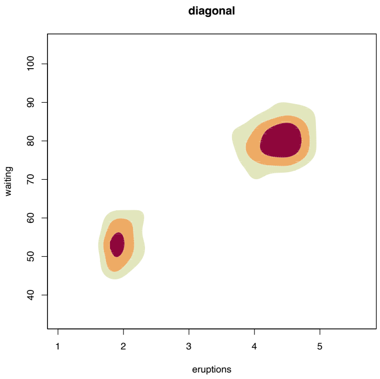


```r
# Manual plotting using the kde object structure
image(kdeHpi1$eval.points[[1]], kdeHpi1$eval.points[[2]],
      kdeHpi1$estimate, col = viridis::viridis(20))
points(kdeHpi1$x)
```

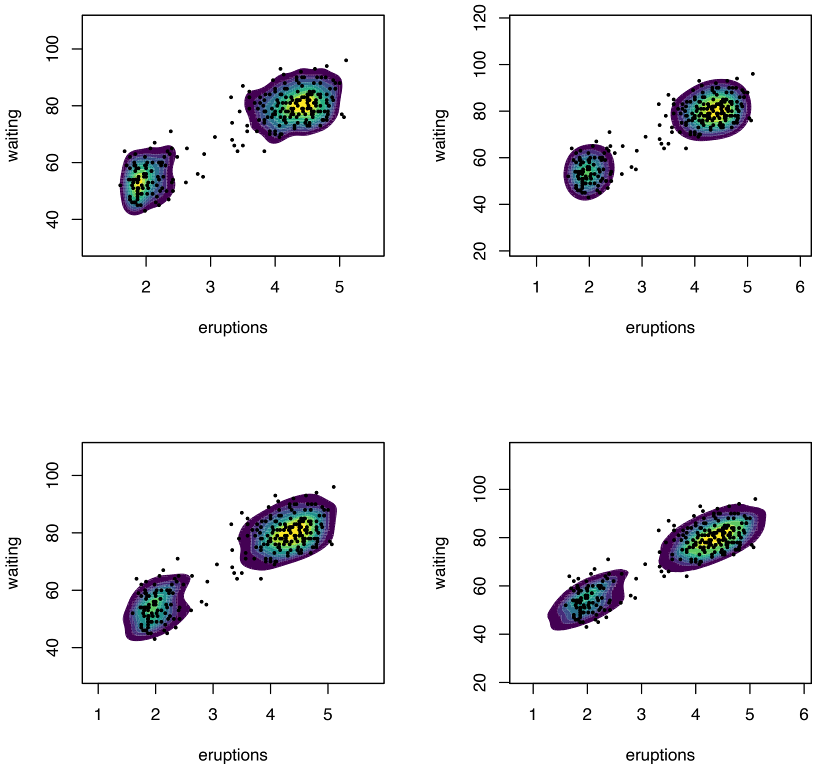


```r
# Diagonal vs. full
Hpi2 <- ks::Hpi.diag(x = faithful)
kdeHpi2 <- ks::kde(x = faithful, H = Hpi2)
plot(kdeHpi1, display = "filled.contour2", cont = c(25, 50, 75),
     main = "full")

plot(kdeHpi2, display = "filled.contour2", cont = c(25, 50, 75),
     main = "diagonal")
```





```
# Comparison of selectors along predefined contours
x <- faithful
Hlscv0 <- ks::Hlscv(x = x)
Hbcv0 <- ks::Hbcv(x = x)
Hpi0 <- ks::Hpi(x = x)
Hns0 <- ks::Hns(x = x)
par(mfrow = c(2, 2))
p <- lapply(list(Hlscv0, Hbcv0, Hpi0, Hns0), function(H) {
  # col.fun for custom colors
  plot(ks::kde(x = x, H = H), display = "filled.contour2",
       cont = seq(10, 90, by = 10), col.fun = viridis::viridis)
  points(x, cex = 0.5, pch = 16)
})
```



Kernel density estimation can be used to visualize density level sets in 3D too, as illustrated as follows with the iris dataset.

```
# Normal scale bandwidth
Hns1 <- ks::Hns(iris[, 1:3])

# Show high nested contours of high density regions
plot(ks::kde(x = iris[, 1:3], H = Hns1))
rgl::points3d(x = iris[, 1:3])
rgl::rglwidget()
```

**Exercise 6.2.** Consider the normal mixture

$$w_1\mathcal{N}_2(\mu_{11},\mu_{12},\sigma_{11}^2,\sigma_{12}^2,\rho_1)+w_2\mathcal{N}_2(\mu_{21},\mu_{22},\sigma_{21}^2,\sigma_{22}^2,\rho_2),$$

where $w_1=0.3$, $w_2=0.7$, $(\mu_{11},\mu_{12})=(1,1)$, $(\mu_{21},\mu_{22})=(-1,-1)$, $\sigma_{11}^2=\sigma_{21}^2=1$, $\sigma_{12}^2=\sigma_{22}^2=2$, $\rho_1=0.5$, and $\rho_2=-0.5$.

Perform the following simulation exercise:

1. Plot the density of the mixture using `ks::dnorm.mixt` and overlay points simulated employing `ks::rnorm.mixt`. You may want to use `ks::contourLevels` to have density plots comparable to the kde plots performed in the next step.
2. Compute the kde employing $\hat{\mathbf{H}}_{\mathrm{DPI}}$, both for full and diagonal bandwidth matrices. Are there any gains on considering full bandwidths? What if $\rho_2=0.7$?
3. Consider the previous point with $\hat{\mathbf{H}}_{\mathrm{LSCV}}$ instead of $\hat{\mathbf{H}}_{\mathrm{DPI}}$. Are the conclusions the same?

## 6.2 *Kernel regression estimation*

### 6.2.1 *Nadaraya–Watson estimator*

Our objective is to estimate the regression function $m:\mathbb{R}^p\to\mathbb{R}$ nonparametrically (recall that we are considering the simplest situation: one continuous predictor, so $p=1$). Due to its definition, we can rewrite $m$ as

$$\begin{aligned}m(x)=&\,\mathbb{E}[Y\mid X=x]\\=&\int yf_{Y|X=x}(y)\,\mathrm{d}y\\=&\,\frac{\int yf(x,y)\,\mathrm{d}y}{f_X(x)}.\end{aligned}\tag{6.13}$$

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 $\{(X_i,Y_i)\}_{i=1}^n$, a nonparametric estimate of $m$ may follow by replacing the previous densities by their kernel density estimators! From the previous section, we know how to do this using the multivariate and univariate kde's given in (6.4) and (6.9), respectively. For the multivariate kde, we can consider the kde (6.12) based on product kernels for the two dimensional case and bandwidths $\mathbf{h}=(h_1,h_2)^\top$, which yields the estimate

$$\hat{f}(x,y;\mathbf{h})=\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)K_{h_2}(y-Y_i)\tag{6.14}$$

of the joint pdf of $(X,Y)$. On the other hand, considering the same bandwidth $h_1$ for the kde of $f_X$, we have

$$\hat{f}_X(x;h_1)=\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i).\tag{6.15}$$

We can therefore define the estimator of $m$ that results from replacing $f$ and $f_X$ in (6.13) by (6.14) and (6.15):

$$\begin{aligned}\frac{\int y\hat{f}(x,y;\mathbf{h})\,\mathrm{d}y}{\hat{f}_X(x;h_1)} &= \frac{\int y\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)K_{h_2}(y-Y_i)\,\mathrm{d}y}{\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)}\\ &= \frac{\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)\int yK_{h_2}(y-Y_i)\,\mathrm{d}y}{\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)}\\ &= \frac{\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)Y_i}{\frac{1}{n}\sum_{i=1}^n K_{h_1}(x-X_i)}\\ &= \sum_{i=1}^n \frac{K_{h_1}(x-X_i)}{\sum_{i=1}^n K_{h_1}(x-X_i)}Y_i.\end{aligned}$$

The resulting estimator[17] is the so-called **Nadaraya–Watson**[18] estimator of the regression function:

$$\hat{m}(x;0,h) := \sum_{i=1}^n \frac{K_h(x-X_i)}{\sum_{i=1}^n K_h(x-X_i)}Y_i = \sum_{i=1}^n W_i^0(x)Y_i, \tag{6.16}$$

where

$$W_i^0(x) := \frac{K_h(x-X_i)}{\sum_{j=1}^n K_h(x-X_j)}.$$

The Nadaraya–Watson estimator can be seen as a **weighted average** of $Y_1,\ldots,Y_n$ by means of the set of weights $\{W_i(x)\}_{i=1}^n$ (they add to one). The set of *varying* weights depends on the evaluation point $x$. That means that the Nadaraya–Watson estimator is a **local mean of** $Y_1,\ldots,Y_n$ **about** $X=x$ (see Figure 6.2.2).

Let's implement from scratch the Nadaraya–Watson estimate to get a feeling of how it works in practice.

```r
# A naive implementation of the Nadaraya-Watson estimator
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 length(x) x n
  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
```

[17] Notice that it does *not* depend on $h_2$, only on $h_1$, the bandwidth employed for smoothing $X$.

[18] Termed due to the contemporaneous proposals by Nadaraya (1964) and Watson (1964).

```
eps <- rnorm(n, sd = 2)
m <- function(x) x^2 * cos(x)
# m <- function(x) x - x^2 # Other possible regression function, works
# equally well
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("top", legend = c("True regression", "Nadaraya-Watson"),
       lwd = 2, col = 1:2)
```

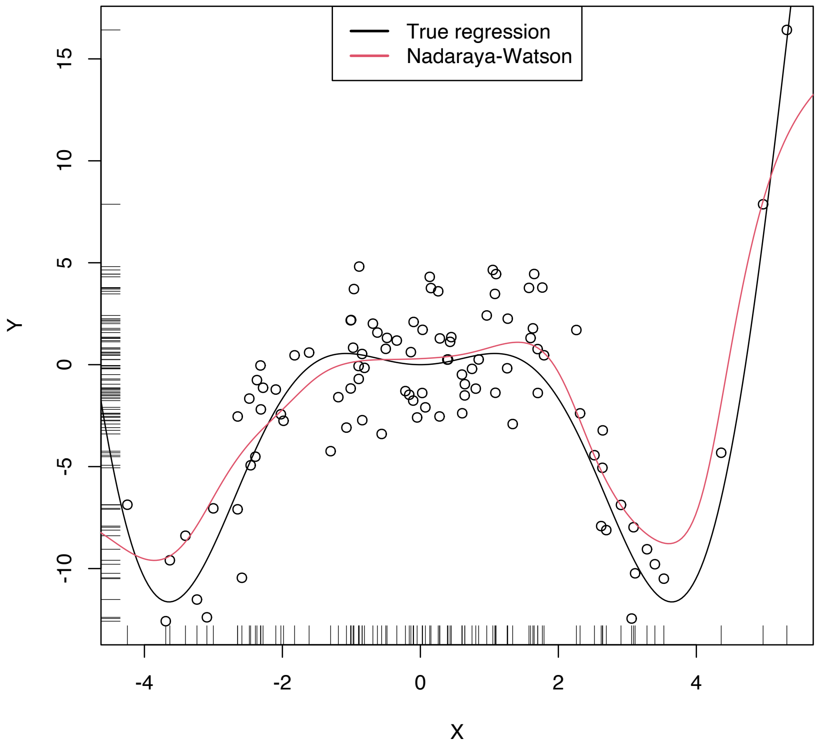


Figure 6.2: The Nadaraya–Watson estimator of an arbitrary regression function $m$.

Similarly to kernel density estimation, in the Nadaraya–Watson estimator the bandwidth has a prominent effect on the shape of the estimator, whereas the kernel is clearly less important. The code below illustrates the effect of varying $h$ using the `manipulate::manipulate` function.

```
# Simple plot of N-W for varying h's
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 = manipulate::slider(min = 0.01, max = 2, initial = 0.5, step = 0.01))
```

**Exercise 6.3.** Implement your own version of the Nadaraya–Watson estimator in R and compare it with `mNW`. 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 the `microbenchmark::microbenchmark` function to measure the running times for a sample with $n = 10000$.

### 6.2.2 *Local polynomial regression*

The Nadaraya–Watson estimator can be seen as a particular case of a wider class of nonparametric estimators, the so called *local polynomial estimators*. Specifically, Nadaraya–Watson corresponds to performing a *local constant fit*. Let's see this wider class of nonparametric estimators and their advantages with respect to the Nadaraya–Watson estimator.

The motivation for the local polynomial fit comes from attempting to find an estimator $\hat{m}$ of $m$ that "minimizes"[19] the RSS

$$\sum_{i=1}^n (Y_i - \hat{m}(X_i))^2 \tag{6.17}$$

without assuming any particular form for the true $m$. This is not achievable directly, since no knowledge on $m$ is available. Recall that what we did in parametric models was to *assume a parametrization* for $m$. For example, in simple linear regression we assumed $m_{\boldsymbol{\beta}}(\mathbf{x}) = \beta_0 + \beta_1 x$, which allowed to tackle the minimization of (6.17) by means of solving

$$m_{\hat{\boldsymbol{\beta}}}(\mathbf{x}) := \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^n (Y_i - m_{\boldsymbol{\beta}}(X_i))^2.$$

The resulting $m_{\hat{\boldsymbol{\beta}}}$ is precisely the estimator that minimizes the RSS among all the *linear estimators*, that is, among the class of estimators that we have parametrized.

When $m$ has no available parametrization and can adopt any mathematical form, an alternative approach is required. The first step is to induce a *local parametrization* for $m$. By a $p$-th[20] order Taylor expression it is possible to obtain that, for $x$ close to $X_i$,

$$\begin{aligned} m(X_i) \approx{}& m(x) + m'(x)(X_i - x) + \frac{m''(x)}{2}(X_i - x)^2 \\ &+ \cdots + \frac{m^{(p)}(x)}{p!}(X_i - x)^p. \end{aligned} \tag{6.18}$$

Then, replacing (6.18) in the population version of (6.17) that replaces $\hat{m}$ with $m$, we have that

$$\sum_{i=1}^n \left( Y_i - \sum_{j=0}^p \frac{m^{(j)}(x)}{j!}(X_i - x)^j \right)^2. \tag{6.19}$$

[19] Obviously, avoiding the spurious perfect fit attained with $\hat{m}(X_i) := Y_i$, $i = 1, \ldots, n$.

[20] Here we employ $p$ for denoting the order of the Taylor expansion and, correspondingly, the order of the associated polynomial fit. Do not confuse $p$ with the number of *original* predictors for explaining $Y$ – there is only one predictor in this section, $X$. However, with a local polynomial fit we expand this predictor to $p$ predictors based on $(X^1, X^2, \ldots, X^p)$.

Expression (6.19) is still not workable: it depends on $m^{(j)}(x)$, $j = 0, \ldots, p$, which of course are unknown, as $m$ is unknown. The **great idea** is to set $\beta_j := \frac{m^{(j)}(x)}{j!}$ and turn (6.19) into a linear regression problem where the unknown parameters are precisely $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\top$. Simply rewriting (6.19) using this idea gives

$$\sum_{i=1}^n \left(Y_i - \sum_{j=0}^p \beta_j (X_i - x)^j\right)^2. \tag{6.20}$$

Now, estimates of $\boldsymbol{\beta}$ automatically produce estimates for $m^{(j)}(x)$! In addition, we know how to obtain an estimate $\hat{\boldsymbol{\beta}}$ that minimizes (6.20), since this is precisely the least squares problem studied in Section 2.2.3. The final touch is to **weight the contributions of each datum** $(X_i, Y_i)$ to the estimation of $m(x)$ according to the proximity of $X_i$ to $x$.[21] We can achieve this precisely by kernels:

$$\hat{\boldsymbol{\beta}}_h := \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} \sum_{i=1}^n \left(Y_i - \sum_{j=0}^p \beta_j (X_i - x)^j\right)^2 K_h(x - X_i). \tag{6.21}$$

Solving (6.21) is easy once the proper notation is introduced. To that end, denote

$$\mathbf{X} := \begin{pmatrix} 1 & X_1 - x & \cdots & (X_1 - x)^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_n - x & \cdots & (X_n - x)^p \end{pmatrix}_{n\times(p+1)}$$

and

$$\mathbf{W} := \mathrm{diag}(K_h(X_1 - x), \ldots, K_h(X_n - x)), \quad \mathbf{Y} := \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}_{n\times 1}.$$

Then we can re-express (6.21) into a *weighted least squares problem*[22] whose exact solution is

$$\begin{aligned} \hat{\boldsymbol{\beta}}_h &= \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top \mathbf{W} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \\ &= (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W} \mathbf{Y}. \end{aligned} \tag{6.22}$$

The estimate[23] for $m(x)$ is therefore computed as

$$\begin{aligned} \hat{m}(x; p, h) &:= \hat{\beta}_{h,0} \\ &= \mathbf{e}_1^\top (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W} \mathbf{Y} \\ &= \sum_{i=1}^n W_i^p(x) Y_i \end{aligned} \tag{6.23}$$

where

$$W_i^p(x) := \mathbf{e}_1^\top (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W} \mathbf{e}_i$$

and $\mathbf{e}_i$ is the $i$-th canonical vector. Just as the Nadaraya–Watson was, the local polynomial estimator is a **weighted linear combination of the responses**.

Two cases deserve special attention on (6.23):

[21] The rationale is simple: $(X_i, Y_i)$ should be more informative about $m(x)$ than $(X_j, Y_j)$ if $x$ and $X_i$ are closer than $x$ and $X_j$. Observe that $Y_i$ and $Y_j$ are ignored in measuring this proximity.

[22] Recall that weighted least squares already appeared in the IRLS of Section 5.2.2.

[23] Recall that the entries of $\hat{\boldsymbol{\beta}}_h$ are estimating $\boldsymbol{\beta} = \left(m(x), m'(x), \frac{m'(x)}{2}, \ldots, \frac{m^{(p)}(x)}{p!}\right)^\top$, so we are indeed estimating $m(x)$ (first entry) *and*, in addition, its derivatives up to order $p$!

- $p = 0$ 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:

  $$W_i^1(x) = \frac{1}{n} \frac{\hat{s}_2(x;h) - \hat{s}_1(x;h)(X_i - x)}{\hat{s}_2(x;h)\hat{s}_0(x;h) - \hat{s}_1(x;h)^2} K_h(x - X_i),$$

  where $\hat{s}_r(x;h) := \frac{1}{n}\sum_{i=1}^n (X_i - x)^r K_h(x - X_i)$.

Recall that the local polynomial fit is computationally **more expensive** than the local constant fit: $\hat{m}(x;p,h)$ is obtained as the solution of a weighted linear problem, whereas $\hat{m}(x;0,h)$ can be directly computed as a weighted mean of the responses.

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 available here.

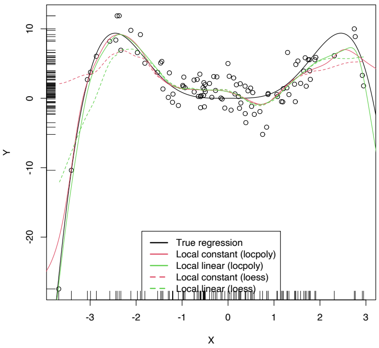
The local polynomial estimator $\hat{m}(\cdot;p,h)$ of $m$ performs a **series of weighted polynomial fits**; as many as points $x$ on which $\hat{m}(\cdot;p,h)$ is to be evaluated.

An inefficient implementation of the local polynomial estimator can be done relatively straightforwardly from the previous insight and from expression (6.22). However, several R packages provide implementations, such as `KernSmooth::locpoly` and R's `loess`[24] (but this one has a different control of the bandwidth plus a set of other modifications). Below are some examples of their usage.

```r
# Generate some data
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)

# KernSmooth::locpoly fits
h <- 0.25
lp0 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = 0,
```

[24] The `lowess` estimator, related with `loess`, is the one employed in R's `panel.smooth`, which is the function in charge of displaying the smooth fits in `lm` and `glm` regression diagnostics. For those diagnostics, it employs a *prefixed* and not data-driven smoothing span of 2/3 – which makes it inevitably a bad choice for certain data patterns. An example of data pattern for which the span 2/3 is not appropriate is the one in upper right panel in Figure 5.12.

```
                             range.x = c(-10, 10), gridsize = 500)
lp1 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = 1,
                             range.x = c(-10, 10), gridsize = 500)
# Provide the evaluation points by range.x and gridsize

# loess fits
span <- 0.25 # The default span is 0.75, which works very bad in this scenario
lo0 <- loess(Y ~ X, degree = 0, span = span)
lo1 <- loess(Y ~ X, degree = 1, span = span)
# loess employs an "span" argument that plays the role of an variable bandwidth
# "span" gives the proportion of points of the sample that are taken into
# account for performing the local fit about x and then uses a triweight kernel
# (not a normal kernel) for weighting the contributions. Therefore, the final
# estimate differs from the definition of local polynomial estimator, although
# the principles in which are based are the same

# Prediction at x = 2
x <- 2
lp1$y[which.min(abs(lp1$x - x))] # Prediction by KernSmooth::locpoly
## [1] 5.445975
predict(lo1, newdata = data.frame(X = x)) # Prediction by loess
##        1
## 5.379652
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)
lines(xGrid, predict(lo0, newdata = data.frame(X = xGrid)), col = 2, lty = 2)
lines(xGrid, predict(lo1, newdata = data.frame(X = xGrid)), col = 3, lty = 2)
legend("bottom", legend = c("True regression", "Local constant (locpoly)",
                             "Local linear (locpoly)", "Local constant (loess)",
                             "Local linear (loess)"),
       lwd = 2, col = c(1:3, 2:3), lty = c(rep(1, 3), rep(2, 2)))
```



As with the Nadaraya–Watson, the local polynomial estimator heavily depends on $h$.

```
# Simple plot of local polynomials for varying h's
manipulate::manipulate({

  # Plot data
  lpp <- KernSmooth::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))

}, p = manipulate::slider(min = 0, max = 4, initial = 0, step = 1),
h = manipulate::slider(min = 0.01, max = 2, initial = 0.5, step = 0.01))
```

A more sophisticated framework for performing nonparametric estimation of the regression function is the np package, which we detail in Section 6.2.4. This package will be the chosen approach for the more challenging situation in which several predictors are present, since the former implementations do not escalate well for more than one predictor.

### 6.2.3 *Asymptotic properties*

What affects the performance of the local polynomial estimator? Is local linear estimation better than local constant estimation? What is the effect of $h$?

The purpose of this section is to provide some highlights on the questions above by examining the theoretical properties of the local polynomial estimator. This is achieved by examining the asymptotic bias and variance of the local linear and local constant estimators.[25] For this goal, we consider the *location-scale model* for $Y$ and its predictor $X$:

$$Y = m(X) + \sigma(X)\varepsilon,$$

where $\sigma^2(x) := \mathbb{V}\mathrm{ar}[Y \mid X = x]$ is the *conditional variance* of $Y$ given $X$ and $\varepsilon$ is such that $\mathbb{E}[\varepsilon \mid X = x]] = 0$ and $\mathbb{V}\mathrm{ar}[\varepsilon \mid X = x]] = 1$. Note that since the conditional variance is not forced to be constant we are implicitly allowing for heteroscedasticity.

The following assumptions[26] are the only requirements to perform the asymptotic analysis of the estimator:

- **A1**.[27] $m$ is twice continuously differentiable.
- **A2**.[28] $\sigma^2$ is continuous and positive.
- **A3**.[29] $f$, the marginal pdf of $X$, is continuously differentiable and *bounded away from zero*.[30]
- **A4**.[31] The kernel $K$ is a symmetric and bounded pdf with finite second moment and is square integrable.
- **A5**.[32] $h = h_n$ is a deterministic sequence of bandwidths such that, when $n \to \infty$, $h \to 0$ and $nh \to \infty$.

The bias and variance are studied in their *conditional* versions on the predictor's sample $X_1, \ldots, X_n$. 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. This is in the spirit of what it was done in the parametric inference of Sections 2.4 and 5.3. The main result is the following, which provides useful insights on the effect of $p$, $m$, $f$ (standing from now on for the marginal pdf of $X$), and $\sigma^2$ in the performance of $\hat{m}(\cdot; p, h)$.

**Theorem 6.1.** *Under **A1**–**A5**, the conditional bias and variance of the local constant ($p = 0$) and local linear ($p = 1$) estimators are*[33]

$$\mathrm{Bias}[\hat{m}(x; p, h) \mid X_1, \ldots, X_n] = B_p(x)h^2 + o_{\mathbb{P}}(h^2), \tag{6.24}$$

$$\mathbb{V}\mathrm{ar}[\hat{m}(x; p, h) \mid X_1, \ldots, X_n] = \frac{R(K)}{nhf(x)}\sigma^2(x) + o_{\mathbb{P}}((nh)^{-1}), \tag{6.25}$$

*where*

$$B_p(x) := \begin{cases} \frac{\mu_2(K)}{2}\left\{m''(x) + 2\frac{m'(x)f'(x)}{f(x)}\right\}, & \text{if } p = 0,\\ \frac{\mu_2(K)}{2}m''(x), & \text{if } p = 1.\end{cases}$$

[25] We do not address the analysis of the general case in which $p \geq 1$. The reader is referred to, e.g., Theorem 3.1 of Fan and Gijbels (1996) for the full analysis.

[26] Recall that these are the **only assumptions** done so far in the model! Compared with the ones made for linear models or generalized linear models, they are extremely mild.

[27] This assumption requires certain smoothness of the regression function, allowing thus for Taylor expansions to be performed. This assumption is important in practice: $\hat{m}(\cdot; p, h)$ is infinitely differentiable if the considered kernels $K$ are.

[28] Avoids the situation in which $Y$ is a degenerated random variable.

[29] Avoids the degenerate situation in which $m$ is estimated at regions without observations of the predictors (such as *holes* in the support of $X$).

[30] Meaning that there exist a positive lower bound for $f$.

[31] Mild assumption inherited from the kde.

[32] Key assumption for reducing the bias and variance of $\hat{m}(\cdot; p, h)$ *simultaneously*.

[33] The notation $o_{\mathbb{P}}(a_n)$ stands for a random variable that converges in probability to zero at a rate faster than $a_n \to 0$. It is mostly employed for denoting non-important terms in asymptotic expansions, like the ones in (6.24)–(6.25).

The bias and variance expressions (6.24) and (6.25) yield very interesting insights:

1. Bias.

   - The **bias decreases with $h$ *quadratically*** for both $p = 0, 1$. That means that small bandwidths $h$ give estimators with low bias, whereas large bandwidths provide largely biased estimators.
   - The bias at $x$ is directly proportional to $m''(x)$ if $p = 1$ or affected by $m''(x)$ if $p = 0$. Therefore:
     - The *bias is negative* in regions where $m$ is concave, i.e., $\{x \in \mathbb{R} : m''(x) < 0\}$. These regions correspond to *peaks and modes of $m$*.
     - Conversely, the *bias is positive* in regions where $m$ is convex, i.e., $\{x \in \mathbb{R} : m''(x) > 0\}$. These regions correspond to *valleys of $m$*.
     - All in all, **the "wilder" the curvature** $m''$, the larger the bias and **the harder to estimate** $m$.
   - The bias for $p = 0$ at $x$ is affected by $m'(x)$, $f'(x)$, and $f(x)$. All of them are quantities that are not present in the bias when $p = 1$. Precisely, for the local constant estimator, the lower the density $f(x)$, the larger the bias. Also, the faster $m$ and $f$ change at $x$ (derivatives), 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 the bias depends on $f'(x)$ and $f(x)$ is referred to as the *design bias* since it depends merely on the predictor's distribution.

2. Variance.

   - The main term of the **variance is the same for** $p = 0, 1$. In addition, it depends directly on $\frac{\sigma^2(x)}{f(x)}$. As a consequence, the lower the density, the more variable $\hat{m}(x; p, h)$ is.[34] Also, the larger the conditional variance at $x$, $\sigma^2(x)$, the more variable $\hat{m}(x; p, h)$ is.[35]
   - The **variance decreases at a factor of** $(nh)^{-1}$. This is related to the so-called *effective sample size $nh$*, which can be thought of as the amount of data in the neighborhood of $x$ that is employed for performing the regression.[36]

> The main takeaway of the analysis of $p = 0$ vs. $p = 1$ is that $p = 1$ **has smaller bias than** $p = 0$ (but of the same order) while **keeping the same variance as** $p = 0$.

An extended version of Theorem 6.1, given in Theorem 3.1 of Fan and Gijbels (1996), shows that this phenomenon extends to

[34] Recall that this makes perfect sense: low density regions of $X$ imply less information about $m$ available.

[35] The same happened in the linear model with the error variance $\sigma^2$.

[36] The variance of an unweighted mean is reduced by a factor $n^{-1}$ when $n$ observations are employed. For computing $\hat{m}(x; p, h)$, $n$ observations are used but in a *weighted* fashion that roughly amounts to considering $nh$ *unweighted* observations.

higher orders: **odd order** ($p = 2\nu + 1, \nu \in \mathbb{N}$) polynomial fits introduce an extra coefficient for the polynomial fit that allows them to **reduce the bias**, while maintaining the **same variance** of the precedent even order ($p = 2\nu$). So, for example, local cubic fits are preferred to local quadratic fits. This motivates the claim that local polynomial fitting is an "odd world" (Fan and Gijbels (1996)).

### *6.2.4 Bandwidth selection*

Bandwidth selection, as for density estimation, has a crucial practical importance for kernel regression estimation. Several bandwidth selectors have been by following cross-validatory and plug-in ideas similar to the ones seen in Section 6.1.3. For simplicity, we briefly mention[37] the DPI analogue for local linear regression for a single continuous predictor and focus mainly on **least squares cross-validation**, as it is a bandwidth selector that readily generalizes to the more complex settings of Section 6.3.

Following the derivation of the DPI for the kde, the first step is to define a suitable error criterion for the estimator $\hat{m}(\cdot; p, h)$. The *conditional* (on the sample of the predictor) MISE of $\hat{m}(\cdot; p, h)$ is often considered:

$$\begin{aligned}\mathrm{MISE}[\hat{m}(\cdot; p, h) \mid X_1, \ldots, X_n] :=& \mathbb{E}\left[\int (\hat{m}(x; p, h) - m(x))^2 f(x)\,\mathrm{d}x \mid X_1, \ldots, X_n\right]\\ =& \int \mathbb{E}\left[(\hat{m}(x; p, h) - m(x))^2 \mid X_1, \ldots, X_n\right] f(x)\,\mathrm{d}x\\ =& \int \mathrm{MSE}\left[\hat{m}(x; p, h) \mid X_1, \ldots, X_n\right] f(x)\,\mathrm{d}x.\end{aligned}$$

Observe that this definition is very similar to the kde's MISE, except for the fact that $f$ appears weighting the quadratic difference: what matters is to minimize the estimation error of $m$ on the regions were the density of $X$ is higher. Recall also that the MISE follows by integrating the conditional MSE, which amounts to the squared bias (6.24) plus the variance (6.25) given in Theorem 6.1. These operations produce the conditional AMISE:

$$\mathrm{AMISE}[\hat{m}(\cdot; p, h) \mid X_1, \ldots, X_n] = h^2 \int B_p(x)^2 f(x)\,\mathrm{d}x + \frac{R(K)}{nh} \int \sigma^2(x)\,\mathrm{d}x$$

and, if $p = 1$, the resulting optimal AMISE bandwidth is

$$h_{\mathrm{AMISE}} = \left[\frac{R(K) \int \sigma^2(x)\,\mathrm{d}x}{2\mu_2^2(K)\theta_{22} n}\right]^{1/5},$$

where $\theta_{22} := \int (m''(x))^2 f(x)\,\mathrm{d}x$.

As happened in the density setting, the AMISE-optimal bandwidth cannot be readily employed, as knowledge about the "curvature" of $m$, $\theta_{22}$, and about $\int \sigma^2(x)\,\mathrm{d}x$ is required. As with the DPI selector, a series of nonparametric estimations of $\theta_{22}$ and high-order curvature terms follow, concluding with a necessary estimation of a higher-order curvature based on a "block polynomial fit".[38] The estimation of $\int \sigma^2(x)\,\mathrm{d}x$ is carried out by assuming homoscedasticity and a compactly supported density $f$. The resulting bandwidth

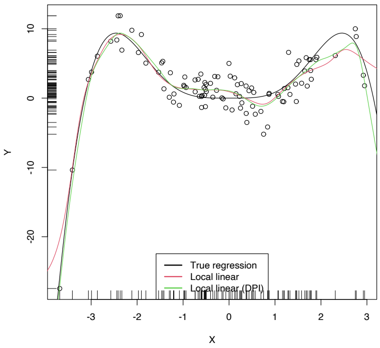
[37] Further details are available in Section 5.8 of Wand and Jones (1995) and references therein.

[38] A fit based on ordinal polynomial fits but done in different blocks of the data.

selector, $\hat{h}_{\mathrm{DPI}}$, has a much faster convergence rate to $h_{\mathrm{MISE}}$ than cross-validatory selectors. However, it is notably more convoluted, and as a consequence is less straightforward to extend to more complex settings.

The DPI selector for the local linear estimator is implemented in `KernSmooth::dpill`.

```r
# Generate some data
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)

# DPI selector
hDPI <- KernSmooth::dpill(x = X, y = Y)

# Fits
lp1 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = 0.25, degree = 0,
                           range.x = c(-10, 10), gridsize = 500)
lp1DPI <- KernSmooth::locpoly(x = X, y = Y, bandwidth = hDPI, degree = 1,
                              range.x = c(-10, 10), gridsize = 500)

# Compare fits
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(lp1$x, lp1$y, col = 2)
lines(lp1DPI$x, lp1DPI$y, col = 3)
legend("bottom", legend = c("True regression", "Local linear",
                            "Local linear (DPI)"),
       lwd = 2, col = 1:3)
```

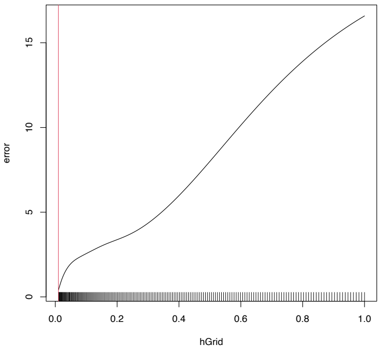


We turn now our attention to cross validation. Following an analogy with the fit of the linear model, we could look for the bandwidth $h$ such that it minimizes an RSS of the form

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}(X_i; p, h))^2. \tag{6.26}$$

As it looks, this is a bad idea. Attempting to minimize (6.26) always leads to $h \approx 0$ that results in a useless interpolation of the data, as illustrated below.

```r
# Grid for representing (6.26)
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)
```



As we know, the root of the problem is the comparison of $Y_i$ with $\hat{m}(X_i; p, h)$, since there is nothing forbidding $h \to 0$ and as a consequence $\hat{m}(X_i; p, h) \to Y_i$. As discussed in (3.18),[39] a solution is to compare $Y_i$ with $\hat{m}_{-i}(X_i; p, h)$, the **leave-one-out estimate** of $m$ computed without the $i$-th datum $(X_i, Y_i)$, yielding the **least**

[39] Recall that $h$ is a tuning parameter!

**squares cross-validation** error

$$\mathrm{CV}(h) := \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}_{-i}(X_i; p, h))^2 \tag{6.27}$$

and then choose

$$\hat{h}_{\mathrm{CV}} := \arg\min_{h>0} \mathrm{CV}(h).$$

The optimization of (6.27) might seem as very computationally demanding, since it is required to compute $n$ regressions for just a *single* evaluation of the cross-validation function. There is, however, a simple and neat theoretical result that vastly reduces the computational complexity, at the price of increasing the memory demand. This trick allows us to compute, with a *single* fit, the cross-validation function.

**Proposition 6.1.** *For any $p \geq 0$, the weights of the leave-one-out estimator $\hat{m}_{-i}(x; p, h) = \sum_{\substack{j=1\\ j\neq i}}^{n} W^p_{-i,j}(x) Y_j$ can be obtained from $\hat{m}(x; p, h) = \sum_{i=1}^{n} W_i^p(x) Y_i$:*

$$W^p_{-i,j}(x) = \frac{W_j^p(x)}{\sum_{\substack{k=1\\ k\neq i}}^{n} W_k^p(x)} = \frac{W_j^p(x)}{1 - W_i^p(x)}.$$

*This implies that*

$$\mathrm{CV}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{m}(X_i; p, h)}{1 - W_i^p(X_i)}\right)^2. \tag{6.28}$$

The result can be proved using that the weights $\{W_i^p(x)\}_{i=1}^n$ add to one, for any $x$, and that $\hat{m}(x; p, h)$ is a linear combination[40] of the responses $\{Y_i\}_{i=1}^n$.

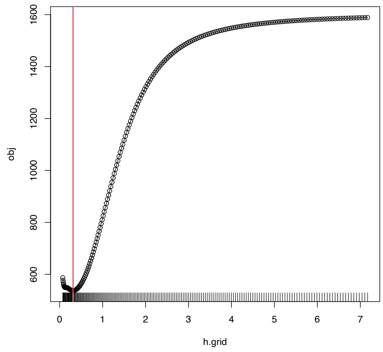
Computing (6.28) requires evaluating the local polynomial estimator at the sample $\{X_i\}_{i=1}^n$ *and* obtaining $\{W_i^p(X_i)\}_{i=1}^n$ (which are needed for evaluating $\hat{m}(X_i; p, h)$). Both tasks can be achieved simultaneously from the $n \times n$ matrix $\left(W_i^p(X_j)\right)_{ij}$ and, if $p = 0$, directly from the *symmetric* $n \times n$ matrix $\left(K_h(X_i - X_j)\right)_{ij}$, whose storage costs $\mathcal{O}\left(\frac{n^2-n}{2}\right)$ (the diagonal is constant).

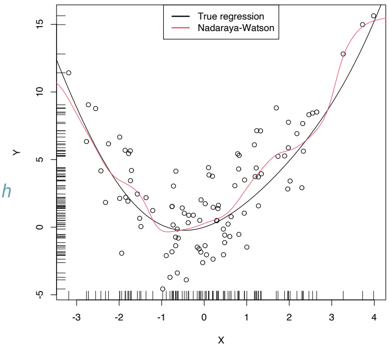
Let's implement $\hat{h}_{\mathrm{CV}}$ for the Nadaraya–Watson estimator.

```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)
```

[40] Indeed, for any other *linear smoother* of the response, the result also holds.

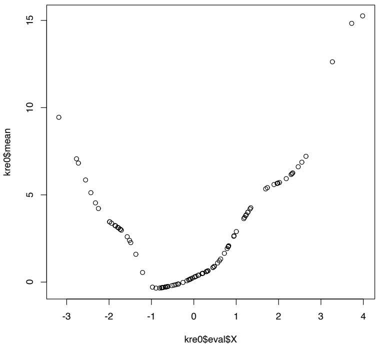
```r
# Objective function
cvNW <- function(X, Y, h, K = dnorm) {

  sum(((Y - mNW(x = X, X = X, Y = Y, h = h, K = K)) /
         (1 - K(0) / colSums(K(outer(X, X, "-") / h))))^2)
  # Beware: outer() is not very memory-friendly!

}

# 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
hCV <- bw.cv.grid(X = X, Y = Y, plot.cv = TRUE)


hCV
## [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 = hCV), col = 2)
legend("top", legend = c("True regression", "Nadaraya-Watson"),
       lwd = 2, col = 1:2)
```



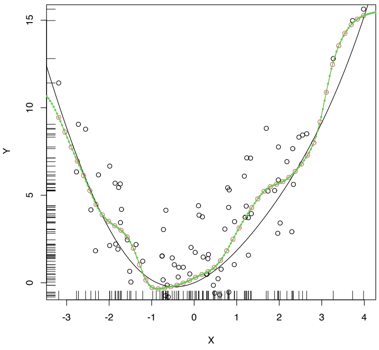


A more sophisticated cross-validation bandwidth selection can be achieved by np::npregbw and np::npreg, as shown in the code below.

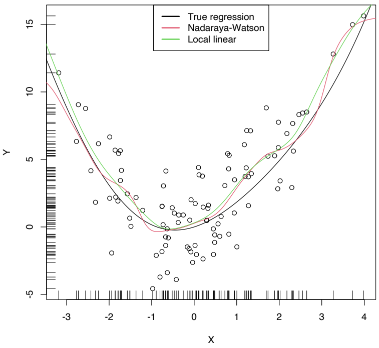
```r
# Turn off the "multistart" messages in the np package
options(np.messages = FALSE)

# np::npregbw computes by default the least squares CV bandwidth associated with
# a local constant fit
bw0 <- np::npregbw(formula = Y ~ X)

# Multiple initial points can be employed for minimizing the CV function (for
# one predictor, defaults to 1)
bw0 <- np::npregbw(formula = Y ~ X, nmulti = 2)

# The "rbandwidth" object contains many useful information, see ?np::npregbw for
# all the returned objects
bw0
##
## Regression Data (100 observations, 1 variable(s)):
##
##                        X
## Bandwidth(s): 0.3112962
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: Y ~ X
## Bandwidth Type: Fixed
## Objective Function Value: 5.368999 (achieved on multistart 1)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
# Recall that the fit is very similar to hCV

# Once the bandwidth is estimated, np::npreg can be directly called with the
# "rbandwidth" object (it encodes the regression to be made, the data, the kind
# of estimator considered, etc). The hard work goes on np::npregbw, not on
# np::npreg
kre0 <- np::npreg(bw0)
kre0
##
## Regression Data: 100 training points, in 1 variable(s)
##                       X
## Bandwidth(s): 0.3112962
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1

# The evaluation points of the estimator are by default the predictor's sample
# (which is not sorted!)
# The evaluation of the estimator is given in "mean"
plot(kre0$eval$X, kre0$mean)
```



```
# The evaluation points can be changed using "exdat"
kre0 <- np::npreg(bw0, exdat = xGrid)

# Plot directly the fit via plot() -- it employs different evaluation points
# than exdat
plot(kre0, col = 2, type = "o")
points(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(kre0$eval$xGrid, kre0$mean, col = 3, type = "o", pch = 16, cex = 0.5)
```



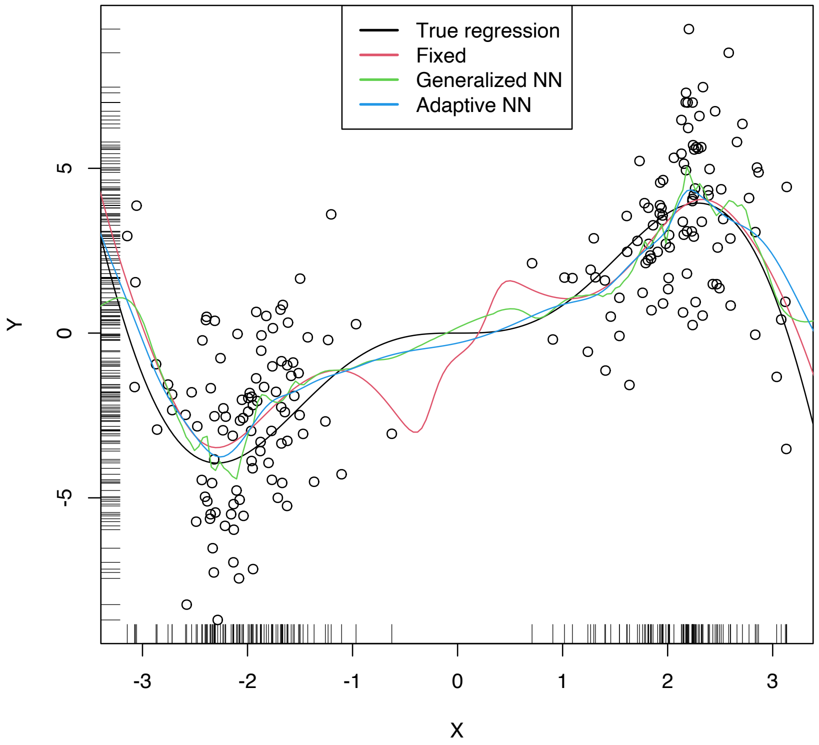
```
# Using the evaluation points

# Local linear fit -- find first the CV bandwidth
bw1 <- np::npregbw(formula = Y ~ X, regtype = "ll")
# regtype = "ll" stands for "local linear", "lc" for "local constant"

# Local linear fit
kre1 <- np::npreg(bw1, exdat = xGrid)

# Comparison
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(kre0$eval$xGrid, kre0$mean, col = 2)
lines(kre1$eval$xGrid, kre1$mean, col = 3)
legend("top", legend = c("True regression", "Nadaraya-Watson", "Local linear"),
       lwd = 2, col = 1:3)
```



The ***adequate* bandwidths** for the **local linear estimator** are usually **larger** than the adequate bandwidths for the local constant estimator. The reason: the extra flexibility of the local linear estimator allows to adapt faster to variations in $m$, whereas the local constant estimator can only achieve this by shrinking the neighborhood about $x$ by means of a small $h$.

There are more sophisticated options for bandwidth selection in `np::npregbw`. For example, the argument `bwtype` allows to estimate data-driven *variable bandwidths* $\hat{h}(x)$ that depend on the evaluation point $x$, rather than fixed bandwidths $\hat{h}$, as we have considered. Roughly speaking, these variable bandwidths are related to the variable bandwidth $\hat{h}_k(x)$ that is necessary to contain the $k$ *nearest neighbors* $X_1, \ldots, X_k$ of $x$ in the neighborhood $(x - \hat{h}_k(x), x + \hat{h}_k(x))$. There is a potential gain in employing variable bandwidths, as the estimator can adapt the amount of smoothing according to the density of the predictor. We do not investigate this approach in detail but just point to its implementation.

```r
# Generate some data with bimodal density
set.seed(12345)
n <- 100
eps <- rnorm(2 * n, sd = 2)
m <- function(x) x^2 * sin(x)
X <- c(rnorm(n, mean = -2, sd = 0.5), rnorm(n, mean = 2, sd = 0.5))
Y <- m(X) + eps
xGrid <- seq(-10, 10, l = 500)

# Constant bandwidth
bwc <- np::npregbw(formula = Y ~ X, bwtype = "fixed", regtype = "ll")
krec <- np::npreg(bwc, exdat = xGrid)

# Variable bandwidths
bwg <- np::npregbw(formula = Y ~ X, bwtype = "generalized_nn", regtype = "ll")
kreg <- np::npreg(bwg, exdat = xGrid)
bwa <- np::npregbw(formula = Y ~ X, bwtype = "adaptive_nn", regtype = "ll")
krea <- np::npreg(bwa, exdat = xGrid)

# Comparison
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(xGrid, m(xGrid), col = 1)
lines(krec$eval$xGrid, krec$mean, col = 2)
lines(kreg$eval$xGrid, kreg$mean, col = 3)
lines(krea$eval$xGrid, krea$mean, col = 4)
legend("top", legend = c("True regression", "Fixed", "Generalized NN",
                         "Adaptive NN"),
       lwd = 2, col = 1:4)
```



```
# Observe how the fixed bandwidth may yield a fit that produces serious
# artifacts in the low density region. At that region the NN-based bandwidths
# enlarge to borrow strength from the points in the high density regions,
# whereas in the high density regions they shrink to adapt faster to the
# changes of the regression function
```

## 6.3 *Kernel regression with mixed multivariate data*

Until now, we have studied the simplest situation for performing nonparametric estimation of the regression function: a *single, continuous*, predictor $X$ is available for explaining $Y$, a continuous response. This served for introducing the main concepts without the additional technicalities associated with more complex predictors. We now extend the study to the case in which there are

- **multiple predictors** $X_1, \ldots, X_p$ and
- possible non-continuous predictors, namely **categorical predictors** and discrete predictors.

The first point is how to extend the local polynomial estimator $\hat{m}(\cdot; q, h)$ [41] to deal with $p$ *continuous* predictors. Although this can be done for $q \geq 0$, we focus on the **local constant and linear estimators** ($q = 0, 1$) for avoiding excessive technical complications.[42] Also, to avoid a quick escalation of the number of smoothing bandwidths, it is customary to consider **product kernels**. With these two restrictions, the estimators for $m$ based on a sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$ extend easily from the developments in Sections 6.2.1 and 6.2.2:

- **Local constant estimator**. We can replicate the argument in (6.13) with a multivariate kde for $f_{\mathbf{X}}$ based on product kernels with

[41] Now $q$ denotes the order of the polynomial fit since $p$ stands for the number of predictors.

[42] In particular, the consideration of Taylor expansions of $m : \mathbb{R}^p \to \mathbb{R}$ of more than two orders that involve the consideration of the *vector* of partial derivatives $\mathrm{D}^{\otimes s} m(\mathbf{x})$ formed by $\frac{\partial^s m(\mathbf{x})}{\partial x_1^{s_1} \cdots \partial x_p^{s_1}}$, where $s = s_1 + \cdots + s_p$. For example: if $s = 1$, then $\mathrm{D}^{\otimes 1} m(\mathbf{x})$ is just the transpose of the gradient $\nabla m(\mathbf{x})^\top$; if $s = 2$, then $\mathrm{D}^{\otimes 2} m(\mathbf{x})$ is the column stacking of the Hessian matrix $\mathcal{H} m(\mathbf{x})$; and if $s \geq 3$ the arrangement of derivatives is made in terms of tensors containing the derivatives $\frac{\partial^s m(\mathbf{x})}{\partial x_1^{s_1} \cdots \partial x_p^{s_1}}$.

bandwidth vector $\mathbf{h}$, which gives

$$\hat{m}(\mathbf{x};0,\mathbf{h}) := \sum_{i=1}^n \frac{K_{\mathbf{h}}(\mathbf{x}-\mathbf{X}_i)}{\sum_{i=1}^n K_{\mathbf{h}}(\mathbf{x}-\mathbf{X}_i)} Y_i = \sum_{i=1}^n W_i^0(\mathbf{x}) Y_i$$

where

$$\begin{aligned} K_{\mathbf{h}}(\mathbf{x}-\mathbf{X}_i) &:= K_{h_1}(x_1 - X_{i1}) \times \overset{p}{\cdots} \times K_{h_p}(x_p - X_{ip}),\\ W_i^0(\mathbf{x}) &:= \frac{K_{\mathbf{h}}(\mathbf{x}-\mathbf{X}_i)}{\sum_{i=1}^n K_{\mathbf{h}}(\mathbf{x}-\mathbf{X}_i)}. \end{aligned}$$

- **Local linear estimator**. Considering the Taylor expansion $m(\mathbf{X}_i) \approx m(\mathbf{x}) + \nabla m(\mathbf{x})(\mathbf{X}_i - \mathbf{x})$ [43] instead of (6.18) it is possible to arrive to the analogous of (6.21),

$$\hat{\boldsymbol{\beta}}_{\mathbf{h}} := \arg \min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} \sum_{i=1}^n \left(Y_i - \boldsymbol{\beta}^\top (1, (\mathbf{X}_i - \mathbf{x})^\top)^\top\right)^2 K_{\mathbf{h}}(\mathbf{x}-\mathbf{X}_i),$$

and then solve the problem in the exact same way but now considering

$$\mathbf{X} := \begin{pmatrix} 1 & (\mathbf{X}_1 - \mathbf{x})^\top \\ \vdots & \vdots \\ 1 & (\mathbf{X}_n - \mathbf{x})^\top \end{pmatrix}_{n\times(p+1)}$$

and

$$\mathbf{W} := \mathrm{diag}(K_{\mathbf{h}}(\mathbf{X}_1 - \mathbf{x}), \ldots, K_{\mathbf{h}}(\mathbf{X}_n - \mathbf{x})).$$

The estimate[44] for $m(x)$ is therefore computed as

$$\begin{aligned} \hat{m}(\mathbf{x};1,h) &:= \hat{\beta}_{\mathbf{h},0}\\ &= \mathbf{e}_1^\top (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{W}\mathbf{Y}\\ &= \sum_{i=1}^n W_i^1(\mathbf{x}) Y_i \end{aligned}$$

where

$$W_i^1(\mathbf{x}) := \mathbf{e}_1^\top (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{W}\mathbf{e}_i.$$

The cross-validation bandwidth selection rule studied on Section 6.2.4 extends neatly to the multivariate case:

$$\begin{aligned} \mathrm{CV}(\mathbf{h}) &:= \frac{1}{n}\sum_{i=1}^n (Y_i - \hat{m}_{-i}(\mathbf{X}_i; p, \mathbf{h}))^2,\\ \hat{\mathbf{h}}_{\mathrm{CV}} &:= \arg \min_{h_1,\ldots,h_p>0} \mathrm{CV}(\mathbf{h}). \end{aligned}$$

Obvious complications appear in the optimization of $\mathrm{CV}(\mathbf{h})$, which now is a $\mathbb{R}^p \to \mathbb{R}$ function. Importantly, the trick described in Proposition 6.1 also holds with obvious modifications.

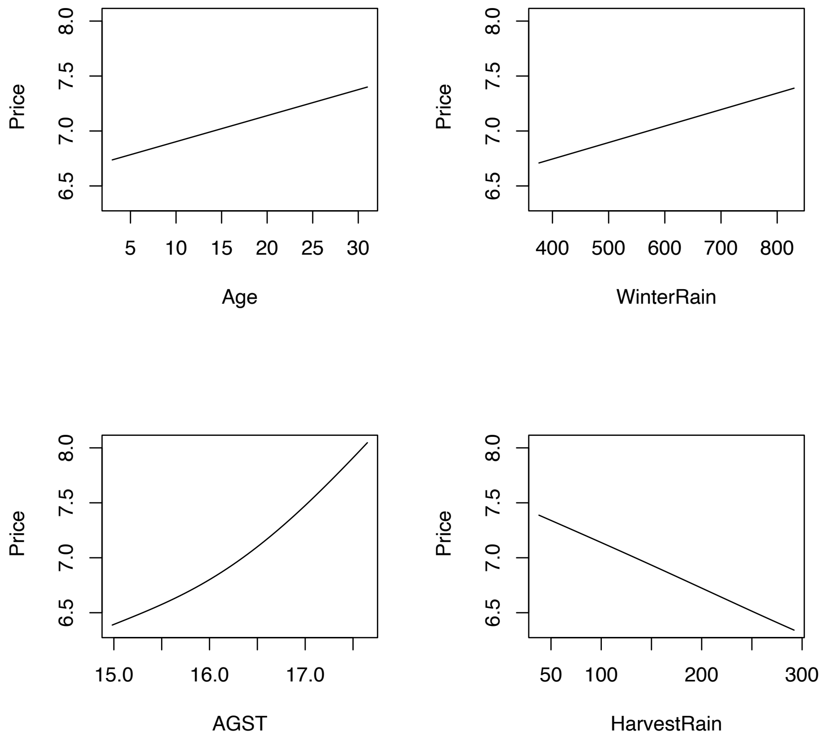
Let's see an application of multivariate kernel regression for the `wine` dataset.

[43] The gradient $\nabla m(\mathbf{x}) = \left(\frac{\partial m(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial m(\mathbf{x})}{\partial x_p}\right)$ is a *row* vector.

[44] Recall that now the entries of $\hat{\boldsymbol{\beta}}_{\mathbf{h}}$ are estimating $\boldsymbol{\beta} = (m(\mathbf{x}), \nabla m(\mathbf{x}))^\top$.

```
# Employing the wine dataset
wine <- read.table(file = "wine.csv", header = TRUE, sep = ",")
```

```
# Bandwidth by CV for local linear estimator -- a product kernel with 4
# bandwidths. Employs 4 random starts for minimizing the CV surface
bwWine <- np::npregbw(formula = Price ~ Age + WinterRain + AGST + HarvestRain,
                      data = wine, regtype = "ll")
bwWine
##
## Regression Data (27 observations, 4 variable(s)):
##
##                   Age WinterRain      AGST HarvestRain
## Bandwidth(s): 3616691  290170218 0.8725243    106.5079
##
## Regression Type: Local-Linear
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: Price ~ Age + WinterRain + AGST + HarvestRain
## Bandwidth Type: Fixed
## Objective Function Value: 0.0873955 (achieved on multistart 4)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4

# Regression
fitWine <- np::npreg(bwWine)
summary(fitWine)
##
## Regression Data: 27 training points, in 4 variable(s)
##                   Age WinterRain      AGST HarvestRain
## Bandwidth(s): 3616691  290170218 0.8725243    106.5079
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 0.2079691
## R-squared: 0.8889649
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4

# Plot marginal effects of each predictor on the response (the rest of
# predictors are fixed to their marginal medians)
plot(fitWine)
# Therefore:
# - Age is positively related to Price (almost linearly)
# - WinterRain is positively related to Price (with a subtle nonlinearity)
# - AGST is positively related to Price, but now we see what it looks like a
#   quadratic pattern
# - HarvestRain is negatively related to Price (almost linearly)
```



The $R^2$ outputted by the `summary` of `np::npreg` is defined as

$$R^2 := \frac{(\sum_{i=1}^n (Y_i - \bar{Y})(\hat{Y}_i - \bar{Y}))^2}{(\sum_{i=1}^n (Y_i - \bar{Y})^2)(\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2)}$$

and is *neither* the $r^2_{y\hat{y}}$ (because it is not guaranteed that $\bar{Y} = \bar{\hat{Y}}$!) *nor* "the percentage of variance explained" by the model – this interpretation only makes sense within the linear model context. It is however a quantity in $[0,1]$ that attains $R^2 = 1$ when the fit is perfect.

Non-continuous variables can be taken into account by defining suitably adapted kernels. The two main possibilities for non-continuous data are:

- *Categorical or unordered discrete variables*. For example, `iris$species` is a categorical variable in which ordering does not make sense. These variables are specified in R by `factor`. Due to the lack of ordering, the basic mathematical operation behind a kernel, a distance computation,[45] is senseless. That motivates the Aitchison and Aitken (1976) kernel.

  Assume that the categorical random variable $X_d$ has $c_d$ different levels. Then, it can be represented as $X_d \in C_d := \{0, 1, \ldots, c_d - 1\}$. For $x_d, X_d \in C_d$, the Aitchison and Aitken (1976) **unordered discrete kernel** is

$$l(x_d, X_d; \lambda) = \begin{cases} 1 - \lambda, & \text{if } x_d = X_d, \\ \frac{\lambda}{c_d - 1}, & \text{if } x_d \neq X_d, \end{cases}$$

[45] Recall $K_h(x - X_i)$.

where $\lambda \in [0, (c_d - 1)/c_d]$ is the bandwidth.

- *Ordinal or ordered discrete variables*. For example, wine$Year is a discrete variable with a clear order, but it is not continuous. These variables are specified by ordered (an ordered factor). In these variables there is ordering, but distances are discrete.

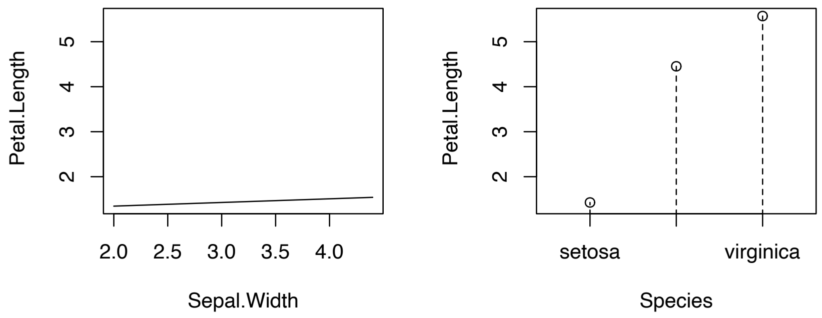
  If the ordered discrete random variable $X_d$ can take $c_d$ different ordered values, then it can be represented as $X_d \in C_d := \{0, 1, \ldots, c_d - 1\}$. For $x_d, X_d \in C_d$, a possible (Li and Racine, 2007) **ordered discrete kernel** is

  $$l(x_d, X_d; \lambda) = \lambda^{|x_d - X_d|},$$

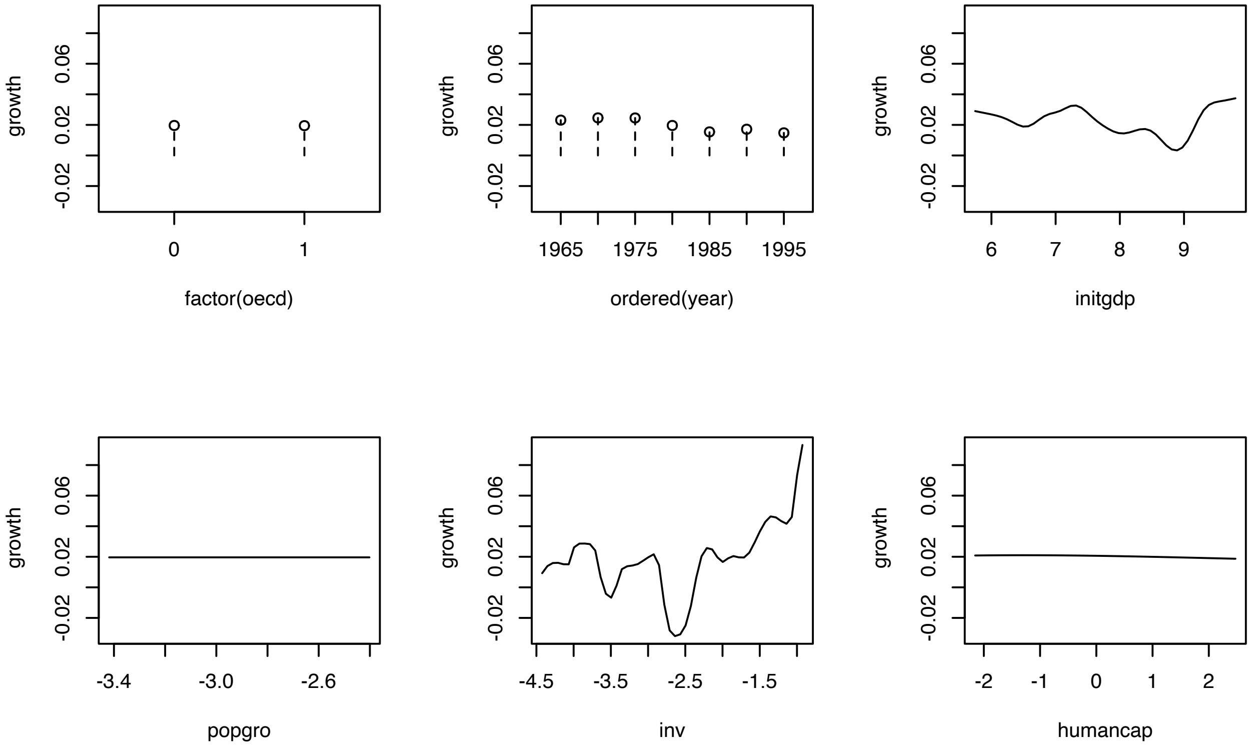
  where $\lambda \in [0, 1]$ is the bandwidth.

The np package employs a variation of the previous kernels. The following examples illustrate their use.

```
# Bandwidth by CV for local linear estimator
# Recall that Species is a factor!
bwIris <- np::npregbw(formula = Petal.Length ~ Sepal.Width + Species,
                      data = iris, regtype = "ll")
bwIris
##
## Regression Data (150 observations, 2 variable(s)):
##
##                Sepal.Width      Species
## Bandwidth(s):    898696.6 2.357536e-07
##
## Regression Type: Local-Linear
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: Petal.Length ~ Sepal.Width + Species
## Bandwidth Type: Fixed
## Objective Function Value: 0.1541057 (achieved on multistart 1)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
# Product kernel with 2 bandwidths

# Regression
fitIris <- np::npreg(bwIris)
summary(fitIris)
##
## Regression Data: 150 training points, in 2 variable(s)
##                Sepal.Width      Species
## Bandwidth(s):    898696.6 2.357536e-07
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 0.3775005
## R-squared: 0.9539633
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1

# Plot marginal effects of each predictor on the response
par(mfrow = c(1, 2))
plot(fitIris, plot.par.mfrow = FALSE)
```



```
# Options for the plot method for np::npreg available at ?np::npplot
# For example, xq = 0.5 (default) indicates that all the predictors are set to
# their medians for computing the marginal effects

# Example from ?np::npreg: modeling of the GDP growth of a country from
# economic indicators of the country
# The predictors contain a mix of unordered, ordered, and continuous variables

# Load data
data(oecdpanel, package = "np")

# Bandwidth by CV for local constant -- use only two starts to reduce the
# computation time
bwOECD <- np::npregbw(formula = growth ~ factor(oecd) + ordered(year) +
                        initgdp + popgro + inv + humancap, data = oecdpanel,
                      regtype = "lc", nmulti = 2)
bwOECD
##
## Regression Data (616 observations, 6 variable(s)):
##
##                factor(oecd) ordered(year)   initgdp  popgro        inv humancap
## Bandwidth(s):   0.02413475     0.8944088 0.1940763 9672508 0.09140376 1.223202
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: growth ~ factor(oecd) + ordered(year) + initgdp + popgro + inv +
##     humancap
## Bandwidth Type: Fixed
## Objective Function Value: 0.0006545946 (achieved on multistart 2)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
##
## Ordered Categorical Kernel Type: Li and Racine
## No. Ordered Categorical Explanatory Vars.: 1

# Regression
fitOECD <- np::npreg(bwOECD)
summary(fitOECD)
##
## Regression Data: 616 training points, in 6 variable(s)
##                factor(oecd) ordered(year)   initgdp  popgro        inv humancap
## Bandwidth(s):   0.02413475     0.8944088 0.1940763 9672508 0.09140376 1.223202
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.01814086
## R-squared: 0.6768302
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
##
## Ordered Categorical Kernel Type: Li and Racine
## No. Ordered Categorical Explanatory Vars.: 1

# Plot marginal effects of each predictor
par(mfrow = c(2, 3))
plot(fitOECD, plot.par.mfrow = FALSE)
```



## 6.4 *Prediction and confidence intervals*

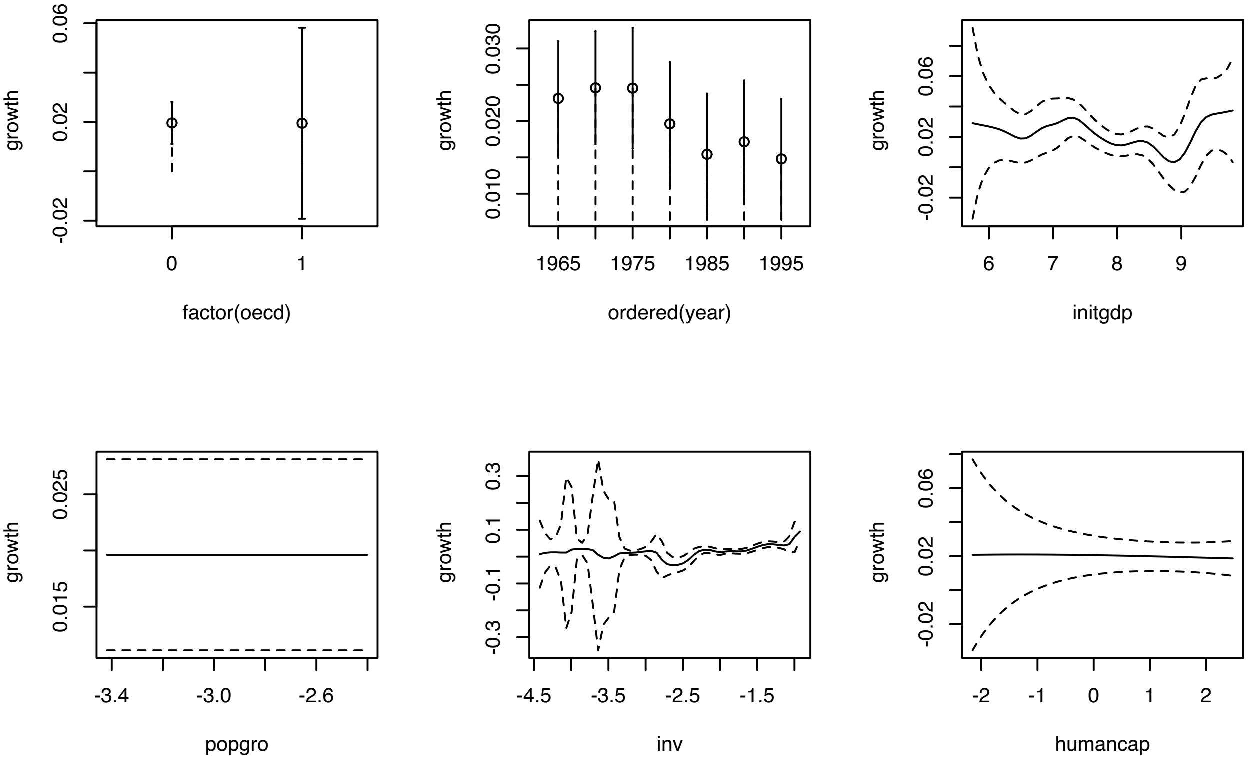
The prediction of the conditional response $m(\mathbf{x}) = \mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$ with the local polynomial estimator reduces to evaluate $\hat{m}(\mathbf{x}; p, \mathbf{h})$. The fitted values are, therefore, $\hat{Y}_i := \hat{m}(\mathbf{X}_i; p, \mathbf{h})$, $i = 1, \ldots, n$. The `np` package has methods to perform these operations via the `predict` and `fitted` functions.

More interesting is the discussion about the *uncertainty* of $\hat{m}(\mathbf{x}; p, \mathbf{h})$ and, as a consequence, of the predictions. Differently to what happened in parametric models, in nonparametric regression there is no parametric distribution of the response that can help to carry out the inference and, consequently, to address the uncertainty of the estimation. Because of this, it is required to resort to somehow convoluted asymptotic expressions[46] (carried out by `predict` if `se.fit = TRUE`) or to *bootstrap resampling procedures*. The default bootstrap resampling procedure in `np` is the so-called ***wild* bootstrap** (Liu, 1988), which is particularly well-suited for regression problems with potential heteroscedasticity.

Due to their increasing complexity, we just cover very superficially these methods with the following example.

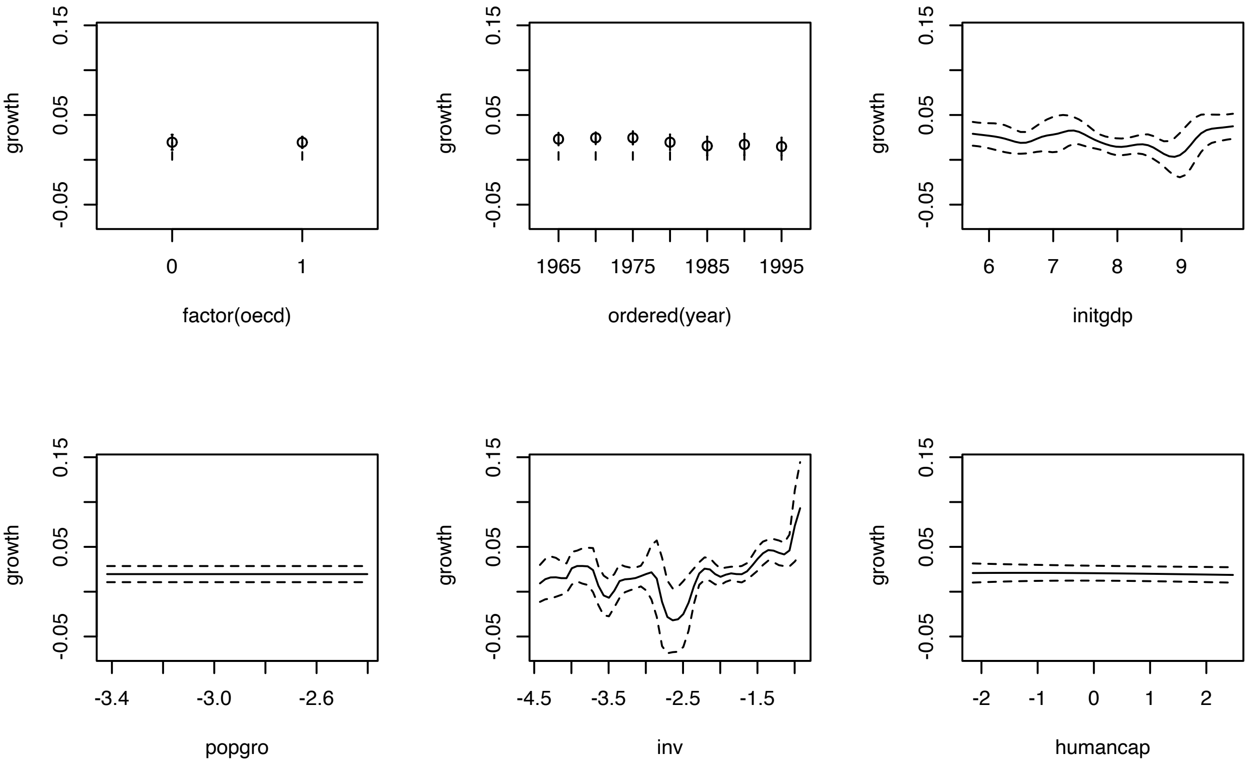
[46] Precisely, the ones associated with the asymptotic normality of $\hat{m}(\mathbf{x}; p, \mathbf{h})$ and that are based on the results of Theorem 6.1.

```
# Asymptotic confidence bands for the marginal effects of each predictor on the
# response
par(mfrow = c(2, 3))
plot(fitOECD, plot.errors.method = "asymptotic", common.scale = FALSE,
     plot.par.mfrow = FALSE)
```



```
# Bootstrap confidence bands
# They take more time to compute because a resampling + refitting takes place
par(mfrow = c(2, 3))
plot(fitOECD, plot.errors.method = "bootstrap", plot.par.mfrow = FALSE)
```

```
# The asymptotic standard error associated with the regression evaluated at the
# evaluation points are in $merr
head(fitOECD$merr)
## [1] 0.004534787 0.006105144 0.001558307 0.002400982 0.006213965 0.005274139

# Recall that in $mean we had the regression evaluated at the evaluation points,
# by default the sample of the predictors, so in this case the same as the
# fitted values
head(fitOECD$mean)
## [1] 0.02877743 0.02113513 0.03592755 0.04027579 0.01099637 0.03888485

# Prediction for the first 3 points + standard errors
pred <- predict(fitOECD, newdata = oecdpanel[1:3, ], se.fit = TRUE)

# Approximate (based on assuming asymptotic normality) 100(1 - alpha)% CI for
# the conditional mean of the first 3 points
alpha <- 0.05
pred$fit + (qnorm(1 - alpha / 2) * pred$se.fit) %o% c(-1, 1)
##             [,1]       [,2]
## [1,] 0.019889408 0.03766545
## [2,] 0.009169271 0.03310100
## [3,] 0.032873327 0.03898178
```



> Extrapolation with kernel regression estimators is particularly dangerous. Keep in mind that the kernel estimators *smooth the data* in order to estimate $m$, the trend. If there is no data close to the point $\mathbf{x}$ at which $m(\mathbf{x})$ is estimated, then the estimation will heavily depend on the closest data point to $\mathbf{x}$. If employing compactly-supported kernels (so that they can take the value 0 exactly), the estimate may not be even properly defined.

## 6.5 *Local likelihood*

We explore in this section an extension of the local polynomial estimator that reproduces the extension generalized linear models made of linear models. 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 modeling. Theorem B.1 shows that, under the assumptions given in Section 2.3, the ML estimate of $\boldsymbol{\beta}$ in the linear model

$$Y \mid (X_1, \ldots, X_p) \sim \mathcal{N}(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p, \sigma^2) \tag{6.29}$$

is equivalent to the least squares estimate, $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$. The reason was the form of the conditional (on $X_1, \ldots, X_p$) likelihood:

$$\begin{aligned}\ell(\boldsymbol{\beta}) = &-\frac{n}{2}\log(2\pi\sigma^2)\\ &-\frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \cdots - \beta_p X_{ip})^2.\end{aligned}$$

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 (6.29) by $X^j$. This results in

$$Y \mid X \sim \mathcal{N}(\beta_0 + \beta_1 X + \cdots + \beta_p X^p, \sigma^2). \tag{6.30}$$

Therefore, the *weighted log-likelihood* of the linear model (6.30) about $x$ is

$$\begin{aligned}\ell_{x,h}(\boldsymbol{\beta}) = &-\frac{n}{2}\log(2\pi\sigma^2)\\ &-\frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_i - \cdots - \beta_p X_i^p)^2 K_h(x - X_i).\end{aligned} \tag{6.31}$$

Maximizing with respect to $\boldsymbol{\beta}$ the *local* log-likelihood (6.31) provides $\hat{\beta}_0 = \hat{m}(x; p, h)$, the local polynomial estimator, as it was obtained in (6.21), but now from a likelihood-based perspective. The key insight is to realize that the very 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 available here.

We illustrate the local likelihood principle for the logistic regression. In this case, $\{(x_i, Y_i)\}_{i=1}^n$ with

$$Y_i \mid X_i = x_i \sim \mathrm{Ber}(\mathrm{logistic}(\eta(x_i))), \quad i = 1, \ldots, n,$$

with the polynomial term[47]

$$\eta(x) := \beta_0 + \beta_1 x + \cdots + \beta_p x^p.$$

The log-likelihood of $\boldsymbol{\beta}$ is

$$\begin{aligned}\ell(\boldsymbol{\beta}) &= \sum_{i=1}^n \{Y_i \log(\mathrm{logistic}(\eta(x_i))) + (1 - Y_i)\log(1 - \mathrm{logistic}(\eta(x_i)))\}\\ &= \sum_{i=1}^n \ell(Y_i, \eta(x_i)),\end{aligned}$$

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 $\boldsymbol{\beta}$).

[47] If $p = 1$, then we have the usual simple logistic model.

[48] No analytical solution for the optimization problem, numerical optimization is needed.

The local log-likelihood of $\boldsymbol{\beta}$ about $x$ is then

$$\ell_{x,h}(\boldsymbol{\beta}) := \sum_{i=1}^{n} \ell(Y_i, \eta(X_i - x))K_h(x - X_i). \tag{6.32}$$

Maximizing[48] the local log-likelihood (6.32) with respect to $\boldsymbol{\beta}$ provides

$$\hat{\boldsymbol{\beta}}_h = \arg\max_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} \ell_{x,h}(\boldsymbol{\beta}),$$

where $\hat{\boldsymbol{\beta}}_h = (\hat{\beta}_{h,0}, \hat{\beta}_{h,1}, \ldots, \hat{\beta}_{h,p})^\top$. The local likelihood estimate of $\eta(x)$ is

$$\hat{\eta}(x) := \hat{\beta}_{h,0}.$$

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

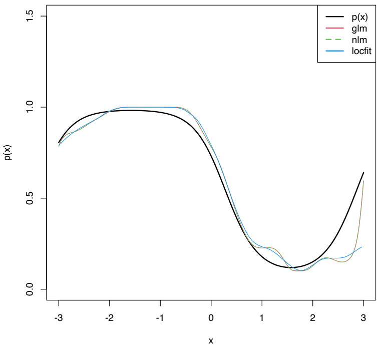
$$\hat{m}_\ell(x; h, p) := g^{-1}\left(\hat{\eta}(x)\right) = \mathrm{logistic}(\hat{\beta}_{h,0}). \tag{6.33}$$

Each evaluation of $\hat{m}_\ell(x; h, p)$ in a different $x$ requires, thus, a weighted fit of the underlying logistic model.

The code below shows three different ways of implementing the local logistic regression (of first degree) in R.

```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)

# Approach 1: optimize the weighted log-likelihood through the workhorse
# function underneath glm, glm.fit
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]
  })
)

# Approach 2: optimize directly the weighted log-likelihood
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]
  })
)

# Approach 3: employ locfit::locfit
# Bandwidth cannot be controlled explicitly -- only through nn in ?lp
fitLocfit <- locfit::locfit(Y ~ locfit::lp(X, deg = 1, nn = h),
                            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 = 3, lty = 2)
plot(fitLocfit, add = TRUE, col = 4)
legend("topright", legend = c("p(x)", "glm", "nlm", "locfit"), lwd = 2,
       col = c(1, 2, 3, 4), lty = c(1, 1, 2, 1))
```



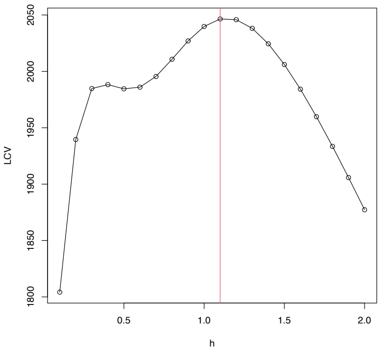
Bandwidth selection can be done by means of *likelihood cross-validation*. The objective is to maximize the local likelihood fit at $(x_i, Y_i)$ but removing the influence by the datum itself. That is, maximizing

$$\mathrm{LCV}(h) = \sum_{i=1}^{n} \ell(Y_i, \hat{\eta}_{-i}(X_i)), \tag{6.34}$$

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 (6.33) forbids a simplifying result as Proposition 6.1. Thus, in principle,[49] it is required to fit $n$ local likelihoods for sample size $n-1$ for obtaining a single evaluation of (6.34).

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)
```



[49] The interested reader is referred to Sections 4.3.3 and 4.4.3 of Loader (1999) for an *approximation* of (6.34) that only requires a local likelihood fit for a single sample.

# A
# Informal review on hypothesis testing

The process of hypothesis testing has an interesting analogy with a (simplified) trial. The analogy helps understand the elements present in a formal hypothesis test in an intuitive way.

| Hypothesis test | Trial |
|---|---|
| Null hypothesis $H_0$ | The **defendant**: an individual **accused** of committing a crime. He is backed up by the **presumption of innocence**, which means that he is *not guilty* until there are enough evidence to support his guilt. |
| Sample $X_1, \ldots, X_n$ | Collection of small **evidence supporting innocence and guilt** of the defendant. This evidence contains a certain degree of uncontrollable randomness due to how they were collected and the context regarding the case.[1] |
| Test statistic[2] $T_n$ | **Summary of the evidence presented** by the prosecutor and defense lawyer. |
| Distribution of $T_n$ under $H_0$ | The **judge** conducting the trial. Evaluates and measures the evidence presented by both sides and presents a verdict for the defendant. |
| Significance level $\alpha$ | $1-\alpha$ is the **strength of evidence** required by the judge for condemning the defendant. The judge allows evidence that, on average, condemn $100\alpha\%$ of the innocents, due to the randomness inherent to the evidence collection process. The level $\alpha = 0.05$ is considered to be reasonable.[3] |

[1] Think about phenomena that may randomly support innocence or guilt of the defendant, irrespective of his true condition. For example: spurious coincidences ("happen to be in the wrong place at the wrong time"), loss of evidence during the case, previous past statements of the defendant, doubtful identification by witness, imprecise witness testimonies, unverifiable alibi, etc.

[2] Usually simply referred to as *statistic*.

[3] As the judge has to have the power of condemning a guilty defendant. Setting $\alpha = 0$ (no innocents are declared guilty) would result in a judge that systematically declares everybody *not guilty*. Therefore, a compromise is needed.

| Hypothesis test | Trial |
|---|---|
| $p$-value | **Decision** of the judge that measures the degree of compatibility, in a scale 0–1, of the presumption of innocence with the summary of the evidence presented. If $p$-value $< \alpha$, the defendant is declared *guilty*, as the evidence supporting his guilt is strong enough to override his presumption of innocence. Otherwise, he is declared *not guilty*. |
| $H_0$ is rejected | The defendant is declared *guilty*: there are **strong evidence supporting his guilt**. |
| $H_0$ is not rejected | The defendant is declared *not guilty*: either he is **innocent or there is not enough evidence supporting his guilt**. |

More formally, the $p$-value of a hypothesis test about $H_0$ is defined as:

> The $p$-value is the probability of obtaining a test 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 likely that the evidence against $H_0$ is 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).

Let's quickly illustrate the previous fact with the well-known *Kolmogorov–Smirnov test*. This test evaluates whether the unknown cdf of $X$, $F$, equals a specified cdf $F_0$. In other words, it tests the null hypothesis[4]

$$H_0 : F = F_0$$

versus the *alternative hypothesis*[5]

$$H_1 : F \neq F_0.$$

[4] Understood as $F(x) = F_0(x)$ for *all* $x \in \mathbb{R}$. For $F \neq F_0$, we mean that $F(x) \neq F_0(x)$ for *at least one* $x \in \mathbb{R}$.

[5] Formally, a null hypothesis $H_0$ is tested *against* an alternative hypothesis $H_1$. The concept of alternative hypothesis was not addressed in the trial analogy for the sake of simplicity, but you may think of $H_1$ as the defendant being *not guilty* or as the plaintiff or complainant. The alternative hypothesis $H_1$ represents the "alternative truth" to $H_0$ and the test decides between $H_0$ and $H_1$, only rejecting $H_0$ in favor of $H_1$ if there are enough evidence on the data against $H_0$ or supporting $H_1$. Obviously, $H_0 \cap H_1 = \emptyset$. But recall that also $H_1 \subset \neg H_0$ or, in other words, $H_1$ may be more restrictive than the negation of $H_0$.

For that purpose, given a sample $X_1, \ldots, X_n$ of $X$, the Kolmogorov–Smirnov statistic $D_n$ is computed:

$$\begin{aligned} D_n &:= \sqrt{n} \sup_{x \in \mathbb{R}} |F_n(x) - F_0(x)| = \max(D_n^+, D_n^-), && \text{(A.1)} \\ D_n^+ &:= \sqrt{n} \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - U_{(i)} \right\}, \\ D_n^- &:= \sqrt{n} \max_{1 \leq i \leq n} \left\{ U_{(i)} - \frac{i-1}{n} \right\}. \end{aligned}$$

where $F_n$ represents the empirical cdf of $X_1, \ldots, X_n$ and $U_{(j)}$ stands for the $j$-th **sorted** $U_i := F_0(X_i)$, $i = 1, \ldots, n$. If $H_0$ holds, then $D_n$ tends to be small. Conversely, when $F \neq F_0$, larger values of $D_n$ are expected, so the test rejects when $D_n$ is large.

If $H_0$ holds, then $D_n$ has an asymptotic[6] cdf given by the Kolmogorov–Smirnov's $K$ function:

$$\lim_{n \to \infty} \mathbb{P}[D_n \leq x] = K(x) := 1 - 2 \sum_{m=1}^{\infty} (-1)^{m-1} e^{-2m^2x^2}. \qquad \text{(A.2)}$$

The test statistic $D_n$, the asymptotic cdf $K$, and the associated asymptotic $p$-value[7] are readily implemented in R through the `ks.test` function.

**Exercise A.1.** Implement the Kolmogorov–Smirnov test from the equations above. This amounts to:

a. Provide a function for computing the test statistic (A.1) from a sample $X_1, \ldots, X_n$ and a cdf $F_0$.
b. Implement the $K$ function (A.2).
c. Call the previous functions from a routine that returns the asymptotic $p$-value of the test.

Compare that the implementations coincide with the ones of the `ks.test` function when `exact = FALSE`. *Note*: `ks.test` computes $D_n/\sqrt{n}$ instead of $D_n$.

```r
# Sample data from a N(0, 1)
set.seed(3245678)
n <- 50
x <- rnorm(n)

# Kolmogorov-Smirnov test for H_0 : F = N(0, 1). Does not reject.
ks.test(x, "pnorm")
##
##  Exact one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.050298, p-value = 0.9989
## alternative hypothesis: two-sided

# Simulation of p-values when H_0 is true
M <- 1e4
pValues_H0 <- sapply(1:M, function(i) {
  x <- rnorm(n) # N(0, 1)
  ks.test(x, "pnorm")$p.value
})

# Simulation of p-values when H_0 is false -- the data does not
```

[6] When the sample size $n$ is large: $n \to \infty$.

[7] Which is $\lim_{n\to\infty} \mathbb{P}[d_n > D_n] = 1 - K(d_n)$, where $d_n$ is the observed statistic and $D_n$ is the random variable (A.1).

```
# come from a N(0, 1) but from a N(0, 1.5)
pValues_H1 <- sapply(1:M, function(i) {
  x <- rnorm(n, mean = 0, sd = sqrt(1.5)) # N(0, 1.5)
  ks.test(x, "pnorm")$p.value
})

# Comparison of p-values
par(mfrow = 1:2)
hist(pValues_H0, breaks = seq(0, 1, l = 20), probability = TRUE,
     main = expression(H[0]), ylim = c(0, 2.5))
abline(h = 1, col = 2)
hist(pValues_H1, breaks = seq(0, 1, l = 20), probability = TRUE,
     main = expression(H[1]), ylim = c(0, 2.5))
abline(h = 1, col = 2)
```

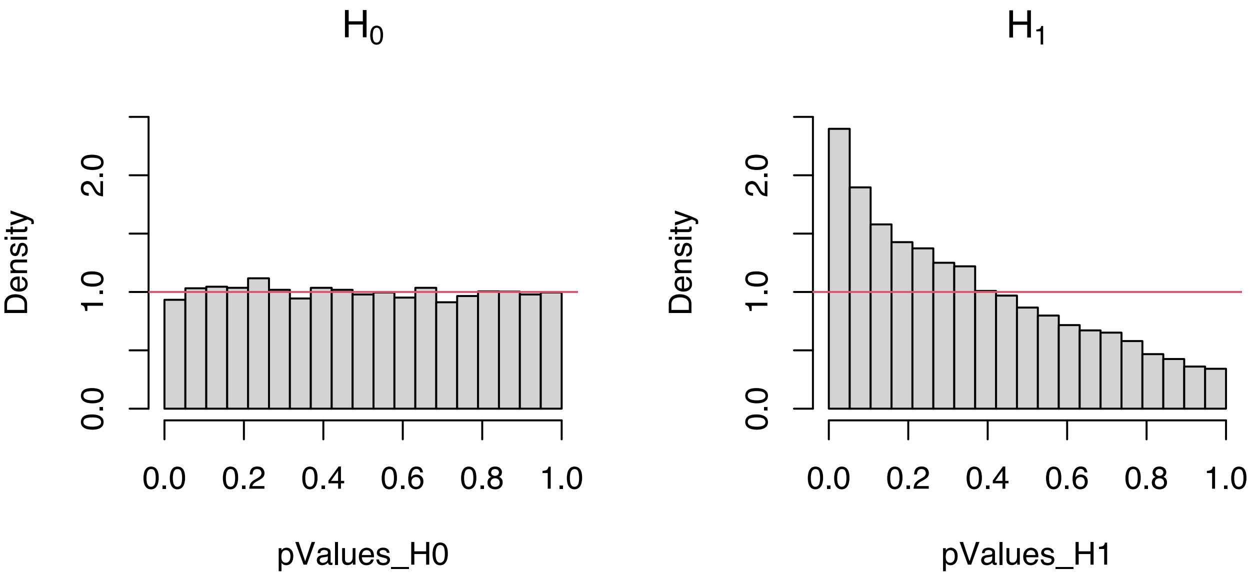


Figure A.1: Comparison of the distribution of $p$-values under $H_0$ and $H_1$ for the Kolmogorov–Smirnov test. Observe that the frequency of low $p$-values, associated with the rejection of $H_0$, grows when $H_0$ does not hold. Under $H_0$, the distribution of the $p$-values is uniform.

# *B*
# *Least squares and maximum likelihood estimation*

Least squares had a prominent role in linear models. In certain sense, this is strange. After all, it is a purely *geometrical argument* for fitting a plane to a cloud of points and therefore it seems to do not rely on any *statistical grounds* for estimating the unknown parameters $\boldsymbol{\beta}$.

However, as we will see, **least squares estimation is equivalent to maximum likelihood estimation** under the assumptions of the model seen in Section 2.3.[1] 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 $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$, due to the assumptions introduced in Section 2.3, we have that:

$$Y_i \mid (X_{i1} = x_{i1}, \ldots, X_{ip} = x_{ip}) \sim \mathcal{N}(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}, \sigma^2),$$

with $Y_1, \ldots, Y_n$ being independent conditionally on the sample of predictors. Equivalently stated in a compact matrix way (recall the notation behind (2.6)):

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

From these two equations we can obtain the log-likelihood function of $Y_1, \ldots, Y_n$ conditionally[2] on $\mathbf{X}_1, \ldots, \mathbf{X}_n$ as

$$\ell(\boldsymbol{\beta}) = \log\left(\phi(\mathbf{Y}; \mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I})\right) = \sum_{i=1}^n \log\left(\phi(Y_i; (\mathbf{X}\boldsymbol{\beta})_i, \sigma)\right). \tag{B.1}$$

Maximization of (B.1) with respect to $\boldsymbol{\beta}$ gives the maximum likelihood estimator $\hat{\boldsymbol{\beta}}_{\mathrm{ML}}$.

Now we are ready to show the next result.

**Theorem B.1.** *Under the assumptions i–iv in Section 2.3, the maximum likelihood estimate of $\boldsymbol{\beta}$ is the least squares estimate (2.7):*

$$\hat{\boldsymbol{\beta}}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \ell(\boldsymbol{\beta}) = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}\mathbf{Y}.$$

*Proof.* Expanding the first equality at (B.1) gives[3]

$$\ell(\boldsymbol{\beta}) = -\log\left((2\pi)^{n/2}\sigma^n\right) - \frac{1}{2\sigma^2}(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

[1] Normality is especially important here due to the squares present in the exponential of the normal pdf.

[2] Since we assume that the randomness is on the response only.

[3] Recall that $|\sigma^2\mathbf{I}|^{1/2} = \sigma^n$.

In order to differentiate with respect to $\boldsymbol{\beta}$, we use that, for two vector-valued functions $f$ and $g$:

$$\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A} \text{ and } \frac{\partial f(\mathbf{x})^\top g(\mathbf{x})}{\partial \mathbf{x}} = f(\mathbf{x})^\top \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} + g(\mathbf{x})^\top \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}.$$

Then, differentiating with respect to $\boldsymbol{\beta}$ and equating to zero gives

$$\frac{1}{\sigma^2}(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top \mathbf{X} = \frac{1}{\sigma^2}(\mathbf{Y}^\top \mathbf{X} - \boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{X}) = 0.$$

This means that optimizing $\ell$ does not require knowledge of $\sigma^2$! This is a very convenient fact that allows us to solve the above equation, yielding

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

□

A final comment on the benefits of relying on maximum likelihood estimation follows.

Maximum likelihood estimation is **asymptotically optimal** when estimating the unknown parameters of a model. This is a very appealing property that 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.

# C
# *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 $\{1, \ldots, J\}$. Given the predictors $X_1, \ldots, X_p$, *multinomial logistic regression* models the probability of each level $j$ of $Y$ by

$$\begin{aligned} p_j(\mathbf{x}) :=& \mathbb{P}[Y = j \mid X_1 = x_1, \ldots, X_p = x_p] \\ =& \frac{e^{\beta_{0j}+\beta_{1j}X_1+\cdots+\beta_{pj}X_p}}{1+\sum_{\ell=1}^{J-1} e^{\beta_{0\ell}+\beta_{1\ell}X_1+\cdots+\beta_{p\ell}X_p}} \end{aligned} \tag{C.1}$$

for $j = 1, \ldots, J-1$ and (for the last level $J$)

$$\begin{aligned} p_J(\mathbf{x}) :=& \mathbb{P}[Y = J \mid X_1 = x_1, \ldots, X_p = x_p] \\ =& \frac{1}{1+\sum_{\ell=1}^{J-1} e^{\beta_{0\ell}+\beta_{1\ell}X_1+\cdots+\beta_{p\ell}X_p}}. \end{aligned} \tag{C.2}$$

Note that (C.1) and (C.2) imply that $\sum_{j=1}^{J} p_j(\mathbf{x}) = 1$ and that there are $(J-1) \times (p+1)$ coefficients.[1] Also, (C.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 it is the tradition to choose the last one).

The multinomial logistic model has an interesting interpretation in terms of logistic regressions. Taking the quotient between (C.1) and (C.2) gives

$$\frac{p_j(\mathbf{x})}{p_J(\mathbf{x})} = e^{\beta_{0j}+\beta_{1j}X_1+\cdots+\beta_{pj}X_p} \tag{C.3}$$

for $j = 1, \ldots, J-1$. Therefore, applying a logarithm to both sides we have:

$$\log\left(\frac{p_j(\mathbf{x})}{p_J(\mathbf{x})}\right) = \beta_{0j} + \beta_{1j}X_1 + \cdots + \beta_{pj}X_p. \tag{C.4}$$

This equation is indeed very similar to (5.8). 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 (C.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

[1] $(J-1)$ intercepts and $(J-1) \times p$ slopes.

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 (C.3) gives also interpretation on the coefficients of the model since

$$p_j(\mathbf{x}) = e^{\beta_{0j}+\beta_{1j}X_1+\cdots+\beta_{pj}X_p} p_J(\mathbf{x}).$$

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 $X_1=\cdots=X_p=0$. 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 $Y=J$.
- $e^{\beta_{\ell j}}$, $\ell\geq 1$: is the **multiplicative** increment of the ratio between $p_j(\mathbf{x})/p_J(\mathbf{x})$ for an increment of one unit in $X_\ell=x_\ell$, provided that the remaining variables $X_1,\ldots,X_{\ell-1},X_{\ell+1},\ldots,X_p$ *do not change*. If $e^{\beta_{\ell j}}>1$ (equivalently, $\beta_{\ell j}>0$), then $Y=j$ becomes more likely than $Y=J$ for each increment in $X_j$. If $e^{\beta_{\ell j}}<1$ ($\beta_{\ell j}<0$), then $Y=j$ becomes less likely than $Y=J$.

The following code illustrates how to compute a basic multinomial regression employing the `nnet` package.

```r
# Data from the voting intentions in the 1988 Chilean national plebiscite
data(Chile, package = "carData")
summary(Chile)
##  region     population     sex           age        education       income          statusquo           vote
##  C :600   Min.   :  3750   F:1379   Min.   :18.00   P   :1107   Min.   :  2500   Min.   :-1.80301   A   :187
##  M :100   1st Qu.: 25000   M:1321   1st Qu.:26.00   PS  : 462   1st Qu.:  7500   1st Qu.:-1.00223   N   :889
##  N :322   Median :175000            Median :36.00   S   :1120   Median : 15000   Median :-0.04558   U   :588
##  S :718   Mean   :152222            Mean   :38.55   NA's:   11   Mean   : 33876   Mean   : 0.00000   Y   :868
##  SA:960   3rd Qu.:250000            3rd Qu.:49.00               3rd Qu.: 35000   3rd Qu.: 0.96857   NA's:168
##           Max.   :250000            Max.   :70.00               Max.   :200000   Max.   : 2.04859
##                                     NA's   :1                   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 <- nnet::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:
## nnet::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 easier interpretations
Chile$vote <- relevel(Chile$vote, ref = "N")
mod2 <- nnet::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:
## nnet::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 -> conservatism 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).

# D
# 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, that is, they **exclude** the **cases** in which there is at least one NA. This may be problematic in certain circumstances. 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           Month            Day
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000   Min.   : 1.0
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000   1st Qu.: 8.0
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000   Median :16.0
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993   Mean   :15.8
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000   Max.   :31.0
##  NA's   :37       NA's   :7

# 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 <- MASS::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 MASS::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 <- MASS::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 MASS::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 whether 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 im-

plemented very easily by using the `complete.cases` or `na.exclude` functions. However, an undesirable consequence is that we may lose a substantial amount of data and therefore the precision of the estimators will be 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 response or predictors).

2. **Remove predictors with much missing data**. This is another simple solution that is useful 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 artificial values that try to preserve the dataset structure:

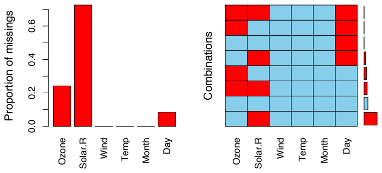
   - When the response is missing, we can use a **predictive model to predict the missing response**, then create a new fully-observed dataset containing the predictions instead of the missing values, and finally re-estimate the predictive model in this expanded dataset. This approach is attractive if most of the missing data is in the response.
   - 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). Another approach is to input sample means for the predictors, then use the reconstructed dataset to predict the missing responses. These and more sophisticated imputation methods, based on predictive models, are available within the `mice` package.[1] This approach is interesting if the data contains many `NA`s scattered in different predictors (hence a complete-cases analysis will be inefficient).

Let's put in practice these three approaches in the previous example.

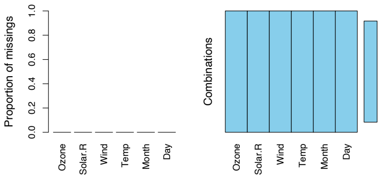
```
# The complete cases approach is the default in R
summary(lm(Ozone ~ ., data = airquality))
##
## Call:
## lm(formula = Ozone ~ ., data = airquality)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.790 -10.910  -2.249  10.960  33.246
##
## Coefficients:
##              Estimate 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)
```

[1] Check the available methods for `mice::mice` at `?mice`. There are specific methods for different kinds of variables (numeric, factor with two levels, factors of more than two levels) and fairly advanced imputation methods.

```
## 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           Month            Day       
##  Min.   : 8.00   Min.   : 13.0   Min.   : 6.300   Min.   :58.00   Min.   :5.00   Min.   : 1.00  
##  1st Qu.:13.00   1st Qu.: 61.5   1st Qu.: 9.575   1st Qu.:65.50   1st Qu.:5.00   1st Qu.:11.25  
##  Median :17.00   Median :178.5   Median :11.500   Median :71.50   Median :6.00   Median :16.50  
##  Mean   :28.21   Mean   :161.8   Mean   :11.887   Mean   :72.54   Mean   :6.75   Mean   :15.79  
##  3rd Qu.:36.25   3rd Qu.:255.0   3rd Qu.:14.300   3rd Qu.:79.50   3rd Qu.:9.00   3rd Qu.:20.25  
##  Max.   :96.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   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 for the most important combinations of NA's.
VIM::aggr(airquality)
```



```
VIM::aggr(airqualityNoNA)
```



```
# Stepwise regression without NA's -- no problem
modBIC1 <- MASS::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 <- MASS::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 with the variable Solar.R

# Input 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:
# - mice::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 could be random).
# - 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 -13.35375 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] -13.3537515  33.3839709 -15.1032941  -6.4344422  15.0745276  43.7185093  34.5128480   0.1488983  57.7500138
## [10]  62.1313290  30.1891953  70.4905697  72.0216949  75.6909337  38.1841717  43.5104926  57.9764970  69.7165869
## [19]  63.0884481  51.9374205  50.0400961  56.8792515  41.2844048  49.6257521  33.0354266  67.4490679  48.9905942
## [28]  54.0457773  51.9941898  50.1697328  46.1697595  71.3235772  49.9344726  39.0222991  26.9297719  77.0827188
## [37]  26.9508942
```

```
# 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          33          34          35          36
## -13.3537515  33.3839709 -15.1032941  -6.4344422  15.0745276  43.7185093  34.5128480   0.1488983  57.7500138  62.1313290
##          37          39          42          43          45          46          52          53          54          55
##  30.1891953  70.4905697  72.0216949  75.6909337  38.1841717  43.5104926  57.9764970  69.7165869  63.0884481  51.9374205
##          56          57          58          59          60          61          65          72          75          83
##  50.0400961  56.8792515  41.2844048  49.6257521  33.0354266  67.4490679  48.9905942  54.0457773  51.9941898  50.1697328
##          84         102         103         107         115         119         150
##  46.1697595  71.3235772  49.9344726  39.0222991  26.9297719  77.0827188  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  43.7185093  34.5128480   0.1488983  57.7500138
## [10]  62.1313290  30.1891953  70.4905697  72.0216949  75.6909337  38.1841717  43.5104926  57.9764970  69.7165869
## [19]  63.0884481  51.9374205  50.0400961  56.8792515  41.2844048  49.6257521  33.0354266  67.4490679  48.9905942
## [28]  54.0457773  51.9941898  50.1697328  46.1697595  71.3235772  49.9344726  39.0222991  26.9297719  77.0827188
## [37]  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      64 12.6   74     5   3
## 4    18      44 11.5   62     5  11
## 5    19      27 14.3   56     5   5
## 6    28     286 14.9   66     5  22
```

# E
# 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 analyzed 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 relatively simple solution for performing valid inference in a data-driven selected model is to **split the dataset in two parts**: one part for performing *model-selection* (selection of important variables and model structure; inside this part we could have two subparts for training and validation) and another for *fitting* the model and *carrying out inference* on the coefficients based on that fit. Obviously, this approach has the undesirable consequence of losing power in the estimation and inference parts due to the sample splitting, but it guarantees valid inference in a simple and general way.

The next simulation exercise exemplifies the previous remarks. Consider the following linear model

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \varepsilon, \tag{E.1}$$

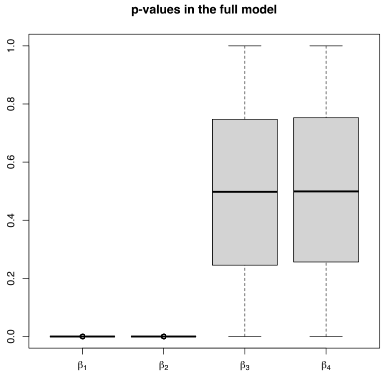
where $\beta_1 = \beta_2 = 1$, $\beta_3 = \beta_4 = 0$, and $\varepsilon \sim \mathcal{N}(0,1)$. The next chunk of code analyses the significances of the four coefficients for:

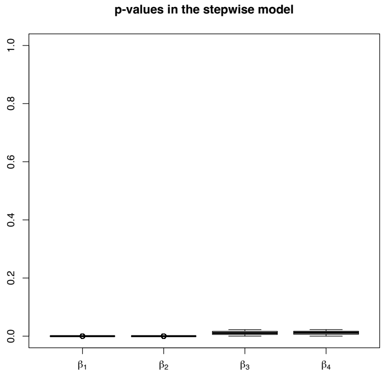
1. The model with **all the predictors**. The inferences for the coefficients are correct: the distribution of the $p$-values (`pvalues1`) is uniform whenever $H_0 : \beta_j = 0$ holds (for $j = 3, 4$) and concentrated around 0 when $H_0$ does not hold (for $j = 1, 2$).

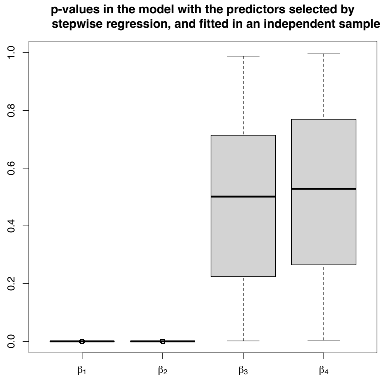
2. The model with **predictors selected by stepwise regression**. The inferences for the coefficients are biased: when $X_3$ and $X_4$ are included in the model is because they are highly significant for the given sample by mere chance. Therefore, the distribution of the $p$-values (`pvalues2`) is not 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 and the inference is correct.

```r
# 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 <- MASS::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 0.9774 0.9766

# Boxplots of significances
boxplot(pvalues1, names = expression(beta[1], beta[2], beta[3], beta[4]),
        main = "p-values in the full model", ylim = c(0, 1))


boxplot(pvalues2, names = expression(beta[1], beta[2], beta[3], beta[4]),
        main = "p-values in the stepwise model", ylim = c(0, 1))


boxplot(pvalues3, names = expression(beta[1], beta[2], 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))



# Test uniformity of the p-values associated with the coefficients that are 0
apply(pvalues1[, (p0 + 1):p], 2, function(x) ks.test(x, y = "punif")$p.value)
## [1] 0.4755784 0.4965309
apply(pvalues2[, (p0 + 1):p], 2, function(x) ks.test(x, y = "punif")$p.value)
## [1] 0 0
apply(pvalues3[, (p0 + 1):p], 2, function(x) ks.test(x, y = "punif")$p.value)
## [1] 0.8322453 0.1940926
```







# *References*

Aitchison, J. and Aitken, C. G. G. (1976). Multivariate binary discrimination by the kernel method. *Biometrika*, 63(3):413–420.

Allaire, J. J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2020). *rmarkdown: Dynamic Documents for R*. R package version 2.5.

Ashenfelter, O., Ashmore, D., and Lalonde, R. (1995). Bordeaux wine vintage quality and the weather. *CHANCE*, 8(4):7–14.

Box, G. E. P. and Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society, Series B (Methodological)*, 26(2):211–252.

Chacón, J. E. and Duong, T. (2018). *Multivariate Kernel Smoothing and its Applications*, volume 160 of *Monographs on Statistics and Applied Probability*. CRC Press, Boca Raton.

DasGupta, A. (2008). *Asymptotic Theory of Statistics and Probability*. Springer Texts in Statistics. Springer, New York.

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. *Journal of Environmental Economics and Management*, 5(1):81–102.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York.

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.

Kuhn, M. and Johnson, K. (2013). *Applied Predictive Modeling*. Springer, New York.

Li, Q. and Racine, J. S. (2007). *Nonparametric Econometrics*. Princeton University Press, Princeton.

Liu, R. Y. (1988). Bootstrap procedures under some non-i.i.d. models. *The Annals of Statistics*, 16(4):1696–1708.

Loader, C. (1999). *Local Regression and Likelihood*. Statistics and Computing. Springer, New York.

Marron, J. S. and Wand, M. P. (1992). Exact mean integrated squared error. *The Annals of Statistics*, 20(2):712–736.

McCullagh, P. and Nelder, J. A. (1983). *Generalized Linear Models*. Monographs on Statistics and Applied Probability. Chapman & Hall, London.

Miller, A. J. (1992). Algorithm AS 274: least squares routines to supplement those of Gentleman. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, 41(2):458–478.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117.

Nadaraya, E. A. (1964). On estimating regression. *Teoriya Veroyatnostei i ee Primeneniya*, 9(1):157–159.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076.

Peña, D. (2002). *Regresión y Diseño de Experimentos*. Alianza Editorial, Madrid.

Presidential Commission on the Space Shuttle Challenger Accident (1986). *Report of the Presidential Commission on the Space Shuttle Challenger Accident (Vols. 1 & 2)*. U.S. Goverment Printing Office, Washington.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. Vienna.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27(3):832–837.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

Seber, G. A. F. (1984). *Multivariate observations*. Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics. John Wiley & Sons, New York.

Seber, G. A. F. and Lee, A. J. (2003). *Linear regression analysis*. Wiley Series in Probability and Statistics. Wiley-Interscience, Hoboken.

Shao, J. (1993). Linear model selection by cross-validation. *Journal of the American Statistical Association*, 88(422):486–494.

Sheather, S. J. and Jones, M. C. (1991). A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society, Series B (Methodological)*, 53(3):683–690.

Siddhartha, R. D., Fowlkes, E. B., and Hoadley, B. (1989). Risk analysis of the space shuttle: Pre-Challenger prediction of failure. *Journal of the American Statistical Association*, 84(408):945–957.

Úcar, I. (2018). *Energy Efficiency in Wireless Communications for Mobile User Devices*. PhD thesis, Universidad Carlos III de Madrid.

Wand, M. P. and Jones, M. C. (1995). *Kernel Smoothing*, volume 60 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London.

Wasserman, L. (2004). *All of Statistics*. Springer Texts in Statistics. Springer-Verlag, New York.

Wasserman, L. (2006). *All of Nonparametric Statistics*. Springer Texts in Statistics. Springer-Verlag, New York.

Watson, G. S. (1964). Smooth regression analysis. *Sankhyā, Series A*, 26(4):359–372.

Wood, S. N. (2006). *Generalized Additive Models*. Texts in Statistical Science Series. Chapman & Hall/CRC, Boca Raton.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. The R Series. CRC Press, Boca Raton.

Xie, Y. (2020). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.30.

Xie, Y. and Allaire, J. J. (2020). *tufte: Tufte's Styles for R Markdown Documents*. R package version 0.8.

Yeo, I.-K. and Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959.

Zhao, P. and Yu, B. (2006). On model selection consistency of Lasso. *Journal of Machine Learning Research*, 7:2541–2563.