

# Machine Learning and Neural Networks

Dr. Hailiang Du

2023-06-01



# Contents

<b>Welcome</b>	<b>7</b>
<b>1 Introduction</b>	<b>9</b>
1.1 What is Machine Learning? . . . . .	9
1.2 Motivation . . . . .	10
1.3 What do we need? . . . . .	13
1.4 A Note on These Notes . . . . .	15
1.5 Key References . . . . .	15
<b>2 Definitions and Basic Concepts</b>	<b>17</b>
2.1 Types of data . . . . .	17
2.2 Supervised and Unsupervised Learning . . . . .	17
2.3 Classification and Regression . . . . .	18
2.4 Parametric vs non-parametric models . . . . .	19
2.5 Uncertainty . . . . .	20
2.6 Correlation and Causation . . . . .	21
<b>3 Linear Regression</b>	<b>25</b>
3.1 Simple Linear Regression: Introduction . . . . .	25
3.2 Simple Regression: Coefficient of Determination . . . . .	29
3.3 Simple Linear Regression: Assumptions . . . . .	35
3.4 Simple Linear Regression: Inference . . . . .	46
3.5 Simple Linear Regression: Confidence and Prediction intervals . . . . .	53
3.6 Multiple Linear Regression: Introduction . . . . .	58
<b>4 Model Assessment and Selection</b>	<b>69</b>
4.1 In-sample vs out-of-sample . . . . .	69
4.2 Cross-Validation . . . . .	70
4.3 Overfitting vs Underfitting . . . . .	71
4.4 Bias Variance Trade-off . . . . .	73
4.5 Model selection for Linear regression . . . . .	76
4.6 Model Selection Criteria . . . . .	77
<b>5 Model Search Methods</b>	<b>81</b>
5.1 Best Subset Selection . . . . .	81
5.2 Forwards Stepwise Selection . . . . .	82

5.3 Backwards Stepwise Selection . . . . .	83
5.4 Practical Demonstration . . . . .	84
<b>6 Shrinkage Methods</b>	<b>103</b>
6.1 Motivation . . . . .	103
6.2 Ridge Regression . . . . .	107
6.3 Lasso Regression . . . . .	117
6.4 Choosing $\lambda$ . . . . .	125
<b>7 Principal Component Analysis</b>	<b>135</b>
7.1 Reminder . . . . .	135
7.2 Linear approximation . . . . .	135
7.3 Decomposing variance . . . . .	139
7.4 Scaling . . . . .	141
7.5 Data compression (Dimension reduction) . . . . .	142
7.6 Principal Component Regression . . . . .	147
<b>8 Polynomial Regression</b>	<b>155</b>
8.1 The Assumption of Linearity . . . . .	155
8.2 Polynomial Regression Models . . . . .	156
8.3 Practical Demonstration . . . . .	159
<b>9 Step Functions</b>	<b>169</b>
9.1 Step Functions . . . . .	169
9.2 Practical Demonstration . . . . .	170
<b>10 Splines</b>	<b>175</b>
10.1 Regression Splines . . . . .	175
10.2 Practical Demonstration . . . . .	180
10.3 Natural Splines . . . . .	184
10.4 Practical Demonstration . . . . .	185
10.5 Smoothing Splines . . . . .	188
10.6 Practical Demonstration . . . . .	190
<b>11 Generalised Additive Models</b>	<b>195</b>
11.1 Review . . . . .	195
11.2 GAMs . . . . .	195
11.3 Practical Demonstration . . . . .	198
<b>12 Logistic Regression</b>	<b>201</b>
12.1 Motivation . . . . .	201
12.2 Simple Logistic Regression . . . . .	206
12.3 Logistic regression with several variables . . . . .	209
12.4 Interpreting coefficients . . . . .	209
12.5 Significance . . . . .	209
<b>13 Tree-based Models</b>	<b>211</b>
13.1 Classification and Regression Trees . . . . .	212

<b>CONTENTS</b>	<b>5</b>
-----------------	----------

13.2 Bagging . . . . .	224
13.3 Random Forests . . . . .	226
13.4 Boosting . . . . .	227
13.5 Practical Demonstration . . . . .	228

## **I Problem Sheets** **237**

<b>Problem Sheets</b>	<b>239</b>
Problem Sheet 1 . . . . .	239
Problem Sheet 1 Solution . . . . .	241
Problem Sheet 2 . . . . .	244
Problem Sheet 2 Solution . . . . .	247
Problem Sheet 3 . . . . .	251
Problem Sheet 3 Solution . . . . .	253
Problem Sheet 4 . . . . .	258
Problem Sheet 4 Solution . . . . .	260

## **II Practical Classes** **265**

<b>Practical Class Sheets 1</b>	<b>267</b>
The <i>faithful</i> data set . . . . .	267
Fitting the simple linear regression model . . . . .	268
Inference on the coefficients . . . . .	272
Estimation and prediction . . . . .	274
Residual Analysis . . . . .	275

<b>Practical Class Sheets 2</b>	<b>279</b>
Body Fat and Body Measurements Data . . . . .	279
Simple Linear Regression . . . . .	285
Multiple Linear Regression . . . . .	288
Variable Selection . . . . .	293

<b>Practical Class Sheets 3</b>	<b>299</b>
Initial Data Analysis . . . . .	299
Ridge Regression . . . . .	302
Lasso Regression . . . . .	303
Principal Component Analysis . . . . .	307
Principal Component Regression . . . . .	310
Predictive Performance of Methods . . . . .	311

<b>Practical Class Sheets 4</b>	<b>315</b>
Polynomial and Step-wise Function Regression . . . . .	315
Splines . . . . .	320
GAMs . . . . .	322
Boston Data . . . . .	324
Logistic Regression . . . . .	326



# Welcome

Welcome to the material for the first half of the Machine Learning and Neural Networks module (MATH3431) at Durham University. These pages consist of relevant lecture notes will be updated as the course progresses.

I would recommend that you use the html version of these notes (they have been designed for use in this way), however, there is also a pdf version of these notes.

In this first half of the module (Michaelmas Term), we will be focusing on “Machine Learning” rather than “Neural Networks”. We will i) go over the fundamental concepts of Machine (Statistical) Learning, ii) study the classic linear models, iii) explore models beyond Linearity and finally iv) look into the simpler yet powerful tree-based models.

If you would like to contact me regarding any of the material in this course, then my email address is [hailiang.du@durham.ac.uk](mailto:hailiang.du@durham.ac.uk)

Credits: Thanks to Dr Tahani Coolen-Maturi, Dr Samuel Jackson and Dr Emmanuel Ogundimu for their kindly providing the materials for this module contents.



# Chapter 1

## Introduction

### 1.1 What is Machine Learning?

Is ML the same as Statistics? Is it Data Science? Is it its own thing? Is it just a buzzword which is good for employment? Just what is Machine Learning?

Well, I don't think there's a single concrete definition for Machine Learning (ML), but let's have a look at several different ones....

A popular definition: "*A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*". Mitchell [1997]

This means there are many different kinds of machine learning, depending on the nature of the task T we wish the system to learn, the nature of the performance measure P we use to evaluate the system, and the nature of the training signal or experience E we give it. Murphy [2022]

Another definition: "*Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy*". Hurwitz and Kirsch [2018]

And one more definition: "*A set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!).*". Murphy [2012]

Well, it seems that the general consensus is that ML is about getting the machine (most likely a computer in some sense) to learn as much as possible, with the least human interaction possible. This may be an admirable aim, however, I would say that there are elements of human interaction and interpretation of results that machines can't do. Machines and humans (bringing an analytical mind as well as the know-how to use the machines) must together perform Statistics, Data Science and Machine Learning in my opinion.

With regards to this course specifically, my half of the course could be seen as Statistical Learning, we view machine learning from a probabilistic perspective and apply statistical tools to machine learning.

I will leave you to decide for yourself about these various names for our science as the course progresses! What I do know is that we are going to cover some important statistical tools if you want to get into “Machine Learning”!

## 1.2 Motivation

We are currently living in a data-driven world, where the amount of digitally-collected data grows exponentially. Being a powerful tool, machine learning derives insights from this tremendous amount of data and helps solve many practical problems.

People who are exploring machine learning as a field to shift their careers or people who are just curious about why there is such a buzz of machine learning all over places often have one burning question in their mind – what all possible things can one achieve with machine learning? Well, the short answer is – that the possibility is endless and one’s creativity is the only limit.

Here are a few machine learning examples that are in popular use in today’s world.

### Machine Learning is an integral part of our daily life

Google search engine, now processes over 40,000 searches per second on average and billions of searches daily and yet throws relevant search results within seconds. Did you ever freak out at how Google gives such accurate autocomplete prompts in the search boxes as if it is reading our minds about search context?

Google maps predict which route will be faster and shows such an accurate ETA of reaching the destination.

Ever noticed that spam/junk folder in your university mailbox or many other email providers, for example Gmail, and did you ever open it to see so many spam mails indeed.

Various language translators and automatic subtitles are becoming better and better with time.

Youtube and Tiktok suggestions, which keep people hooked for endless hours. Amazon and Netflix are so good at recommending what you may like.

Yes, I think you are getting my point now, it is machine learning that is behind all the magic.

### Machine Learning in industrial uses

Industries are adopting machine learning across the world.

Banks are using machine learning to detect frauds, pass loans, and maintain investment portfolios.

Hospitals are using machine learning to detect and diagnose diseases with more accuracy than actual doctors.

Companies are running ad campaigns targeted to specific customer segments and managing their supply chain and inventory control using machine learning.

Enterprise chatbots are now the next big thing, with companies adopting them to interface with customers.

Online retailers are building their recommendation systems with more accuracy than before using the latest advancements in machine learning.

Self Driving Cars are the latest fascination that has caught the imaginations of companies like Google, Uber, and Tesla who are investing heavily into this future vision using next-generation machine learning technology.

These are just a few of the popular use cases and are just the tip of the iceberg. Machine learning can be used in the most creative ways to solve social, economical, and business problems. The possibility is just endless.

## Machine Learning – The Artist

This is the coolest part- with so many advancements in Deep Learning a subset field with machine learning things are getting to the next level of reality. Now we are able to produce artwork using machine learning. These are some of the astonishing artworks.

### Deepfakes

Jason Allen, a video game designer in Pueblo, Colorado, spent roughly 80 hours working on his entry to the Colorado State Fair's digital arts competition. Judges awarded his A.I.-generated work ("Théâtre D'opéra Spatial") first place, which came with a \$300 prize.



The beautiful model-looking faces below do not exist and have been generated by machine learning's imagination.



## Neural Style Transfer

Neural style transfer is a cool technique in which a style of an image or artwork is transferred to another image.



## Music Composer

Nowadays, Machine Learning based apps (for example <https://www.aiva.ai/>) can assist you to create various styles of AI-generated music.

Check out this soundtrack music ([https://www.youtube.com/watch?v=Emidxpkyk6o&ab\\_channel=Aiva](https://www.youtube.com/watch?v=Emidxpkyk6o&ab_channel=Aiva)), there is no reason to believe that it was not generated by humans!!

## Writer

This machine learning read a lot of literature from Shakespeare and then produced a piece of text that looked not only meaningful but also Shakespearean.

```

PANDARUS:
Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

```

## Machine Learning – The Gamer

In 2011, IBM's machine learning system Watson participated in a popular TV game show Jeopardy – which competed and defeated the world's best Jeopardy human champions.

In 2016, Deepmind's machine learning system AlphaGo played Go with Lee Sudol (the best player and champion of Go) and defeated him. This feat is considered to be one of the greatest achievements in the field of machine learning.

Machine learning-based AI-player has been outcompeting top human players in various real-time strategy games, such as StarCraft and League of Legends.

## In This Course...

In the first term (Michaelmas term) of this course, we will cover the following topics along with some practical exercises.

- Fundamental concepts of Statistical (Machine) Learning
- Linear models (Logistic Regression, Linear Regression)
- Beyond Linearity (Polynomial Regression, Step Functions, Splines)
- Tree-based methods (Classification and regression trees, Random forests, Boosting)

You may find these topics seem to be nowhere near those fascinating application examples listed above. Well, they are in fact the cornerstone of the more advanced machine learning algorithms including neural networks.

## 1.3 What do we need?

What do we need to achieve “successful” Machine (Statistical) Learning? In my opinion, it requires the following four pillars.

## 1. Sufficient Data

If you ask any data scientist how much data is needed for machine learning, you'll most probably get either "It depends" or "The more, the better." And the thing is, both answers are correct.

It really depends on the type of project you're working on, and it's always a great idea to have as many relevant and reliable examples in the datasets as you can get to receive accurate results. But the question remains: how much is enough?

General speaking, more complex algorithms always require a larger amount of data. The more features (the number of input parameters), model parameters, and variability of the expected output it should take into account, the more data you need. For example, you want to train the model to predict housing prices. You are given a table where each row is a house, and the columns are the location, the neighborhood, the number of bedrooms, floors, bathrooms, etc., and the price. In this case, you train the model to predict prices based on the change of variables in the columns. And to learn how each additional input feature influences the input, you'll need more data examples.

## 2. Sufficient Computational Resources

There are four steps for conducting machine learning, all require sufficient computational resources:

- Preprocessing input data
- Training the machine learning model
- Storing the trained machine learning model
- Deployment of the model

Among all these, training the machine learning model is the most computationally intensive task. For example, training a neural network often requires intensive computational resources to conduct an enormous amount of matrix multiplications.

## 3. Performance metrics

Performance metrics are a part of every machine learning pipeline. They tell you if you're making progress, and put a number on it. All machine learning models, whether it's linear regression or neural networks, need a metric to judge performance.

Many of the machine learning tasks can be broken down into either Regression or Classification. There are dozens of metrics for both problems, for example, Root Mean Squared Error for Regression problem and Accuracy for Classification problem. Choosing a "proper" performance could be crucial to the whole machine learning process. We must carefully choose the metrics for evaluating machine learning performance because

- How the performance of ML algorithms is measured and compared will be dependent entirely on the metric you choose.
- How you weigh the importance of various characteristics in the result will be influenced completely by the metric you choose.

We will explore some of the popular performance metrics for machine learning in the coming lectures.

Note most of the performance metrics (especially in supervised learning) require verification/labelled data. For regression problems, we usually require an observed response variable (verification) in order to calculate performance metrics. And for the classification problems, we usually require labelled data. In machine learning, data labelling is the process of identifying raw data (images, text files, videos, etc.) and adding one or more meaningful and informative labels to provide context so that a machine learning model can learn from it. For example, labels might indicate whether a photo contains a bird or car, which words were uttered in an audio recording, or if an x-ray contains a tumor.

## 4. Proper Machine Learning Algorithm

There are so many algorithms that it can feel overwhelming when algorithm names are thrown around. In this course, we will look into some classic machine learning algorithms including for example linear models, tree-based models and neural networks. Knowing where the algorithms fit is much more important than knowing how to apply them. For example, applying a linear model to the data generated by a nonlinear underlying system is obviously not a good idea.

### 1.4 A Note on These Notes

Since this is a third-year course, you are encouraged to read about it widely, expanding your knowledge beyond the notes contained here. That's because that is precisely what these are - Notes! They are written to complement the lectures and give you a flavour of the topics covered in this course. They are not meant to be a comprehensive textbook, covering every detail that will be necessary to become a well-trained Statistician, Data Scientist or Machine Learner - that requires reading around! ...and practice!

### 1.5 Key References

The key references are the following (All of the recommended books are freely and legally available online):

- James et al. [2013] - *An Introduction to Statistical Learning*, by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. This book is relatively easy to follow and doesn't require strong maths background. There are quite a few online resources about this book available [here](#).
- Faraway [2009] - Linear Models with R, by Julian Faraway. This book gives a thorough introduction to linear models and their applications to make predictions and explaining the relationship between the response and the predictors. Understanding linear models are crucial to a broader competence in the practice of machine learning. <https://search-ebscohost-com.ezphost.dur.ac.uk/login.aspx?direct=true&AuthType=cookie,ip,authns&db=nlebk&AN=1910500&zsite=ehost-live>

- Murphy [2012] - Machine Learning : A Probabilistic Perspective, by Kevin P. Murphy. This book gives a comprehensive introduction to machine learning and offers a comprehensive and self-contained introduction to the field of machine learning, based on a unified, probabilistic approach. The book is written in an informal, accessible style, complete with pseudo-code for the most important algorithms. <https://ebookcentral.proquest.com/lib/durham/detail.action?docID=3339490>
- Murphy [2022] - Probabilistic Machine Learning : An Introduction, by Kevin P. Murphy. This book is similar to Murphy [2012] which covers the most common types of machine learning but from a probabilistic perspective. This book offers a more detailed and up-to-date introduction to machine learning including relevant mathematical background, basic supervised learning as well as more advanced topics. <https://ebookcentral.proquest.com/lib/durham/detail.action?docID=6884472>

# Chapter 2

## Definitions and Basic Concepts

### 2.1 Types of data

Data sets can consist of two types of data:

- Qualitative (categorical) data consist of attributes, labels, or nonnumerical entries. e.g. name of cities, gender etc.
- Quantitative data consist of numerical measurements or counts. e.g. heights, age, temperature. Quantitative data can be distinguished as:
  - Discrete data result when the number of possible values is either a finite number or a “countable” number. e.g. the number of phone calls you received on any given day.
  - Continuous data result from infinitely many possible values that correspond to some continuous scale that covers a range of values without gaps, interruptions, or jumps. e.g. height, weight, sales and market shares.

### 2.2 Supervised and Unsupervised Learning

Every machine learning task can be broken down into either supervised learning or unsupervised learning.

*Supervised learning* involves building a statistical model for predicting or estimating an output based on one or more inputs.

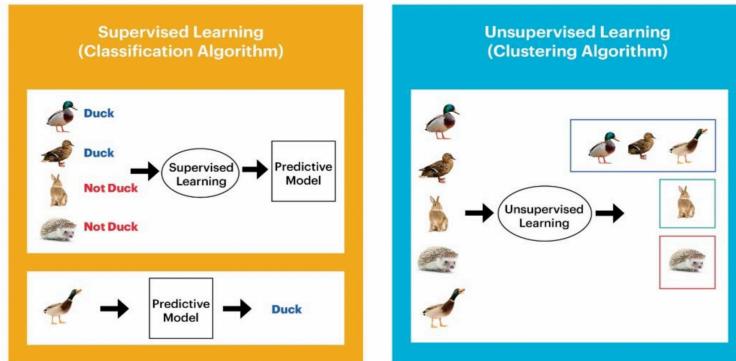
Supervised learning aims to learn a mapping  $f$  from inputs  $\mathbf{x} \in \mathcal{X}$  to outputs  $\mathbf{y} \in \mathcal{Y}$ . The inputs  $\mathbf{x}$  are also called the `_feature_s`, `_covariate_s`, or `_predictor_s`; this is often a fixed-dimensional vector of numbers, such as the height and weight of a person, or the pixels in an image. In this case,  $\mathcal{X} = \mathbb{R}^D$ , where  $D$  is the dimensionality of the vector (i.e., the number of input features). The output  $\mathbf{y}$  is also known as the *label*, *target*, or *response*. In supervised learning, we assume that each input example  $\mathbf{x}$  in the training archive has an associated set of output targets  $\mathbf{y}$ , and our goal is to learn the input-output mapping  $f$ .

The task of *Unsupervised learning* is to try to “make sense of” data, as opposed to just learning a mapping. That is, we just get observed “inputs”  $\mathbf{x} \in \mathcal{X}$  without any corresponding “outputs”  $\mathbf{y} \in \mathcal{Y}$ .

From a probabilistic perspective, we can view the task of unsupervised learning as fitting an unconditional model of the form  $p(\mathbf{x})$ , which can generate new data  $\mathbf{x}$ , whereas supervised learning involves fitting a conditional model,  $p(\mathbf{y}|\mathbf{x})$ , which specifies (a distribution over) outputs given inputs.

Unsupervised learning avoids the need to collect large labelled datasets for training, which can often be time-consuming and expensive (think of asking doctors to label medical images). Unsupervised learning also avoids the need to learn how to partition the world into often arbitrary categories. Finally, unsupervised learning forces the model to “explain” the high-dimensional inputs, rather than just the low-dimensional outputs. This allows us to learn richer models of “how the world works” and discover “interesting structure” in the data.

Here is a simple example that shows the difference between supervised learning and unsupervised learning.



Many classical machine (statistical) learning methods such as linear regression and logistic regression as well as more advanced approaches such as random forests and boosting operate in the supervised learning domain. Most part of this course will be devoted to this setting. For unsupervised learning, we will cover one of the classic linear approaches, principal component analysis, in the first term of the course.

## 2.3 Classification and Regression

Supervised learning can be further divided into two types of problems: Classification and Regression.

- Classification

*Classification* algorithms are used when the output variable is categorical

The task of the classification algorithm is to find the mapping function to map the input( $\mathbf{x}$ ) to the discrete output( $\mathbf{y}$ ).

Example: The best example to understand the Classification problem is Email Spam Detection. The model is trained on the basis of millions of emails on different param-

eters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.

- Regression

*Regression* algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc.

Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of Market Trends, prediction of House prices, etc.

The task of the Regression algorithm is to find the mapping function to map the input variable(x) to the continuous output variable(y).

Example: Suppose we want to do weather forecasting, so for this, we will use the Regression algorithm. In weather prediction, the model is trained on the past data, and once the training is completed, it can easily predict the weather for future days.

## 2.4 Parametric vs non-parametric models

Machine learning models can be parametric or non-parametric. *Parametric* models are those that require the specification of some parameters, while *non-parametric* models do not rely on any specific parameter settings.

- Parametric models

Assumptions about the form of a function can ease the process of machine learning. Parametric models are characterized by the simplification of the function to a known form. A parametric model is a learner that summarizes data through a collection of parameters, . These parameters are of a fixed size. This means that the model already knows the number of parameters it requires, regardless of its data.

As an example, let's look at a simple linear regression model (we will investigate this model thoroughly in the following chapter) in the functional form:

$$y = b_0 + b_1 x + \epsilon$$

where  $b_0$  and  $b_1$  are model parameters and  $\epsilon$  reflects the model error. For such a model, feeding in more data will impact the value of the parameters in the equation above. It will not increase the complexity of the model. And note this model form assumes a linear relationship between the input  $x$  and response  $y$ .

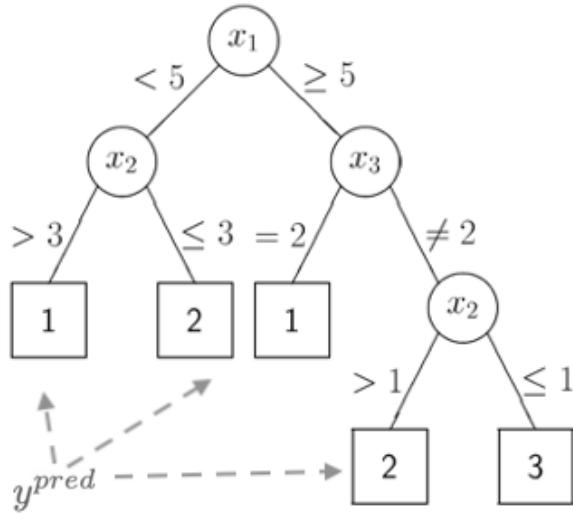
- Non-parametric models

Non-parametric models do not make particular assumptions about the kind of mapping function and do not have a specific form of the mapping function. Therefore they have the freedom to choose any functional form from the training data.

One might think that non-parametric means that there are no parameters. However, this is NOT true. Rather, it simply means that the parameters are (not only) adjustable

but can also change. This leads to a key distinction between parametric and non-parametric algorithms. We mentioned that parametric algorithms have a fixed number of parameters, regardless of the amount of training data. However, in the case of non-parametric ones, the number of parameters is dependent on the amount of training data. Usually, the more training data, the greater the number of parameters. A consequence of this is that non-parametric models may take much longer to train.

The classification and regression tree (which will be introduced later in the course) is an example of a non-parametric model. It makes no distributional assumptions on the data and increasing the amount of training data is very likely to increase the complexity of the tree.



## 2.5 Uncertainty

Arguably, we are living in a deterministic universe without considering quantum theory, which means the underlying system that generates the data is deterministic. One might think that the outcome of rolling dice is purely random. Actually, this is NOT true. If one can observe every related detail including for example the smoothness of the landing surface, the air resistance and the exact direction and velocity of the dice once it left the hand, one shall be able to predict the outcome precisely.

If the underlying system is deterministic, why are we studying machine learning from a probabilistic perspective, for example, we use the form  $p(\mathbf{y}|\mathbf{x})$  in supervised learning. The answer is *uncertainty*. There are two major sources of uncertainty.

- Data uncertainty

If there are data involved, there is uncertainty induced by measurement error. You might think that well-made rulers, clocks and thermometers should be trustworthy, and give the right answers. But for every measurement - even the most careful - there is always a margin of “error”. To account for measurement error, we often use a noise model. For example

$$x = \tilde{x} + \eta$$

where  $x$  is the observed value,  $\tilde{x}$  reflects the “true” value and  $\eta$  is the noise model, for example assuming  $\eta \sim N(0, 1)$ .

- Model discrepancy

Another major source of uncertainty in modelling is due to model discrepancy. Recall the famous quote from George Box, “All models are wrong but some models are useful”. There is always a difference between the (imperfect) model used to approximate reality, and reality itself; this difference is termed *model discrepancy*. Therefore whatever model we construct, we usually include an error term to account for uncertainty due to model discrepancy. For example the  $\epsilon$  in the linear regression model we saw before. And we often assume  $\epsilon$  is a random draw from some distribution.

$$y = b_0 + b_1 x + \epsilon$$

## 2.6 Correlation and Causation

Machine learning is not only widely used for prediction but also widely used for inference. One of the main tasks of inference is to investigate, evaluate and understand the relationship between (for example the predictor and response) variables. Let’s look at the two of the most common concepts, correlation and causation, closely related to this task.

While causation and correlation can exist simultaneously, correlation does not imply causation. *Causation* means one thing causes another—in other words, action A causes outcome B. On the other hand, *correlation* is simply a relationship where action A relates to action B—but one event doesn’t necessarily cause the other event to happen.

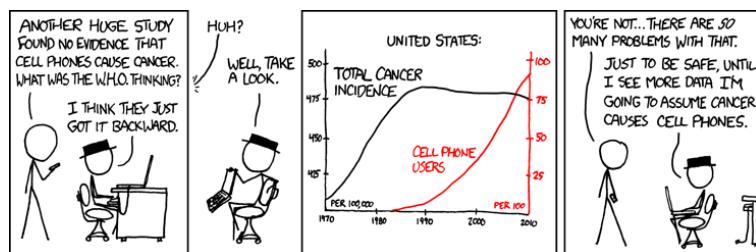
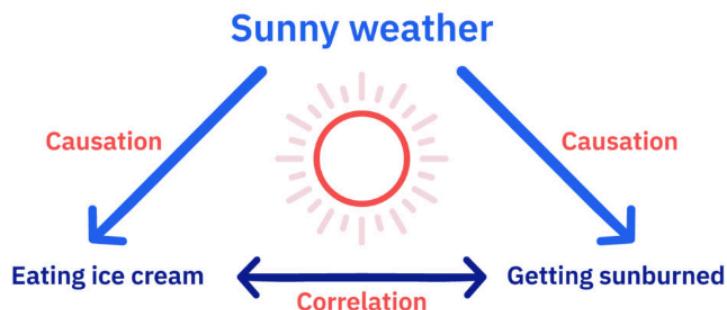


Figure 2.1: <https://xkcd.com/925/>

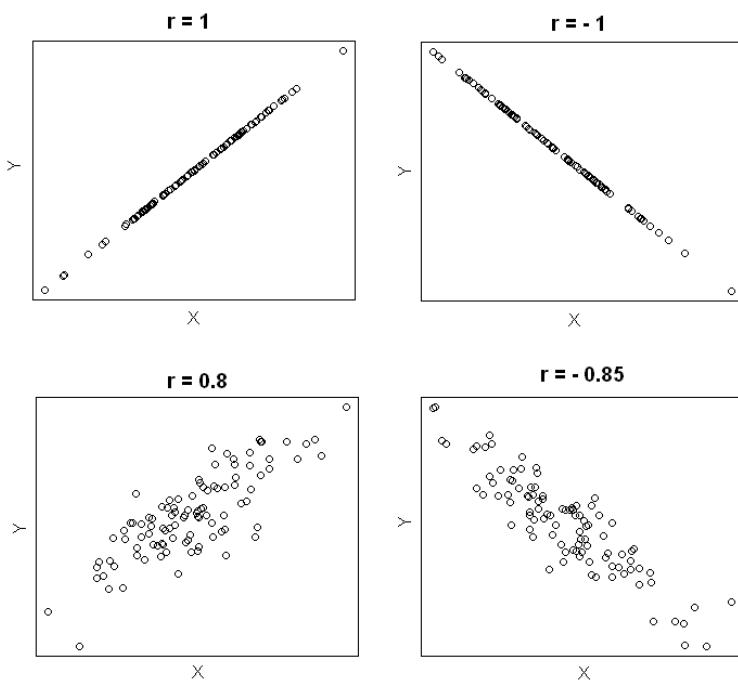


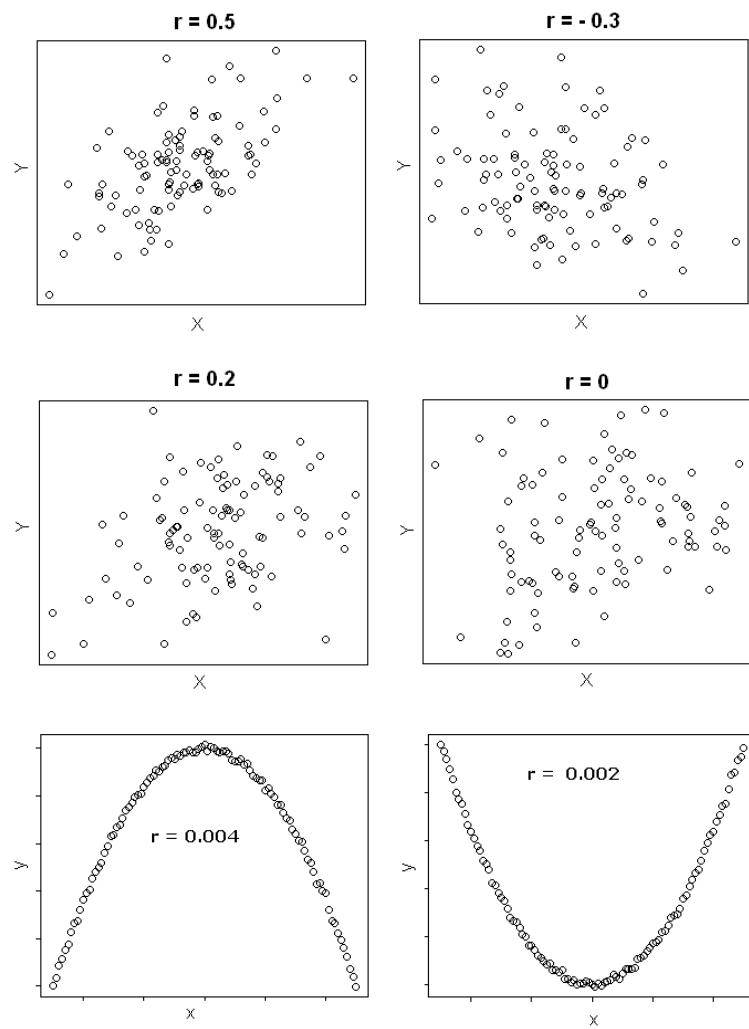
## Pearson correlation coefficient

*Pearson correlation coefficient*( $r$ ), often called sample correlation coefficient, is a measure of the strength and the direction of a **linear relationship** between two variables in the sample,

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2} \sqrt{\sum(y_i - \bar{y})^2}}$$

where  $r$  always lies between -1 and 1. Values of  $r$  near -1 or 1 indicate a strong linear relationship between the variables whereas values of  $r$  near 0 indicate a weak linear relationship between variables. If  $r$  is zero the variables are linearly uncorrelated, that is there is no linear relationship between the two variables.





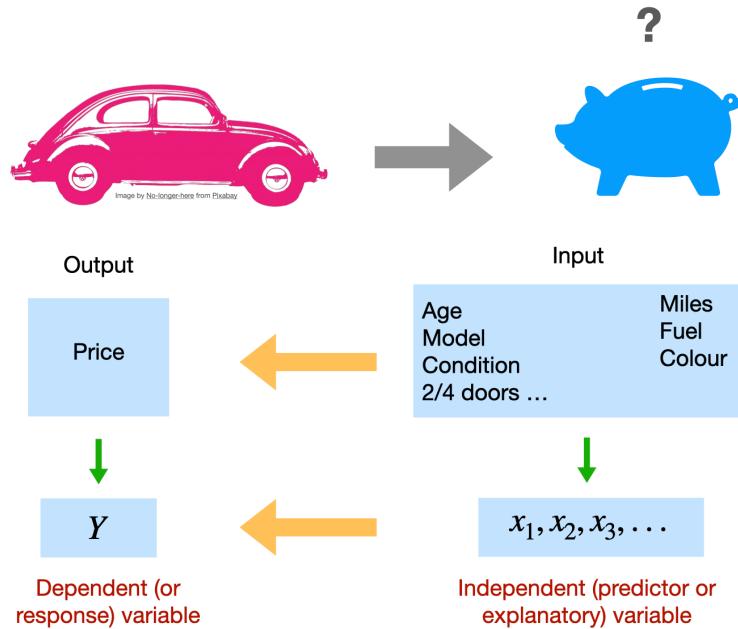


# Chapter 3

## Linear Regression

### 3.1 Simple Linear Regression: Introduction

Motivation: Predicting the Price of a Used Car



The table below displays data on Age (in years) and Price (in hundreds of dollars) for a sample of cars of a particular make and model. Weiss [2012]

Price ( $y$ )	Age ( $x$ )
85	5
103	4
70	6
82	5
89	5
98	5
66	6

Price ( $y$ )	Age ( $x$ )
95	6
169	2
70	7
48	7

Enter the data in R.

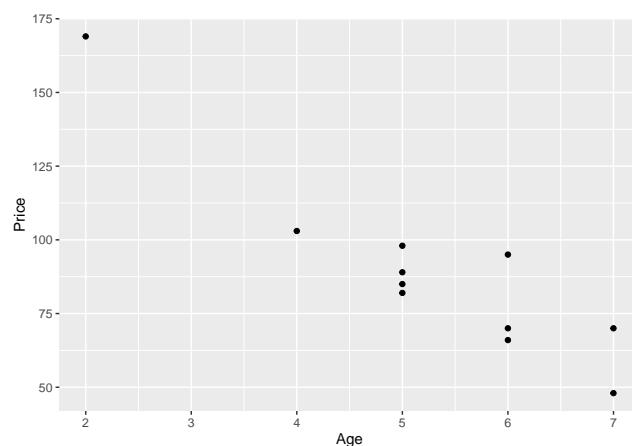
```
Price<-c(85, 103, 70, 82, 89, 98, 66, 95, 169, 70, 48)
Age<- c(5, 4, 6, 5, 5, 6, 6, 2, 7, 7)
carSales<-data.frame(Price, Age)
str(carSales)
```

```
## 'data.frame': 11 obs. of 2 variables:
## $ Price: num 85 103 70 82 89 98 66 95 169 70 ...
## $ Age : num 5 4 6 5 5 6 6 2 7 ...
cor(Age, Price, method = "pearson")
```

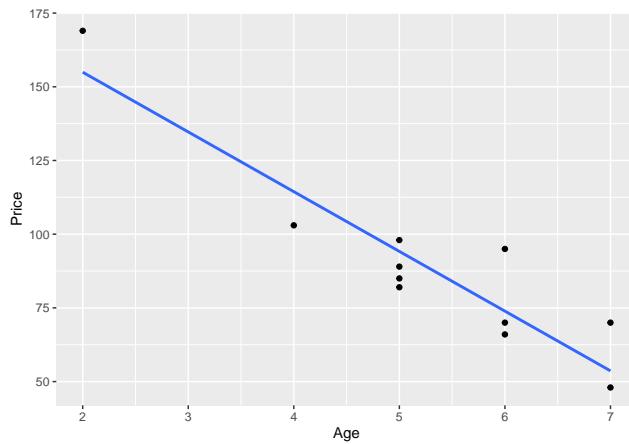
## [1] -0.9237821

Scatterplot: Age vs. Price

```
library(ggplot2)
ggplot(carSales, aes(x=Age, y=Price)) + geom_point()
```



```
ggplot(carSales, aes(x=Age, y=Price)) +
  geom_point()+
  geom_smooth(method=lm, formula= y~x, se=FALSE)
```



Obviously, we see a linear relationship between Age and Price.

## Simple linear regression Model

### Simple linear regression (population)

$$Y = \beta_0 + \beta_1 x + \epsilon$$

In our example:

$$\text{Price} = \beta_0 + \beta_1 \text{Age} + \epsilon$$

### Simple linear regression (sample)

$$\hat{y} = b_0 + b_1 x$$

where the coefficient  $\beta_0$  (and its estimate  $b_0$  or  $\hat{\beta}_0$ ) refers to the  $y$ -intercept or simply the intercept or the constant of the regression line, and the coefficient  $\beta_1$  (and its estimate (and its estimate  $b_1$  or  $\hat{\beta}_1$ ) refers to the slope of the regression line.

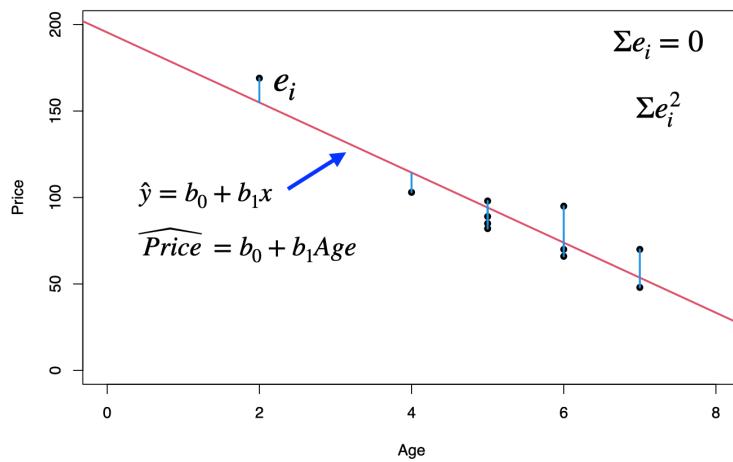
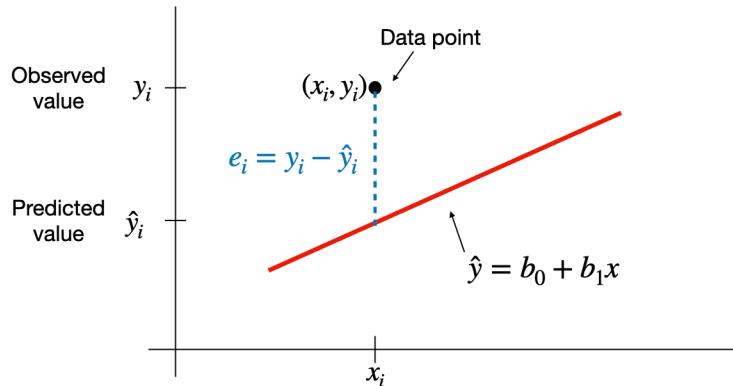
## The Least Squares criterion

Recall that one of the four pillars we need for successful machine learning is performance metric. In order to estimate the simple linear regression model parameters (here coefficients  $\beta_0$  and  $\beta_1$ ), we need to find the right performance metric. Here comes the least squares.

- The *least squares criterion* is that the line that best fits a set of data points is the one having the smallest possible sum of squared errors. The ‘errors’ are the vertical distances of the data points to the line.
- The *regression line* is the line that fits a set of data points according to the least squares criterion.
- The *regression equation* is the equation of the regression line.
- The regression equation for a set of  $n$  data points is  $\hat{y} = b_0 + b_1 x$ , where

$$b_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \quad \text{and} \quad b_0 = \bar{y} - b_1 \bar{x}$$

- $y$  is the dependent variable (or response variable) and  $x$  is the independent variable (predictor variable or explanatory variable).
- $b_0$  is called the  $y$ -intercept and  $b_1$  is called the *slope*.



## SSE and the standard error

This least square regression line minimizes the error sum of squares

$$SSE = \sum e_i^2 = \sum (y_i - \hat{y}_i)^2$$

The standard error of the estimate,  $s_e = \sqrt{SSE/(n-2)}$ , indicates how much, on average, the observed values of the response variable differ from the predicted values of the response variable.

## Prediction

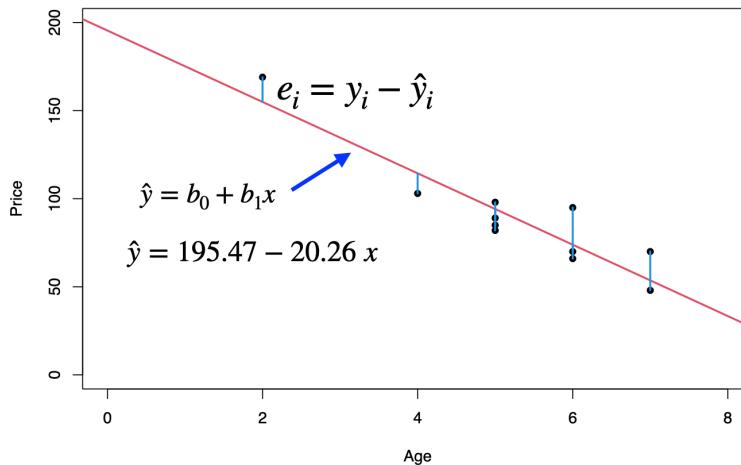
```
# simple linear regression
reg<-lm(Price~Age)
print(reg)
```

```
##
## Call:
```

```
## lm(formula = Price ~ Age)
##
## Coefficients:
## (Intercept)          Age
##       195.47        -20.26
```

To predict the price of a 4-year-old car ( $x = 4$ ):

$$\hat{y} = 195.47 - 20.26(4) = 114.43$$



Note the prediction  $\hat{y}$  has not accounted for any uncertainty yet.

## 3.2 Simple Regression: Coefficient of Determination

### Used cars example

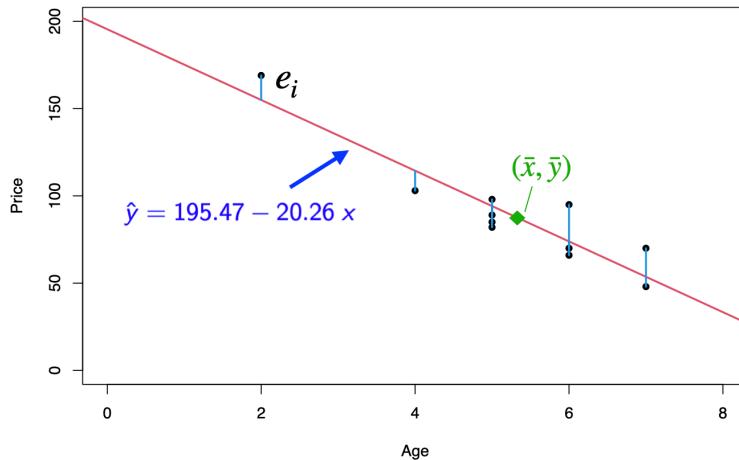
The table below displays data on Age (in years) and Price (in hundreds of dollars) for a sample of cars of a particular make and model. Weiss [2012]

Price ( $y$ )	Age ( $x$ )
85	5
103	4
70	6
82	5
89	5
98	5
66	6
95	6
169	2
70	7
48	7

- For our example, *age* is the predictor variable and *price* is the response variable.

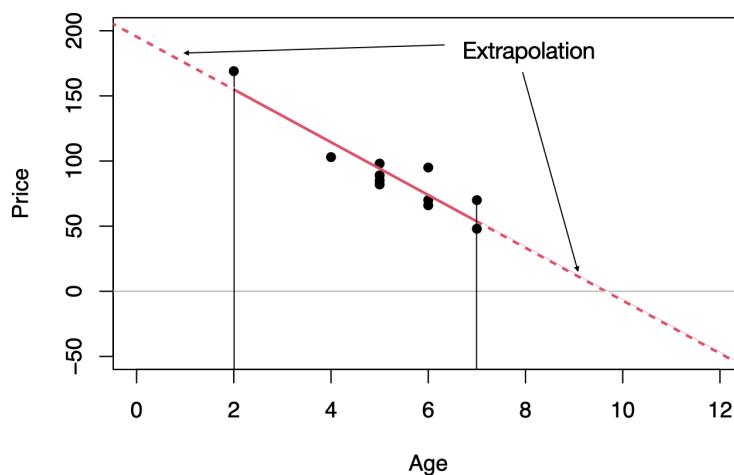
- The regression equation is  $\hat{y} = 195.47 - 20.26 x$ , where the slope  $b_1 = -20.26$  and the intercept  $b_0 = 195.47$
- Prediction: for  $x = 4$ , that is we would like to predict the price of a 4-year-old car,

$$\hat{y} = 195.47 - 20.26(4) = 114.43 \text{ or } \$11443$$



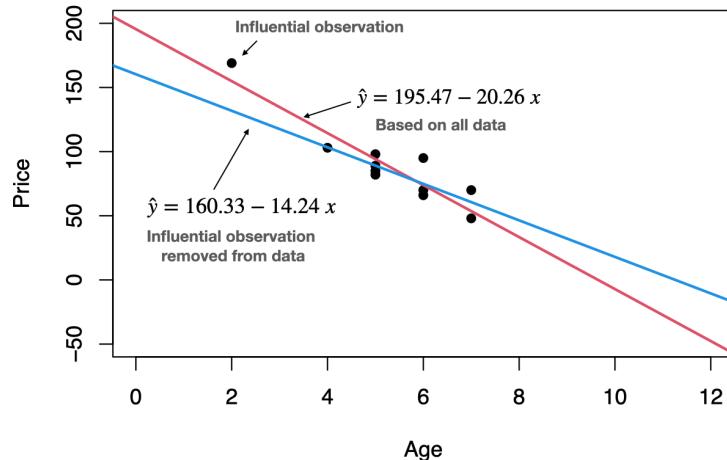
## Extrapolation

- Within the range of the observed values of the predictor variable, we can reasonably use the regression equation to make predictions for the response variable.
- However, to do so outside the range, which is called *Extrapolation*, may not be reasonable because the linear relationship between the predictor and response variables may not hold here.
- To predict the price of an 11-year old car,  $\hat{y} = 195.47 - 20.26(11) = -27.39$  or  $\$ 2739$ , this result is unrealistic as no one is going to pay us  $\$2739$  to take away their 11-year old car.

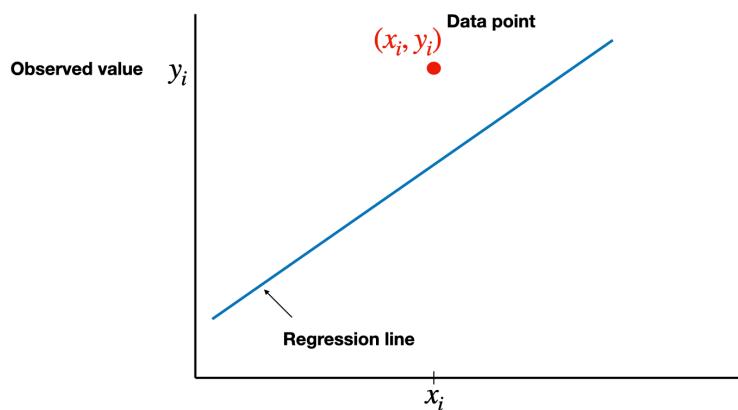


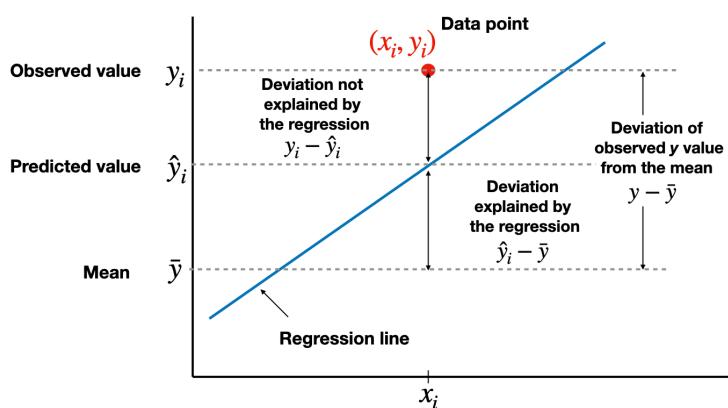
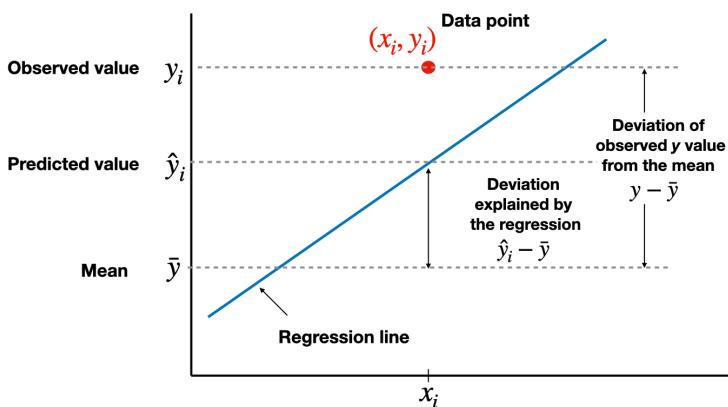
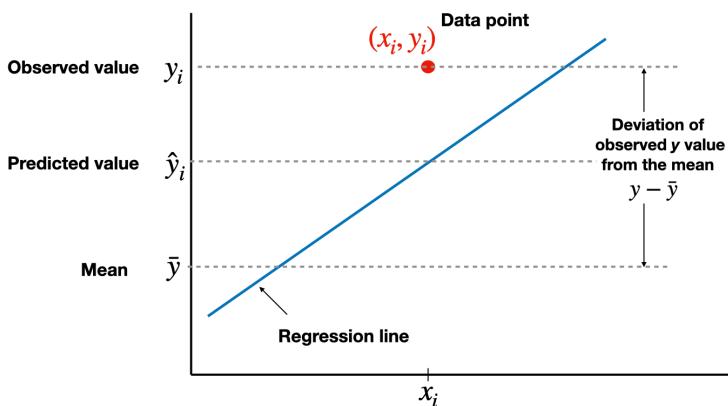
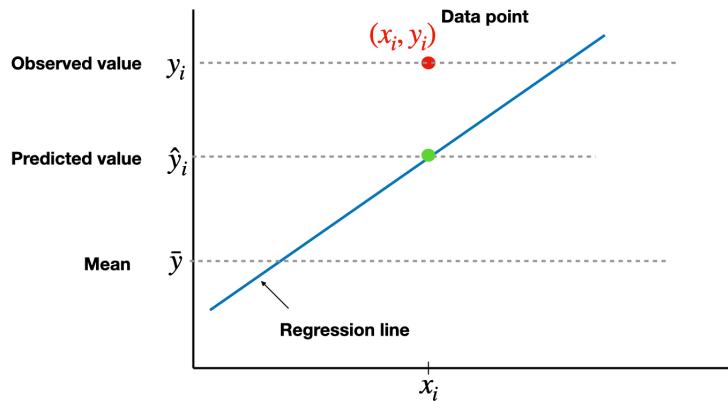
## Outliers and influential observations

- An *outlier* is an observation that lies outside the overall pattern of the data. In the context of regression, an outlier is a data point that lies far from the regression line, relative to the other data points.
- An *influential observation* is a data point whose removal causes the regression equation (and line) to change considerably.
- From the scatterplot, it seems that the data point (2,169) might be an influential observation. Removing that data point and recalculating the regression equation yields  $\hat{y} = 160.33 - 14.24 x$ .



## Coefficient of determination





- The total variation in the observed values of the response variable,  $SST = \sum(y_i - \bar{y})^2$ ,

can be partitioned into two components:

- The variation in the observed values of the response variable explained by the regression:  $SSR = \sum(\hat{y}_i - \bar{y})^2$
- The variation in the observed values of the response variable not explained by the regression:  $SSE = \sum(y_i - \hat{y}_i)^2$
- The coefficient of determination,  $R^2$  (or  $R$ -square), is the proportion of the variation in the observed values of the response variable explained by the regression, which is given by

$$R^2 = \frac{SSR}{SST} = \frac{SST - SSE}{SST} = 1 - \frac{SSE}{SST}$$

where  $SST = SSR + SSE$ .  $R^2$  is a descriptive measure of the utility of the regression equation for making prediction.

- The coefficient of determination  $R^2$  always lies between 0 and 1. A value of  $R^2$  near 0 suggests that the regression equation is not very useful for making predictions, whereas a value of  $R^2$  near 1 suggests that the regression equation is quite “useful” for making predictions.
- For a simple linear regression (one independent variable) ONLY,  $R^2$  is the square of Pearson correlation coefficient,  $r$ .
- Adjusted  $R^2$  is a modification of  $R^2$  which takes into account the number of independent variables, say  $k$ . In a simple linear regression  $k = 1$ . Adjusted- $R^2$  increases only when a significant related independent variable is added to the model. Adjusted- $R^2$  has a crucial role in the process of model building. Adjusted- $R^2$  is given by

$$\text{Adjusted-}R^2 = 1 - (1 - R^2) \frac{n - 1}{n - k - 1}$$

## Notation used in regression

Quantity	Defining formula	Computing formula
$S_{xx}$	$\sum(x_i - \bar{x})^2$	$\sum x_i^2 - n\bar{x}^2$
$S_{xy}$	$\sum(x_i - \bar{x})(y_i - \bar{y})$	$\sum x_i y_i - n\bar{x}\bar{y}$
$S_{yy}$	$\sum(y_i - \bar{y})^2$	$\sum y_i^2 - n\bar{y}^2$

where  $\bar{x} = \frac{\sum x_i}{n}$  and  $\bar{y} = \frac{\sum y_i}{n}$ . And,

$$SST = S_{yy}, \quad SSR = \frac{S_{xy}^2}{S_{xx}}, \quad SSE = S_{yy} - \frac{S_{xy}^2}{S_{xx}}$$

and  $SST = SSR + SSE$ .

## Proof $R^2$ is the square of Pearson correlation coefficient

Suppose that we have  $n$  observations  $(x_1, y_1), \dots, (x_n, y_n)$  from a simple linear regression

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where  $i = 1, \dots, n$ . Let us denote  $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$  for  $i = 1, \dots, n$ , where  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are the least squares estimators of the parameters  $\beta_0$  and  $\beta_1$ . The coefficient of the determination  $R^2$  is defined by

$$R^2 = \frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Using the facts that

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

and  $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$ , we obtain

$$\begin{aligned} \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 &= \sum_{i=1}^n (\hat{\beta}_0 + \hat{\beta}_1 x_i - \bar{y})^2 \\ &= \sum_{i=1}^n (\bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 x_i - \bar{y})^2 \\ &= \hat{\beta}_1^2 \sum_{i=1}^n (x_i - \bar{x})^2 \\ &= \frac{[\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})]^2 \sum_{i=1}^n (x_i - \bar{x})^2}{[\sum_{i=1}^n (x_i - \bar{x})^2]^2} \\ &= \frac{[\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})]^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \end{aligned} \tag{3.1}$$

Hence,

$$\begin{aligned} R^2 &= \frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \\ &= \frac{[\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})]^2}{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2} \\ &= \left( \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}} \right)^2 \\ &= r^2 \end{aligned} \tag{3.2}$$

This shows that the coefficient of determination of a simple linear regression is the square of the sample correlation coefficient of  $(x_1, y_1), \dots, (x_n, y_n)$ .

### 3.3 Simple Linear Regression: Assumptions

Recall that the simple linear regression model for  $Y$  on  $x$  is

$$Y = \beta_0 + \beta_1 x + \epsilon$$

where

$Y$  : the dependent or response variable

$x$  : the independent or predictor variable, assumed known

$\beta_0, \beta_1$  : the regression parameters, the intercept and slope of the regression line

$\epsilon$  : the random regression error around the line.

and the regression equation for a set of  $n$  data points is  $\hat{y} = b_0 + b_1 x$ , where

$$b_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

and

$$b_0 = \bar{y} - b_1 \bar{x}$$

where  $b_0$  is called the **y-intercept** and  $b_1$  is called the **slope**.

The *residual standard error*  $s_e$  can be defined as

$$s_e = \sqrt{\frac{SSE}{n-2}} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}}$$

$s_e$  indicates how much, on average, the observed values of the response variable differ from the predicted values of the response variable.

#### Simple Linear Regression Assumptions (SLR)

We have a collection of  $n$  pairs of observations  $\{(x_i, y_i)\}$ , and the idea is to use them to estimate the unknown parameters  $\beta_0$  and  $\beta_1$ .

$$\epsilon_i = Y_i - (\beta_0 + \beta_1 x_i), \quad i = 1, 2, \dots, n$$

We need to make the following key assumptions on the errors:

- A.  $E(\epsilon_i) = 0$  (errors have mean zero and do not depend on  $x$ )
- B.  $Var(\epsilon_i) = \sigma^2$  (errors have a constant variance, homoscedastic, and do not depend on  $x$ )
- C.  $\epsilon_1, \epsilon_2, \dots, \epsilon_n$  are independent.
- D.  $\epsilon_i$  are all i.i.d.  $N(0, \sigma^2)$ , meaning that the errors are independent and identically distributed as Normal with mean zero and constant variance  $\sigma^2$ .

The above assumptions, and conditioning on  $\beta_0$  and  $\beta_1$ , imply:

- a. Linearity:  $E(Y_i | X_i) = \beta_0 + \beta_1 x_i$

- b. Homogeneity or homoscedasticity:  $\text{Var}(Y_i|X_i) = \sigma^2$
- c. Independence:  $Y_1, Y_2, \dots, Y_n$  are all independent given  $X_i$ .
- d. Normality:  $Y_i|X_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$

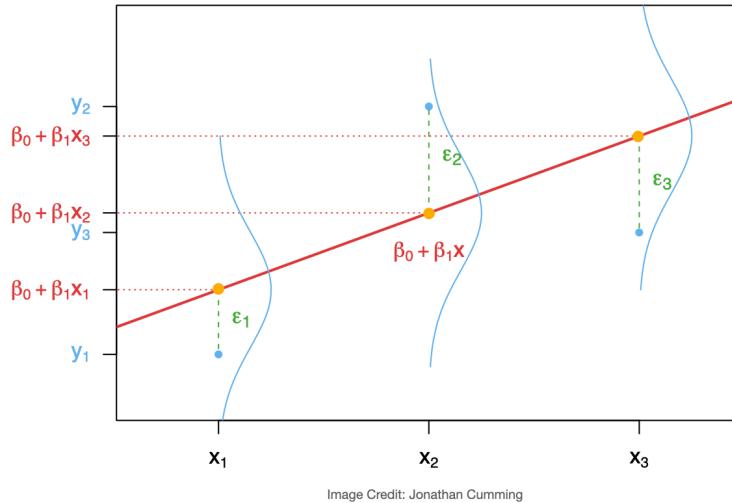


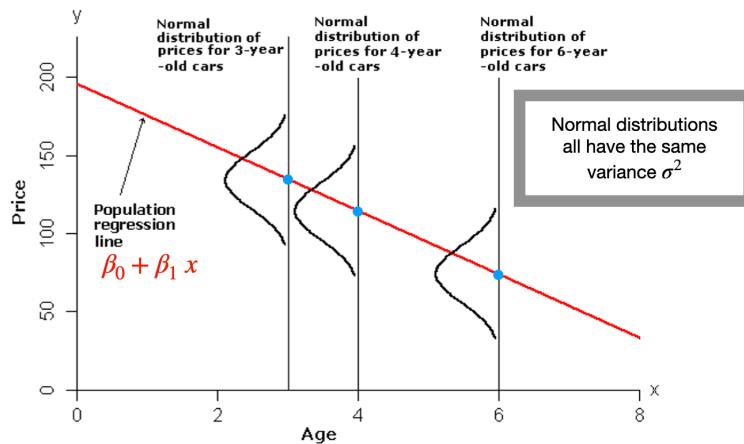
Image Credit: Jonathan Cumming

It is in fact under these assumptions, minimizing the least square criterion is equivalent to maximizing the log-likelihood of the model parameters.

## Used cars example

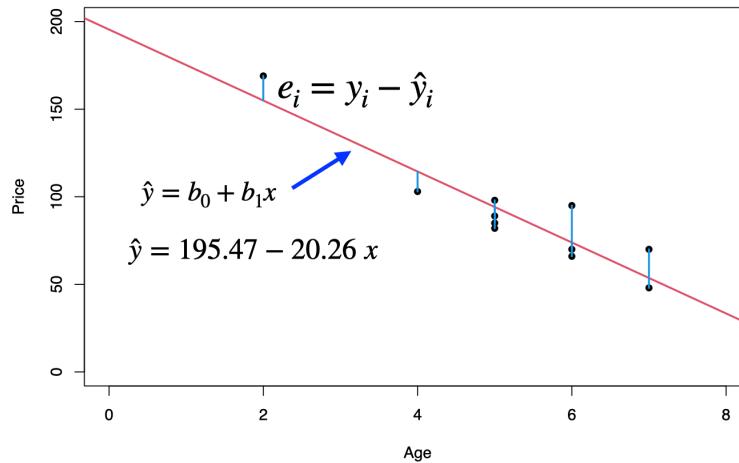
The table below displays data on Age (in years) and Price (in hundreds of dollars) for a sample of cars of a particular make and model. Weiss [2012]

Price ( $y$ )	Age ( $x$ )
85	5
103	4
70	6
82	5
89	5
98	5
66	6
95	6
169	2
70	7
48	7



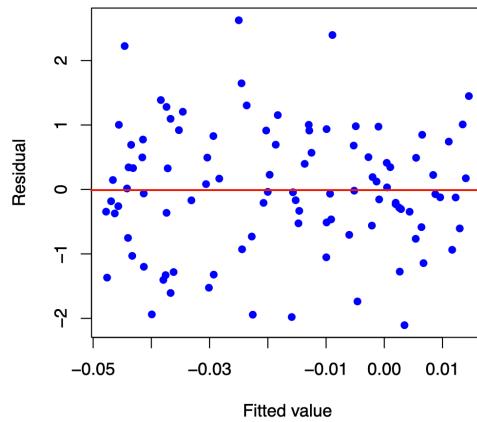
We can see that for each age, the mean price of all cars of that age lies on the regression line  $E(Y|x) = \beta_0 + \beta_1 x$ . And, the prices of all cars of that age are assumed to be normally distributed with mean equal to  $\beta_0 + \beta_1 x$  and variance  $\sigma^2$ . For example, the prices of all 4-year-old cars must be normally distributed with mean  $\beta_0 + \beta_1(4)$  and variance  $\sigma^2$ .

We used the least square method to find the best fit for this data set, and residuals can be obtained as  $e_i = y_i - \hat{y}_i = y_i - (195.47 - 20.26x_i)$ .



## Residual Analysis

The easiest way to check the simple linear regression assumptions is by constructing a scatter-plot of the residuals ( $e_i = y_i - \hat{y}_i$ ) against the fitted values ( $\hat{y}_i$ ) or against  $x$ . If the model is a good fit, then the **residual plot** should show an even and random scatter of the residuals.



## Linearity

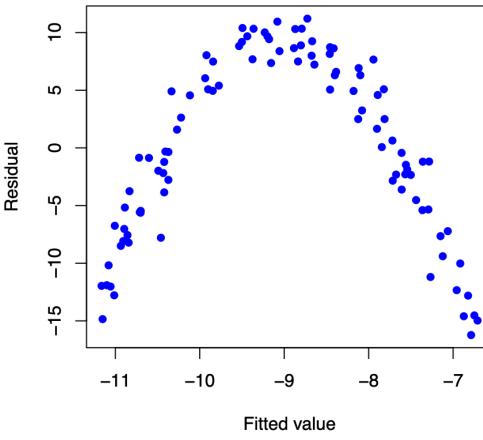
The regression needs to be linear in the parameters.

$$Y = \beta_0 + \beta_1 x + \epsilon$$

$$E(Y_i|X_i) = \beta_0 + \beta_1 x_i \equiv E(\epsilon_i|X_i) = E(\epsilon_i) = 0$$

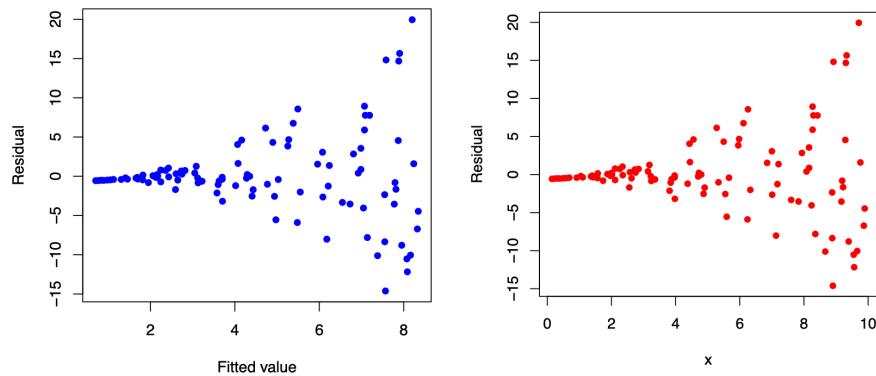
- ✗  $\beta_0 + \beta_1^2 x_i$
- ✓  $\beta_0 + \beta_1 \log(x_i)$
- ✓  $\beta_0 + \beta_1 x_i^2$

The residual plot below shows that a linear regression model is not appropriate for this data.



## Constant error variance (homoscedasticity)

The plot shows the spread of the residuals is increasing as the fitted values (or  $x$ ) increase, which indicates that we have Heteroskedasticity (non-constant variance). The standard errors are biased when heteroskedasticity is present, but the regression coefficients will still be unbiased.



### How to detect?

- Residuals plot (fitted vs residuals)
- Goldfeld–Quandt test
- Breusch-Pagan test

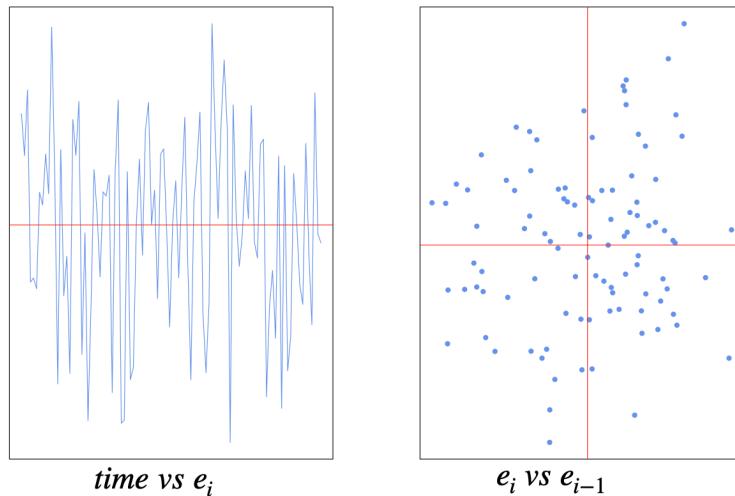
### How to fix?

- White's standard errors
- Weighted least squares model
- Taking the log

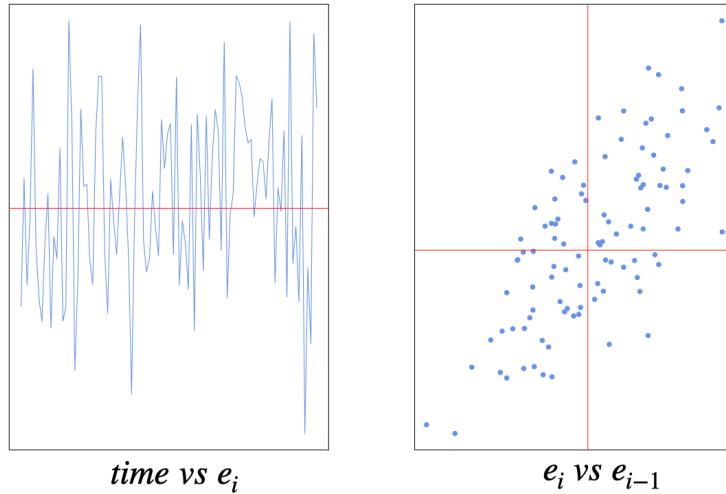
### Independent errors terms (no autocorrelation)

The problem of autocorrelation is most likely to occur in time series data, however, it can also occur in cross-sectional data, e.g. if the model is incorrectly specified. When autocorrelation is present, the regression coefficients will still be unbiased, however, the standard errors and test statistics are no longer valid.

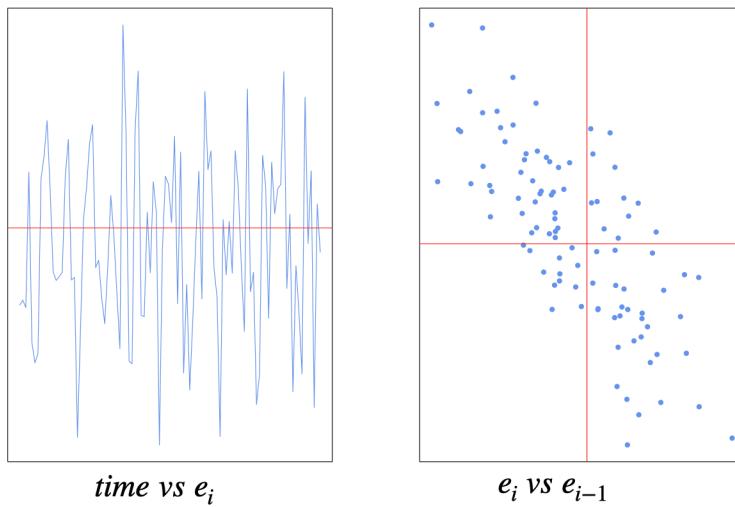
### An example of no autocorrelation



### An example of positive autocorrelation



### An example of negative autocorrelation



### How to detect?

- Residuals plot
- Durbin-Watson test
- Breusch-Godfrey test

### How to fix?

- Investigate omitted variables (e.g. trend, business cycle)
- Use advanced models (e.g. AR model)

### Normality of the errors

We need the errors to be normally distributed. Normality is required for the sampling distributions, hypothesis testing and confidence intervals.

### How to detect?

- Histogram of residuals
- Q-Q plot of residuals
- Kolmogorov–Smirnov test
- Shapiro–Wilk test

### How to fix?

- Change the functional form (e.g. taking the log)
- Larger sample if possible

## Box-Cox Transformation

Normality is an important assumption for many statistical techniques. The Box-Cox transformation transforms our data so that it closely resembles a normal distribution.

Box-Cox transformation applies to a **positive** response  $y$  with the form

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

such that  $y^{(\lambda)}$  satisfies the assumptions:

- $E(Y_i^{(\lambda)}|X_i) = \beta_0 + \beta_1 x_i$
- $Var(Y_i^{(\lambda)}|X_i) = \sigma^2$
- $Y_1^{(\lambda)}, Y_2^{(\lambda)}, \dots, Y_n^{(\lambda)}$  are all independent given  $X_i$ .
- $Y_i^{(\lambda)}|X_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$

As usually  $\lambda$  varies over  $(-2, 2)$ ,  $y^{(\lambda)}$  encompasses the reciprocal transformation ( $\lambda = -1$ ),  $\log$  ( $\lambda = 0$ ), square root ( $\lambda = \frac{1}{2}$ ), the original scale ( $\lambda = 1$ ) and the square transformation ( $\lambda = 2$ ). If the  $y_i$  are not positive we apply the transformation to  $y_i + \gamma$ , where  $\gamma$  is chosen to make all the  $y_i + \gamma$  positive.

The Box-Cox method chooses  $\lambda$  according to the likelihood function. Hint: taking account of the Jacobian of the transformation from  $y^{(\lambda)}$  to  $y$ , namely  $y^{\lambda-1}$ , the density of  $y_i$  is

$$f(y_i|x_i) = \frac{y_i^{\lambda-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp \left[ -\frac{1}{2\sigma^2} (y_i^{(\lambda)} - x_i\beta_1 - \beta_0)^2 \right]$$

In general, one can simply consider  $\lambda$  as a hyperparameter. In machine learning, *hyperparameters* are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. The prefix ‘hyper\_’ suggests that they are ‘top-level’ parameters that control the learning process and the model parameters that result from it. As a machine learning engineer designing a model, you choose and set hyperparameter values that your learning algorithm will use before the training of the

model even begins. For example, in the simple linear regression, we choose the value of  $\lambda$  before estimating the model parameter  $\beta_0$  and  $\beta_1$ .

## Example: Infant mortality and GDP

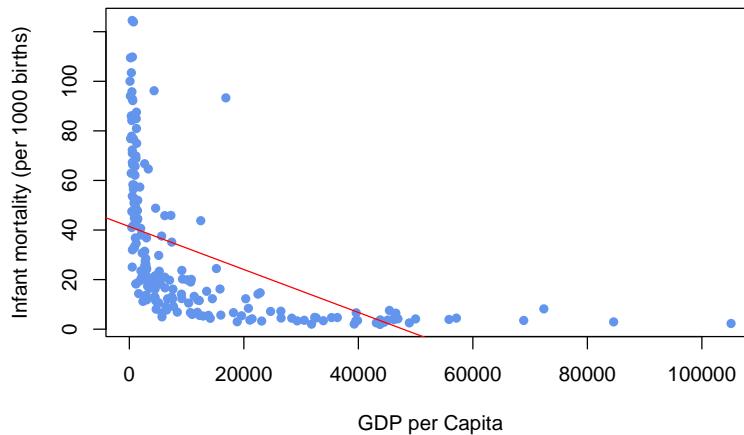
Let us investigate the relationship between infant mortality and the wealth of a country. We will use data on 207 countries of the world gathered by the UN in 1998 (the ‘UN’ data set is available from the R package ‘carData’). The data set contains two variables: the infant mortality rate in deaths per 1000 live births, and the GDP per capita in US dollars. There are some missing data values for some countries, so we will remove the missing data before we fit our model.

```
# install.packages("carData")
library(carData)
data(UN)
options(scipen=999)
# Remove missing data
newUN<-na.omit(UN)
str(newUN)

## 'data.frame': 193 obs. of 7 variables:
## $ region : Factor w/ 8 levels "Africa","Asia",...: 2 4 1 1 5 2 3 8 4 2 ...
## $ group  : Factor w/ 3 levels "oecd","other",...: 2 2 3 3 2 2 2 1 1 2 ...
## $ fertility : num 5.97 1.52 2.14 5.13 2.17 ...
## $ ppgdp   : num 499 3677 4473 4322 9162 ...
## $ lifeExpF : num 49.5 80.4 75 53.2 79.9 ...
## $ pctUrban : num 23 53 67 59 93 64 47 89 68 52 ...
## $ infantMortality: num 124.5 16.6 21.5 96.2 12.3 ...
## - attr(*, "na.action")= 'omit' Named int [1:20] 4 6 21 35 38 54 67 75 77 78 ...
## ..- attr(*, "names")= chr [1:20] "American Samoa" "Anguilla" "Bermuda" "Cayman Islands" ...
fit<- lm(infantMortality ~ ppgdp, data=newUN)
summary(fit)

##
## Call:
## lm(formula = infantMortality ~ ppgdp, data = newUN)
##
## Residuals:
##    Min      1Q  Median      3Q     Max 
## -31.48 -18.65 - 8.59  10.86  83.59 
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)    
## (Intercept) 41.3780016  2.2157454 18.675 < 0.0000000000000002 *** 
## ppgdp       -0.0008656  0.0001041 -8.312  0.0000000000000173 *** 
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

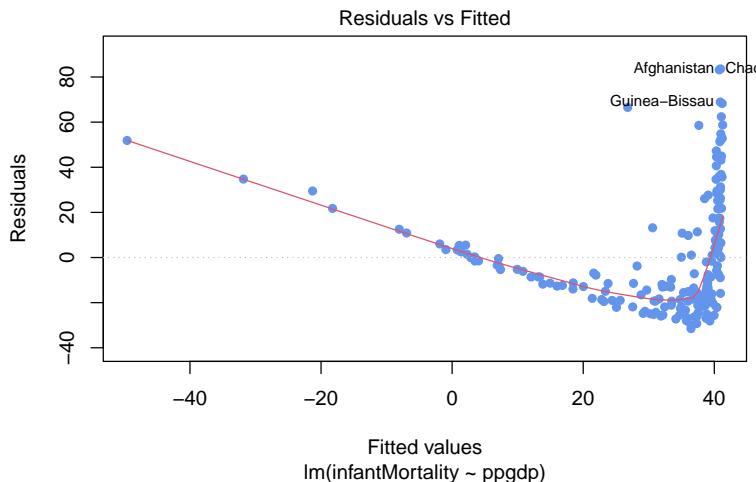
```
##  
## Residual standard error: 25.13 on 191 degrees of freedom  
## Multiple R-squared:  0.2656, Adjusted R-squared:  0.2618  
## F-statistic: 69.08 on 1 and 191 DF,  p-value: 0.000000000000000173  
plot(newUN$infantMortality ~ newUN$ppgdp, xlab="GDP per Capita", ylab="Infant mortality"  
abline(fit,col="red")
```



We can see from the scatterplot that the relationship between the two variables is not linear. There is a concentration of data points at small values of GDP (many poor countries) and a concentration of data points at small values of infant mortality (many countries with very low mortality). This suggests a skewness to both variables which would not conform to the normality assumption. Indeed, the regression line (the red line) we construct is a poor fit and only has an  $R^2$  of 0.266.

From the residual plot below we can see a clear evidence of structure to the residuals suggesting the linear relationship is a poor description of the data, and substantial changes in spread suggesting the assumption of homogeneous variance is not appropriate.

```
# diagnostic plots  
plot(fit,which=1,pch=16,col="cornflowerblue")
```

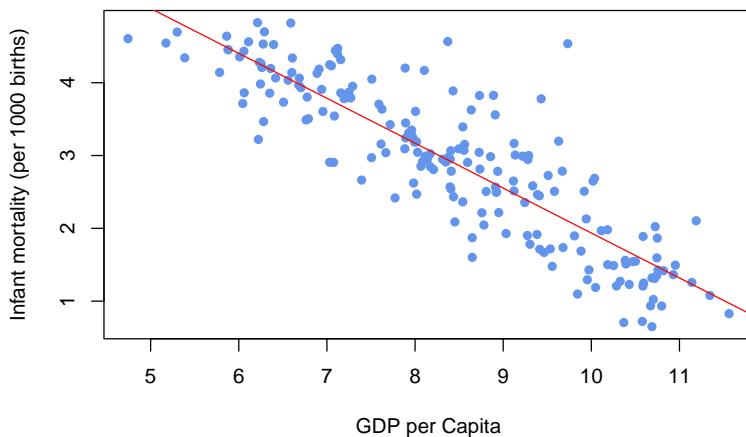


So we can apply a transformation to one or both variables, e.g. taking the log or adding a quadratic form. Notice that this will not affect (violet) the linearity assumption as the

regression will still be linear in the parameters. So if we take the logs of both variables gives us the scatterplot of the transformed data set, below, which appears to show a more promising linear structure. The quality of the regression is now improved, with an  $R^2$  value of 0.766, which is still a little weak due to the rather large spread in the data.

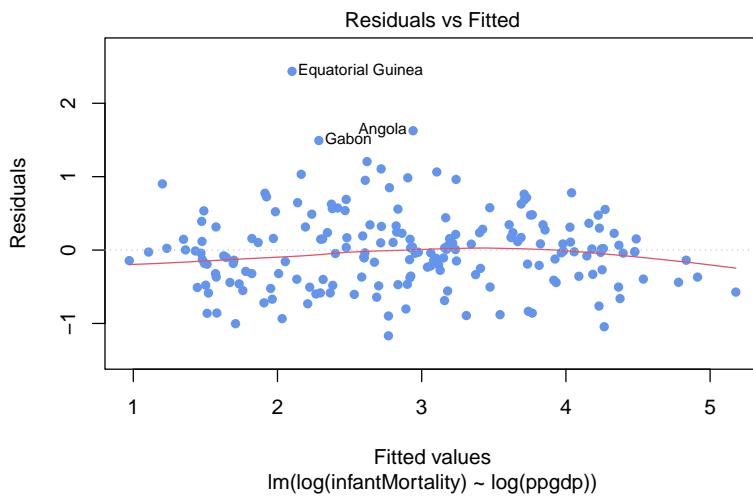
```
fit1<- lm(log(infantMortality) ~ log(ppgdp), data=newUN)
summary(fit1)
```

```
##
## Call:
## lm(formula = log(infantMortality) ~ log(ppgdp), data = newUN)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.16789 -0.36738 -0.02351  0.24544  2.43503
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept) 8.10377   0.21087  38.43 <0.0000000000000002 ***
## log(ppgdp) -0.61680   0.02465 -25.02 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5281 on 191 degrees of freedom
## Multiple R-squared:  0.7662, Adjusted R-squared:  0.765
## F-statistic: 625.9 on 1 and 191 DF,  p-value: < 0.0000000000000022
plot(log(newUN$infantMortality) ~ log(newUN$ppgdp), xlab="GDP per Capita", ylab="Infant
abline(fit1,col="red")
```



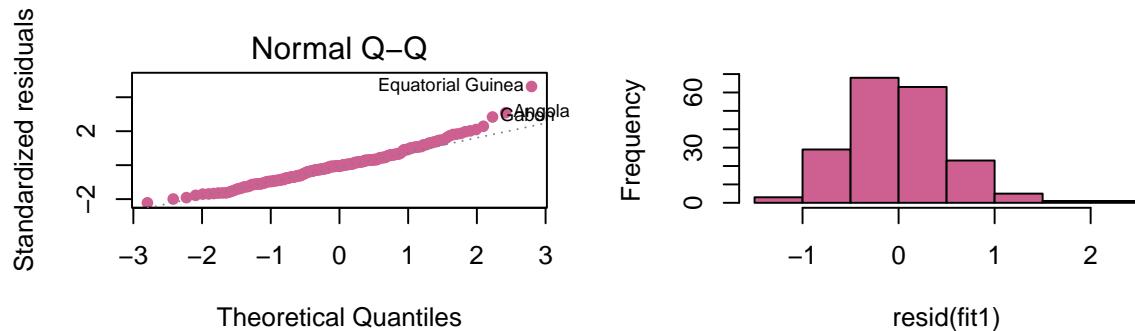
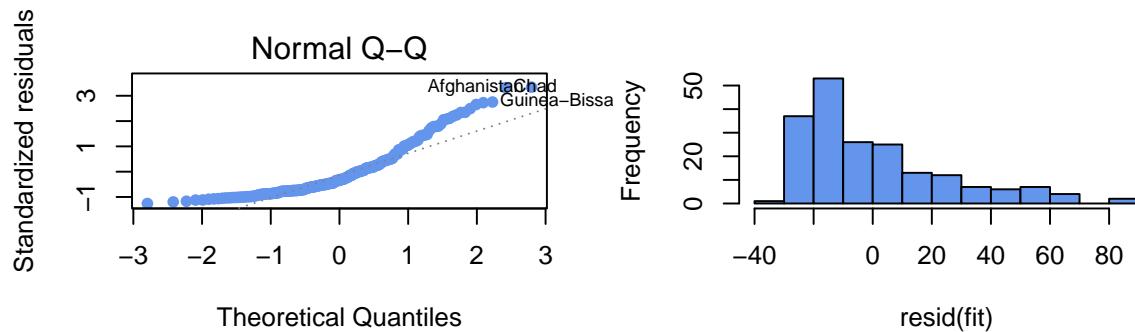
So we check the residuals again, as we can see from the residuals plot below that the log transformation has corrected many of the problems with with residual plot and the residuals now much closer to the expected random scatter.

```
# diagnostic plots
plot(fit1,which=1,pch=16,col="cornflowerblue")
```



Now let us check the Normality of the errors by creating a histogram and normal QQ plot for the residuals, before and after the log transformation. The normal quantile (QQ) plot shows the sample quantiles of the residuals against the theoretical quantiles that we would expect if these values were drawn from a Normal distribution. If the Normal assumption holds, then we would see an approximate straight-line relationship on the Normal quantile plot.

```
par(mfrow=c(2,2))
# before the log transformation.
plot(fit, which = 2, pch=16, col="cornflowerblue")
hist(resid(fit),col="cornflowerblue",main="")
# after the log transformation.
plot(fit1, which = 2, pch=16, col="hotpink3")
hist(resid(fit1),col="hotpink3",main="")
```



The normal quantile plot and the histogram of residuals (before the log transformation) show strong departure from the expectation of an approximately straight line, with curvature in the tails which reflects the skewness of the data. Finally, the normal quantile plot and the histogram of residuals suggest that residuals are much closer to Normality after the transformation, with some minor deviations in the tails.

## 3.4 Simple Linear Regression: Inference

### Simple Linear Regression Assumptions

- Linearity of the relationship between the dependent and independent variables
- Independence of the errors (no autocorrelation)
- Constant variance of the errors (homoscedasticity)
- Normality of the error distribution.

### Simple Linear Regression Model

The simple linear regression model for  $Y$  on  $x$  is

$$Y = \beta_0 + \beta_1 x + \epsilon$$

where

$Y$  : the dependent or response variable

$x$  : the independent or predictor variable, assumed known

$\beta_0, \beta_1$  : the regression parameters, the intercept and the slope of the regression line

$\epsilon$  : the random regression error around the line.

### The simple linear regression equation

- The regression equation for a set of  $n$  data points is  $\hat{y} = b_0 + b_1 x$ , where

$$b_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

and

$$b_0 = \bar{y} - b_1 \bar{x}$$

- $y$  is the dependent variable (or response variable) and  $x$  is the independent variable (predictor variable or explanatory variable).
- $b_0$  is called the **y-intercept** and  $b_1$  is called the **slope**.

## Residual standard error, $s_e$

The residual standard error,  $s_e$ , is defined by

$$s_e = \sqrt{\frac{SSE}{n-2}}$$

where  $SSE$  is the error sum of squares (also known as the residual sum of squares, RSS) which can be defined as

$$SSE = \sum e_i^2 = \sum (y_i - \hat{y}_i)^2 = S_{yy} - \frac{S_{xy}^2}{S_{xx}}$$

$s_e$  indicates how much, on average, the observed values of the response variable differ from the predicted values of the response variable. Under the simple linear regression assumptions,  $s_e$  is an unbiased estimate for the error standard deviation  $\sigma$ .

## Properties of Regression Coefficients

Under the simple linear regression assumptions, the least square estimates  $b_0$  and  $b_1$  are unbiased for the  $\beta_0$  and  $\beta_1$ , respectively, i.e.

$$E[b_0] = \beta_0 \text{ and } E[b_1] = \beta_1.$$

The variances of the least squares estimators in simple linear regression are:

$$Var[b_0] = \sigma_{b_0}^2 = \sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right)$$

$$Var[b_1] = \sigma_{b_1}^2 = \frac{\sigma^2}{S_{xx}}$$

$$Cov[b_0, b_1] = \sigma_{b_0, b_1} = -\sigma^2 \frac{\bar{x}}{S_{xx}}$$

We use  $s_e$  to estimate the error standard deviation  $\sigma$ :

$$s_{b_0}^2 = s_e^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right)$$

$$s_{b_1}^2 = \frac{s_e^2}{S_{xx}}$$

$$s_{b_0, b_1} = -s_e^2 \frac{\bar{x}}{S_{xx}}$$

The proof of some of the properties are provided in the following, the rest can be find in exercise sheets.

- **Proof of**  $E[b_1] = \beta_1$

$$\begin{aligned} E(b_1) &= E\left(\frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}\right) \\ &= E\left(\frac{\sum(x_i - \bar{x})y_i - \bar{y}\sum(x_i - \bar{x}))}{\sum(x_i - \bar{x})^2}\right) \end{aligned} \tag{3.3}$$

Note  $\sum(x_i - \bar{x}) = 0$ , hence

$$\begin{aligned} E(b_1) &= E\left(\frac{\sum(x_i - \bar{x})y_i}{\sum(x_i - \bar{x})^2}\right) \\ &= \frac{\sum(x_i - \bar{x})(\beta_0 + \beta_1 x_i)}{\sum(x_i - \bar{x})^2} \\ &= \frac{\sum(x_i - \bar{x})\beta_1 x_i}{\sum(x_i - \bar{x})^2} \end{aligned} \tag{3.4}$$

Note  $\sum(x_i - \bar{x})^2 = \sum(x_i - \bar{x})x_i$ , hence

$$E(b_1) = \beta_1 \tag{3.5}$$

- **Proof of**  $Var[b_1] = \frac{\sigma^2}{S_{xx}}$

From the proof of  $E[b_1] = \beta_1$ , we have

$$\begin{aligned} Var(b_1) &= Var\left(\frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}\right) \\ &= Var\left(\frac{\sum(x_i - \bar{x})y_i}{\sum(x_i - \bar{x})^2}\right) \\ &= \frac{Var(\sum(x_i - \bar{x})y_i)}{(\sum(x_i - \bar{x})^2)^2} \\ &= \frac{\sum_{i=1}^n Var[(x_i - \bar{x})y_i] + \sum_{i \neq j} 2(x_i - \bar{x})(x_j - \bar{x})Cov(y_i, y_j)}{(\sum(x_i - \bar{x})^2)^2} \end{aligned} \tag{3.6}$$

Since  $y_i$ s are independent, hence

$$\begin{aligned} Var(b_1) &= \frac{\sum_{i=1}^n Var[(x_i - \bar{x})y_i] + \sum_{i \neq j} 2(x_i - \bar{x})(x_j - \bar{x})Cov(y_i, y_j)}{(\sum(x_i - \bar{x})^2)^2} \\ &= \frac{\sum(x_i - \bar{x})^2 \sigma^2}{(\sum(x_i - \bar{x})^2)^2} \\ &= \frac{\sigma^2}{S_{xx}} \end{aligned} \tag{3.7}$$

- **Proof of**  $Var[b_0] = \sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right)$

$$\begin{aligned}
Var(b_0) &= Var(\bar{y} - b_1 \bar{x}) \\
&= Var(\bar{y}) + Var(b_1 \bar{x}) - 2Cov(\bar{y}, b_1 \bar{x}) \\
&= Var\left(\frac{\sum y_i}{n}\right) + \bar{x}^2 Var(b_1) - 2Cov(\bar{y}, b_1 \bar{x}) \\
&= \frac{\sigma^2}{n} + \bar{x}^2 \frac{\sigma^2}{S_{xx}} - 2Cov(\bar{y}, b_1 \bar{x})
\end{aligned} \tag{3.8}$$

where

$$\begin{aligned}
Cov(\bar{y}, b_1 \bar{x}) &= Cov\left(\frac{\sum y_i}{n}, \frac{\sum(x_i - \bar{x})y_i}{\sum(x_i - \bar{x})^2} \bar{x}\right) \\
&= \frac{\bar{x}}{n \sum(x_i - \bar{x})^2} Cov[\sum y_i, \sum(x_i - \bar{x})y_i]
\end{aligned} \tag{3.9}$$

$$\begin{aligned}
Cov[\sum y_i, \sum(x_i - \bar{x})y_i] &= E[(\sum y_i) \sum(x_i - \bar{x})y_i] - E[\sum y_i]E[\sum(x_i - \bar{x})y_i] \\
&= E[\sum(x_i - \bar{x})y_i^2 + \sum_{i \neq j}(x_i - \bar{x})y_i y_j] - nE[y] \sum(x_i - \bar{x})E(y_i) \\
&= \sum(x_i - \bar{x})(Var(y_i) + E(y_i)^2) + \sum_{i \neq j}(x_i - \bar{x})E(y_i)E(y_j) \\
&\quad - nE[y]^2 \sum(x_i - \bar{x}) \\
&= (Var(y) + E(y)^2) \sum(x_i - \bar{x}) + E(y)^2 \sum(x_i - \bar{x}) \\
&\quad - nE[y]^2 \sum(x_i - \bar{x}) \\
&= 0
\end{aligned} \tag{3.10}$$

Hence

$$Var(b_0) = \sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right) \tag{3.11}$$

### Sampling distribution of the least square estimators

For the Normal error simple linear regression model:

$$b_0 \sim N(\beta_0, \sigma_{b_0}^2) \rightarrow \frac{b_0 - \beta_0}{\sigma_{b_0}} \sim N(0, 1)$$

and

$$b_1 \sim N(\beta_1, \sigma_{b_1}^2) \rightarrow \frac{b_1 - \beta_1}{\sigma_{b_1}} \sim N(0, 1)$$

We use  $s_e$  to estimate the error standard deviation  $\sigma$ :

$$\frac{b_0 - \beta_0}{s_{b_0}} \sim t_{n-2}$$

and

$$\frac{b_1 - \beta_1}{s_{b_1}} \sim t_{n-2}$$

## Degrees of Freedom

- In statistics, degrees of freedom are the number of independent pieces of information that go into the estimate of a particular parameter.
- Typically, the degrees of freedom of an estimate of a parameter is equal to the number of independent observations that go into the estimate, minus the number of parameters that are estimated as intermediate steps in the estimation of the parameter itself.
- The sample variance has  $n - 1$  degrees of freedom since it is computed from  $n$  pieces of data, minus the 1 parameter estimated as the intermediate step, the sample mean. Similarly, having estimated the sample mean we only have  $n - 1$  independent pieces of data left, as if we are given the sample mean and any  $n - 1$  of the observations then we can determine the value of the remaining observation exactly.

$$s^2 = \frac{\sum(x_i - \bar{x})^2}{n - 1}$$

- In linear regression, the degrees of freedom of the residuals is  $df = n - k^*$ , where  $k^*$  is the number of estimated parameters (including the intercept). So for the simple linear regression, we are estimating  $\beta_0$  and  $\beta_1$ , thus  $df = n - 2$ .

## Inference for the intercept $\beta_0$

- Hypotheses:

$$H_0 : \beta_0 = 0 \text{ against } H_1 : \beta_0 \neq 0$$

- Test statistic:

$$t = \frac{b_0}{s_{b_0}}$$

has a t-distribution with  $df = n - 2$ , where  $s_{b_0}$  is the standard error of  $b_0$ , and given by

$$s_{b_0} = s_e \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}$$

and

$$s_e = \sqrt{\frac{SSE}{n-2}} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}}$$

We reject  $H_0$  at level  $\alpha$  if  $|t| > t_{\alpha/2}$  with  $df = n - 2$ .

- 100(1- $\alpha$ )% confidence interval for  $\beta_0$ ,

$$b_0 \pm t_{\alpha/2} \cdot s_{b_0}$$

where  $t_{\alpha/2}$  is critical value obtained from the t-distribution table with  $df = n - 2$ .

## Inference for the slope $\beta_1$

- Hypotheses:

$$H_0 : \beta_1 = 0 \text{ against } H_1 : \beta_1 \neq 0$$

- Test statistic:

$$t = \frac{b_1}{s_{b_1}}$$

has a t-distribution with  $df = n - 2$ , where  $s_{b_1}$  is the standard error of  $b_1$ , and given by

$$s_{b_1} = \frac{s_e}{\sqrt{S_{xx}}}$$

and

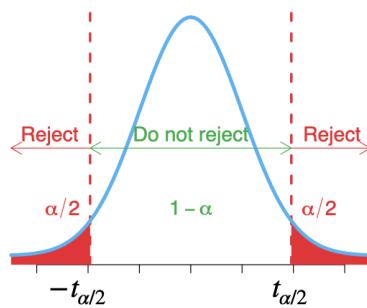
$$s_e = \sqrt{\frac{SSE}{n-2}} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}}$$

We reject  $H_0$  at level  $\alpha$  if  $|t| > t_{\alpha/2}$  with  $df = n - 2$ .

- 100(1- $\alpha$ )% confidence interval for  $\beta_1$ ,

$$b_1 \pm t_{\alpha/2} s_{b_1}$$

where  $t_{\alpha/2}$  is critical value obtained from the t-distribution table with  $df = n - 2$ .



## Is the regression model “useful”?

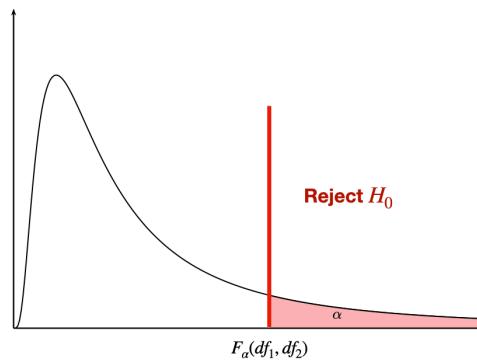
### Goodness of fit test

- We test the null hypothesis  $H_0 : \beta_1 = 0$  against  $H_1 : \beta_1 \neq 0$ , the F-statistic

$$F = \frac{MSR}{MSE} = \frac{SSR}{SSE/(n-2)}$$

has F-distribution with degrees of freedom  $df_1 = 1$  and  $df_2 = n - 2$ .

- We reject  $H_0$ , at level  $\alpha$ , if  $F > F_\alpha(df_1, df_2)$ .
- For a simple linear regression ONLY, F-test is equivalent to t-test for  $\beta_1$ .



## Regression in R (Used cars example)

```
Price<-c(85, 103, 70, 82, 89, 98, 66, 95, 169, 70, 48)
Age<- c(5, 4, 6, 5, 5, 5, 6, 6, 2, 7, 7)
carSales<-data.frame(Price, Age)
str(carSales)
```

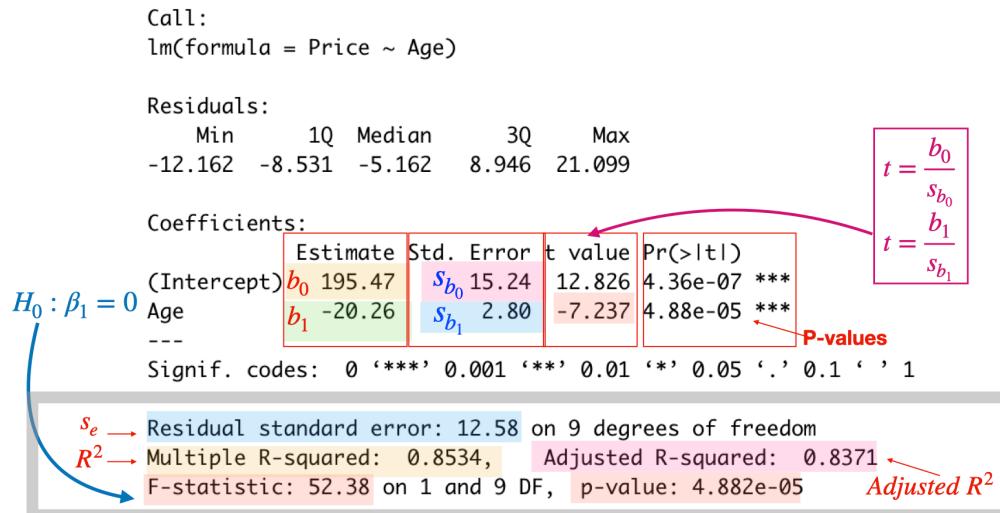
```
## 'data.frame': 11 obs. of 2 variables:
## $ Price: num 85 103 70 82 89 98 66 95 169 70 ...
## $ Age : num 5 4 6 5 5 5 6 6 2 7 ...
# simple linear regression
reg<-lm(Price~Age)
summary(reg)
```

```
##
## Call:
## lm(formula = Price ~ Age)
##
## Residuals:
##      Min    1Q Median    3Q   Max
## -12.162 -8.531 -5.162  8.946 21.099
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 195.47     15.24 12.826 0.000000436 ***
## Age         -20.26      2.80 -7.237 0.000048819 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.58 on 9 degrees of freedom
## Multiple R-squared: 0.8534, Adjusted R-squared: 0.8371
## F-statistic: 52.38 on 1 and 9 DF, p-value: 0.00004882
```

```
# To obtain the confidence intervals
confint(reg, level=0.95)
```

```
##                2.5 %    97.5 %
## (Intercept) 160.99243 229.94451
## Age         -26.59419 -13.92833
```

## R output



## 3.5 Simple Linear Regression: Confidence and Prediction intervals

Earlier we introduced simple linear regression as a basic statistical model for the relationship between two random variables. We used the least square method for estimating the regression parameters.

Recall that the simple linear regression model for  $Y$  on  $x$  is

$$Y = \beta_0 + \beta_1 x + \epsilon$$

where

$Y$  : the dependent or response variable

$x$  : the independent or predictor variable, assumed known

$\beta_0, \beta_1$  : the regression parameters, the intercept and the slope of the regression line

$\epsilon$  : the random regression error around the line.

and the regression equation for a set of  $n$  data points is  $\hat{y} = b_0 + b_1 x$ , where

$$b_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

and

$$b_0 = \bar{y} - b_1 \bar{x}$$

where  $b_0$  is called the **y-intercept** and  $b_1$  is called the **slope**.

**Under the simple linear regression assumptions**, the residual standard error  $s_e$  is an unbiased estimate for the error standard deviation  $\sigma$ , where

$$s_e = \sqrt{\frac{SSE}{n-2}} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}}$$

$s_e$  indicates how much, on average, the observed values of the response variable differ from the predicted values of the response variable.

Below we will see how we can use these least square estimates for prediction. First, we will consider the inference for the conditional mean of the response variable  $y$  given a particular value of the independent variable  $x$ , let us call this particular value  $x^*$ . Next, we will see how to predict the value of the response variable  $Y$  for a given value of the independent variable  $x^*$ . For these confidence and predictive intervals, to be valid, the usual four simple regression assumptions must hold.

### Inference for the regression line $E[Y|x^*]$

Suppose we are interested in the value of the regression line at a new point  $x^*$ . Let's denote the unknown true value of the regression line at  $x = x^*$  as  $\mu^*$ . From the form of the regression line equation we have

$$\mu^* = \mu_{Y|x^*} = E[Y|x^*] = \beta_0 + \beta_1 x^*$$

but  $\beta_0$  and  $\beta_1$  are unknown. We can use the least square regression equation to estimate the unknown true value of the regression line, so we have

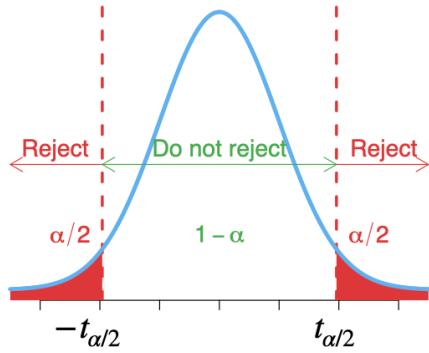
$$\hat{\mu}^* = b_0 + b_1 x^* = \hat{y}^*$$

This is simply a point estimate for the regression line. However, in statistics, a point estimate is often not enough, and we need to express our uncertainty about this point estimate, and one way to do so is via confidence interval.

A  $100(1 - \alpha)\%$  confidence interval for the conditional mean  $\mu^*$  is

$$\hat{y}^* \pm t_{\alpha/2} \cdot s_e \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}}$$

where  $S_{xx} = \sum_{i=1}^n (x_i - \bar{x})^2$ , and  $t_{\alpha/2}$  is the  $\alpha/2$  critical value from the t-distribution with  $df = n - 2$ .



### Inference for the response variable $Y$ for a given $x = x^*$

Suppose now we are interested in predicting the value of  $Y^*$  if we have a new observation at  $x^*$ .

At  $x = x^*$ , the value of  $Y^*$  is unknown and given by

$$Y^* = \beta_0 + \beta_1 x^* + \epsilon$$

where but  $\beta_0$ ,  $\beta_1$  and  $\epsilon$  are unknown. We will use  $\hat{y}^* = b_0 + b_1 x^*$  as a basis for our prediction.

A  $100(1 - \alpha)\%$  prediction interval for  $Y^*$  at  $x = x^*$  is

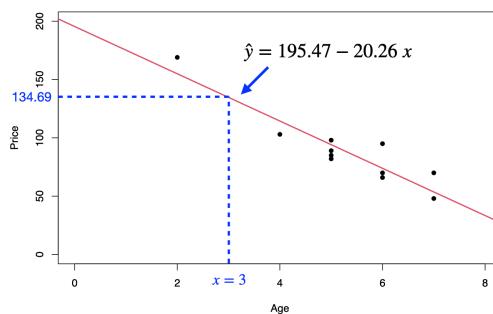
$$\hat{y}^* \pm t_{\alpha/2} \cdot s_e \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}}$$

The extra '1' under the square root sign, we have here accounts for the extra variability of a single observation about the mean.

### Used cars example

Estimate the mean price of all 3-year-old cars,  $E[Y|x = 3]$ :

$$\hat{\mu}^* = 195.47 - 20.26(3) = 134.69 = \hat{y}^*$$



A 95% confidence interval for the mean price of all 3-year-old cars is

$$\hat{y}^* \pm t_{\alpha/2} \times se \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}}$$

$$[195.47 - 20.26(3)] \pm 2.262 \times 12.58 \sqrt{\frac{1}{11} + \frac{(3 - 5.273)^2}{(11 - 1) \times 2.018}}$$

$$134.69 \pm 16.76$$

that is

$$117.93 < \mu^* < 151.45$$

**Predict the price of a 3-year-old car,  $Y|x = 3$ :**

$$\hat{y}^* = 195.47 - 20.26(3) = 134.69$$

A 95% predictive interval for the price of a 3-year-old car is

$$\hat{y}^* \pm t_{\alpha/2} \times se \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}}$$

$$[195.47 - 20.26(3)] \pm 2.262 \times 12.58 \sqrt{1 + \frac{1}{11} + \frac{(3 - 5.273)^2}{(11 - 1) \times 2.018}}$$

$$134.69 \pm 33.025$$

that is

$$101.67 < Y^* < 167.72$$

where  $S_{xx} = \sum_{i=1}^n (x_i - \bar{x})^2 = (n - 1)Var(x)$ .

## Regression in R

```
# Build linear model
Price<-c(85, 103, 70, 82, 89, 98, 66, 95, 169, 70, 48)
Age<- c(5, 4, 6, 5, 5, 6, 6, 2, 7, 7)
carSales<-data.frame(Price=Price,Age=Age)

reg <- lm(Price~Age,data=carSales)
summary(reg)
```

```
##
## Call:
## lm(formula = Price ~ Age, data = carSales)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -100.000 -50.000 -10.000  40.000 100.000
```

```

## -12.162 -8.531 -5.162  8.946 21.099
##
## Coefficients:
##             Estimate Std. Error t value    Pr(>|t|)
## (Intercept) 195.47     15.24 12.826 0.000000436 ***
## Age         -20.26      2.80 -7.237 0.000048819 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.58 on 9 degrees of freedom
## Multiple R-squared:  0.8534, Adjusted R-squared:  0.8371
## F-statistic: 52.38 on 1 and 9 DF,  p-value: 0.00004882
mean(Age)

```

```
## [1] 5.272727
```

```
var(Age)
```

```
## [1] 2.018182
```

```
qt(0.975,9)
```

```
## [1] 2.262157
```

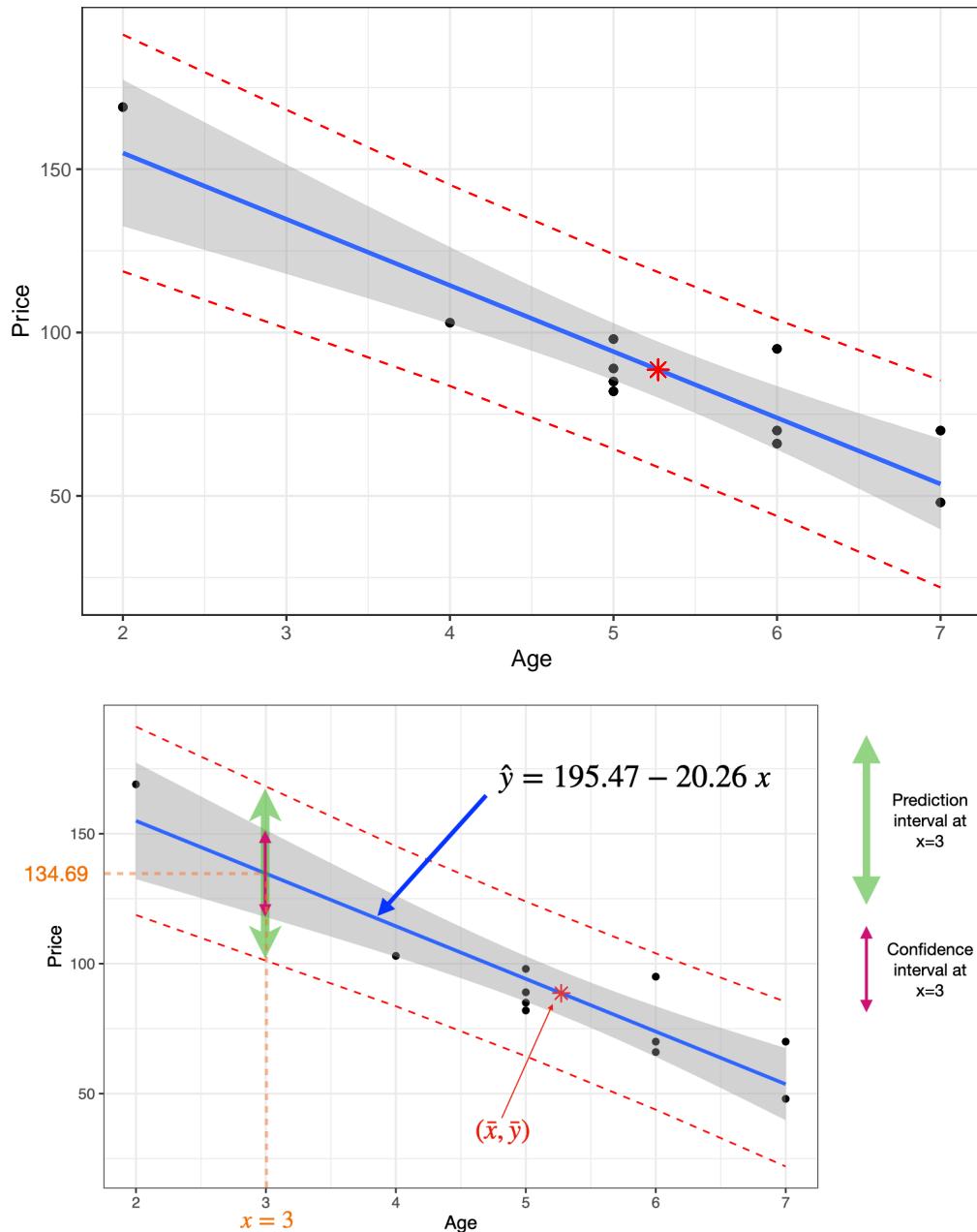
```
newage<- data.frame(Age = 3)
predict(reg, newdata = newage, interval = "confidence")
```

```
##       fit      lwr      upr
## 1 134.6847 117.9293 151.4401
```

```
predict(reg, newdata = newage, interval = "prediction")
```

```
##       fit      lwr      upr
## 1 134.6847 101.6672 167.7022
```

We can plot the confidence and prediction intervals as follows:



## 3.6 Multiple Linear Regression: Introduction

### Multiple linear regression model

In simple linear regression, we have one dependent variable ( $y$ ) and one independent variable ( $x$ ). In multiple linear regression, we have one dependent variable ( $y$ ) and several independent variables ( $x_1, x_2, \dots, x_k$ ).

- The multiple linear regression model, for the **population**, can be expressed as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon$$

where  $\epsilon$  is the error term.

Or using the matrix notation for a sample of  $n$  observations:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,k} \\ 1 & x_{2,1} & \dots & x_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \dots & x_{n,k} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

i.e.

$$\mathbf{Y} = \mathbf{X}\beta +$$

where  $[\mathbf{Y}^T] = (y_1, \dots, y_n)$  is the vector of responses;  $[\mathbf{X}]$ , is the design matrix;  $[\beta^T] = (\beta_0, \beta_1, \dots, \beta_k)$  is the  $k+1$  dimensional parameter vector;  $[\epsilon^T] = (\epsilon_1, \dots, \epsilon_n)$  is the vector of errors.

- The corresponding least square estimate, from the **sample**, of this multiple linear regression model is given by

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_k x_k$$

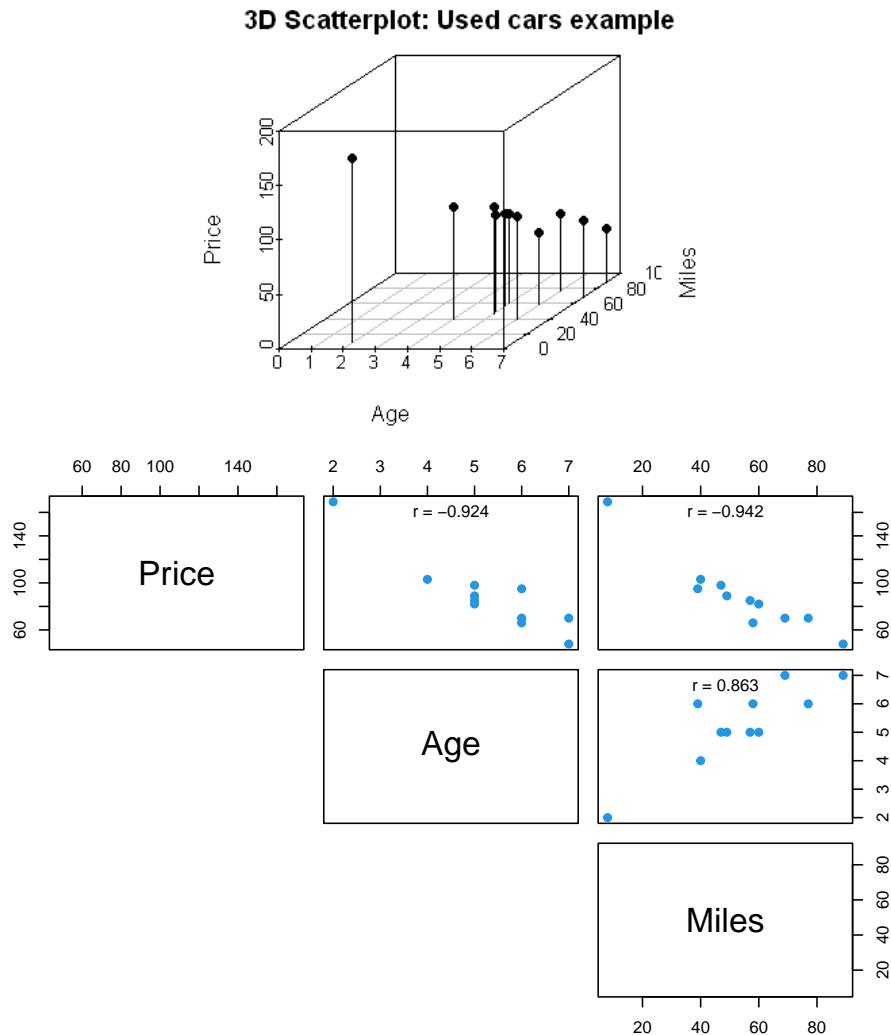
The LS estimator  $\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$

- The coefficient  $b_0$  (or  $\hat{\beta}_0$ ) represents the  $y$ -intercept, that is, the value of  $y$  when  $x_1 = x_2 = \dots = x_k = 0$ . The coefficient  $b_i$  (or  $\hat{\beta}_i$ ) ( $i = 1, \dots, k$ ) is the partial slope of  $x_i$ , holding all other  $x$ 's fixed. So  $b_i$  (or  $\hat{\beta}_i$ ) tells us the change in  $y$  for a unit increase in  $x_i$ , holding all other  $x$ 's fixed.

## Used cars example

The table below displays data on Age, Miles and Price for a sample of cars of a particular make and model.

Price ( $y$ )	Age ( $x_1$ )	Miles ( $x_2$ )
85	5	57
103	4	40
70	6	77
82	5	60
89	5	49
98	5	47
66	6	58
95	6	39
169	2	8
70	7	69
48	7	89



The scatterplot and the correlation matrix show a fairly negative relationship between the price of the car and both independent variables (age and miles). It is desirable to have a relationship between each independent variable and the dependent variable. However, the scatterplot also shows a positive relationship between the age and the miles, which is undesirable as it will cause the issue of Multicollinearity.

### Coefficient of determination, $R^2$ and adjusted $R^2$

- Recall that  $R^2$  is a measure of the proportion of the total variation in the observed values of the response variable that is explained by the multiple linear regression in the  $k$  predictor variables  $x_1, x_2, \dots, x_k$ .
- $R^2$  will increase when an additional predictor variable is added to the model. One should not simply select a model with many predictor variables because it has the highest  $R^2$  value, it is often good to have a model with a high  $R^2$  value but only a few x's included.
- Adjusted  $R^2$  is a modification of  $R^2$  that takes into account the number of predictor variables.

$$\text{Adjusted-}R^2 = 1 - (1 - R^2) \frac{n - 1}{n - k - 1}$$

## The residual standard error, $s_e$

- Recall that,

$$\text{Residual} = \text{Observed value} - \text{Predicted value.}$$

$$e_i = y_i - \hat{y}_i$$

- In a multiple linear regression with  $k$  predictors, the standard error of the estimate,  $s_e$ , is defined by

$$s_e = \sqrt{\frac{SSE}{n - (k + 1)}} \quad \text{where } SSE = \sum(y_i - \hat{y}_i)^2$$

- The standard error of the estimate,  $s_e$ , indicates how much, on average, the observed values of the response variable differs from the predicted values of the response variable. The  $s_e$  is the estimate of the common standard deviation  $\sigma$ .

## Inferences about a particular predictor variable

- To test whether a particular predictor variable, say  $x_i$ , is useful for predicting  $y$  we test the null hypothesis  $H_0 : \beta_i = 0$  against  $H_1 : \beta_i \neq 0$ .
- The test statistic

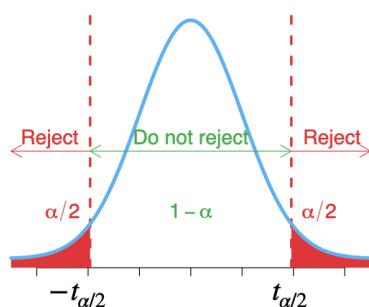
$$t = \frac{b_i}{s_{b_i}}$$

has a  $t$ -distribution with degrees of freedom  $df = n - (k + 1)$ . So we reject  $H_0$ , at level  $\alpha$ , if  $|t| > t_{\alpha/2}$ .

- Rejection of the null hypothesis indicates that  $x_i$  is useful as a predictor for  $y$ . However, failing to reject the null hypothesis suggests that  $x_i$  may not be useful as a predictor of  $y$ , so we may want to consider removing this variable from the regression analysis.
- $100(1-\alpha)\%$  confidence interval for  $\beta_i$  is

$$b_i \pm t_{\alpha/2} \cdot s_{b_i}$$

where  $s_{b_i}$  is the standard error of  $b_i$ .



## Is the multiple regression model “useful”?

### Goodness of fit test

To test how useful is this model, we test the null hypothesis

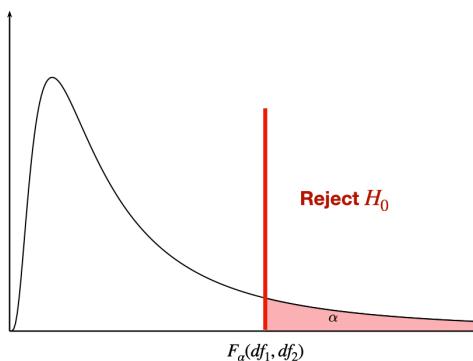
$$H_0 : \beta_1 = \beta_2 = \dots = \beta_k = 0, \text{ against}$$

$H_1$  : at least one of the  $\beta_i$ 's is not zero. - The  $F$ -statistic

$$F = \frac{MSR}{MSE} = \frac{SSR/k}{SSE/(n - k - 1)}$$

with degrees of freedom  $df_1 = k$  and  $df_2 = n - (k + 1)$ .

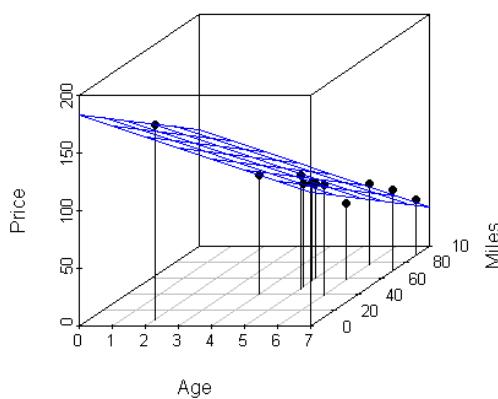
We reject  $H_0$ , at level  $\alpha$ , if  $F > F_\alpha(df_1, df_2)$ .



## Used cars example continued

Multiple regression equation:  $\hat{y} = 183.04 - 9.50x_1 - 0.82x_2$

3D Scatterplot: Used cars example



The predicted price for a 4-year-old car that has driven 45 thousand miles is

$$\hat{y} = 183.04 - 9.50(4) - 0.82(45) = 108.14$$

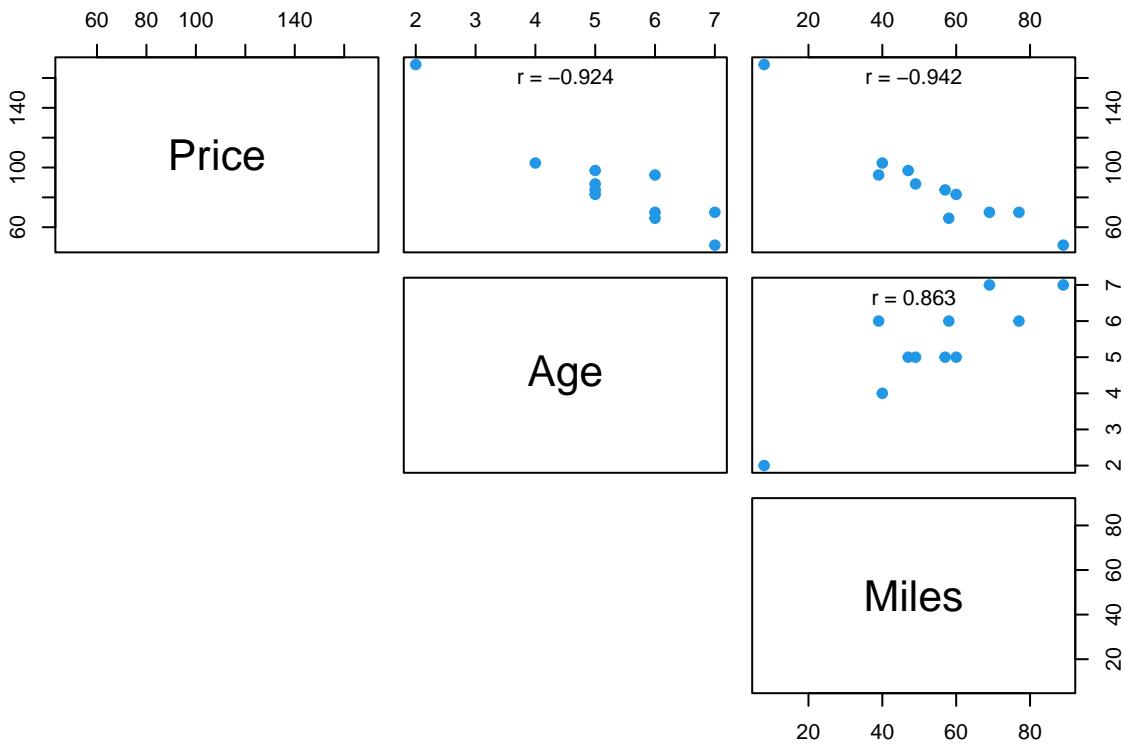
(as units of \$100 were used, this means \$10814)

**Extrapolation:** we need to look at the region (all combined values) not only the range of the observed values of each predictor variable separately.

## Regression in R

```
Price<-c(85, 103, 70, 82, 89, 98, 66, 95, 169, 70, 48)
Age<- c(5, 4, 6, 5, 5, 6, 6, 2, 7, 7)
Miles<-c(57,40,77,60,49,47,58,39,8,69,89)
carSales<-data.frame(Price=Price,Age=Age,Miles=Miles)
```

```
# Scatterplot matrix
# Customize upper panel
upper.panel<-function(x, y){
  points(x,y, pch=19, col=4)
  r <- round(cor(x, y), digits=3)
  txt <- paste0("r = ", r)
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  text(0.5, 0.9, txt)
}
pairs(carSales, lower.panel = NULL,
      upper.panel = upper.panel)
```



```
reg <- lm(Price~Age+Miles,data=carSales)
summary(reg)
```

```
##
## Call:
## lm(formula = Price ~ Age + Miles, data = carSales)
##
```

```

## Residuals:
##      Min     1Q Median     3Q    Max
## -12.364 -5.243  1.028  5.926 11.545
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 183.0352   11.3476 16.130 0.000000219 ***
## Age         -9.5043    3.8742 -2.453  0.0397 *
## Miles       -0.8215    0.2552 -3.219  0.0123 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.805 on 8 degrees of freedom
## Multiple R-squared:  0.9361, Adjusted R-squared:  0.9201
## F-statistic: 58.61 on 2 and 8 DF,  p-value: 0.00001666

confint(reg, level=0.95)

##               2.5 %      97.5 %
## (Intercept) 156.867552 209.2028630
## Age          -18.438166 -0.5703751
## Miles        -1.409991 -0.2329757

```

## Summary

$\hat{y} = 183.04 - 9.50 x_1 - 0.82 x_2$

Coefficients:					
	Estimate	Std. Error	t value	Pr(> t )	
(Intercept)	183.0352	11.3476	16.130	2.19e-07 ***	
Age	-9.5043	3.8742	-2.453	0.0397 *	
Miles	-0.8215	0.2552	-3.219	0.0123 *	
---					
Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'	0.1 ' '

$S_e$  Residual standard error: 8.805 on 8 degrees of freedom  
Multiple R-squared: 0.9361, Adjusted R-squared: 0.9201  
F-statistic: 58.61 on 2 and 8 DF, p-value: 1.666e-05

$H_0 : \beta_1 = \beta_2 = 0$

	$\beta_0$	$\beta_1$	$\beta_2$
$H_0$	$H_0 : \beta_0 = 0$	$H_0 : \beta_1 = 0$	$H_0 : \beta_2 = 0$
$H_1$	$H_1 : \beta_0 \neq 0$	$H_1 : \beta_1 \neq 0$	$H_1 : \beta_2 \neq 0$
Estimate of $\beta_i$	$b_0 = 183.04$	$b_1 = 9.50$	$b_2 = 0.82$
$t = \frac{b_i}{s_{b_i}}$	16.130	-2.453	-3.219
P-value	0	0.040	0.012
Decision*	reject $H_0$	reject $H_0$	reject $H_0$
95% CI for $\beta_i$	(156.868, 209.203)	(-18.438, -0.570)	(-1.410, -0.233)

\* at  $\alpha = 0.05$ .

## Multiple Linear Regression Assumptions

- **Linearity:** For each set of values,  $x_1, x_2, \dots, x_k$ , of the predictor variables, the conditional mean of the response variable  $y$  is  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$ .
- **Equal variance (homoscedasticity):** The conditional variance of the response variable are the same (equal to  $\sigma^2$ ) for all sets of values,  $x_1, x_2, \dots, x_k$ , of the predictor variables.
- **Independent observations:** The observations of the response variable are independent of one another.
- **Normally:** For each set values,  $x_1, x_2, \dots, x_k$ , of the predictor variables, the conditional distribution of the response variable is a normal distribution.
- **No Multicollinearity:** Multicollinearity exists when two or more of the predictor variables are highly correlated.

## Multicollinearity

- Multicollinearity refers to a situation when two or more predictor variables in our multiple regression model are highly (linearly) correlated.
- The least square estimates will remain unbiased, but unstable.
- The standard errors (of the affected variables) are likely to be high.
- Overall model fit (e.g. R-square, F, prediction) is not affected.

## Multicollinearity: Detect

- Scatterplot Matrix
- **Variance Inflation Factors:** the Variance Inflation Factors (VIF) for the  $i^{th}$  predictor is

$$VIF_i = \frac{1}{1 - R_i^2}$$

where  $R_i^2$  is the R-square value obtained by regressing the  $i^{th}$  predictor on the other predictor variables.

- $VIF = 1$  indicates that there is no correlation between  $i^{th}$  predictor variable and the other predictor variables.
- As a rule of thumb if  $VIF > 10$  then multicollinearity could be a problem.

## Multicollinearity: How to fix?

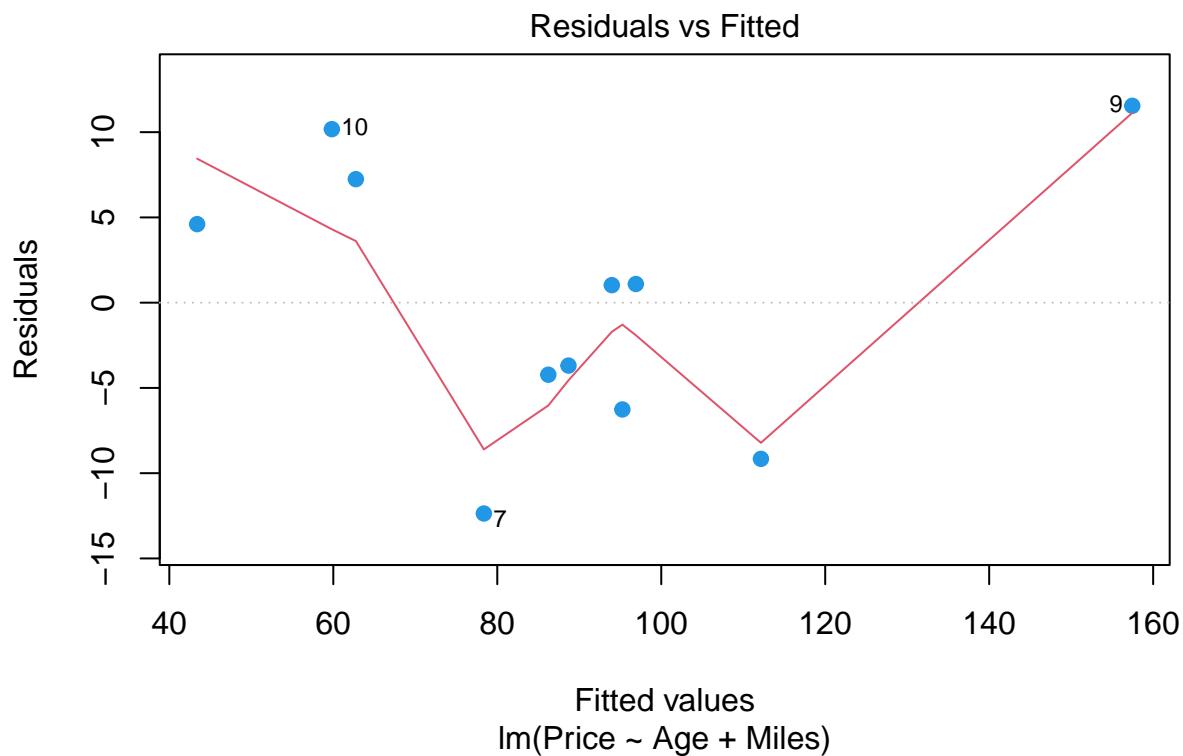
**Ignore:** if the model is going to be used for prediction only.

**Remove:** e.g. see if the variables are providing the same information.

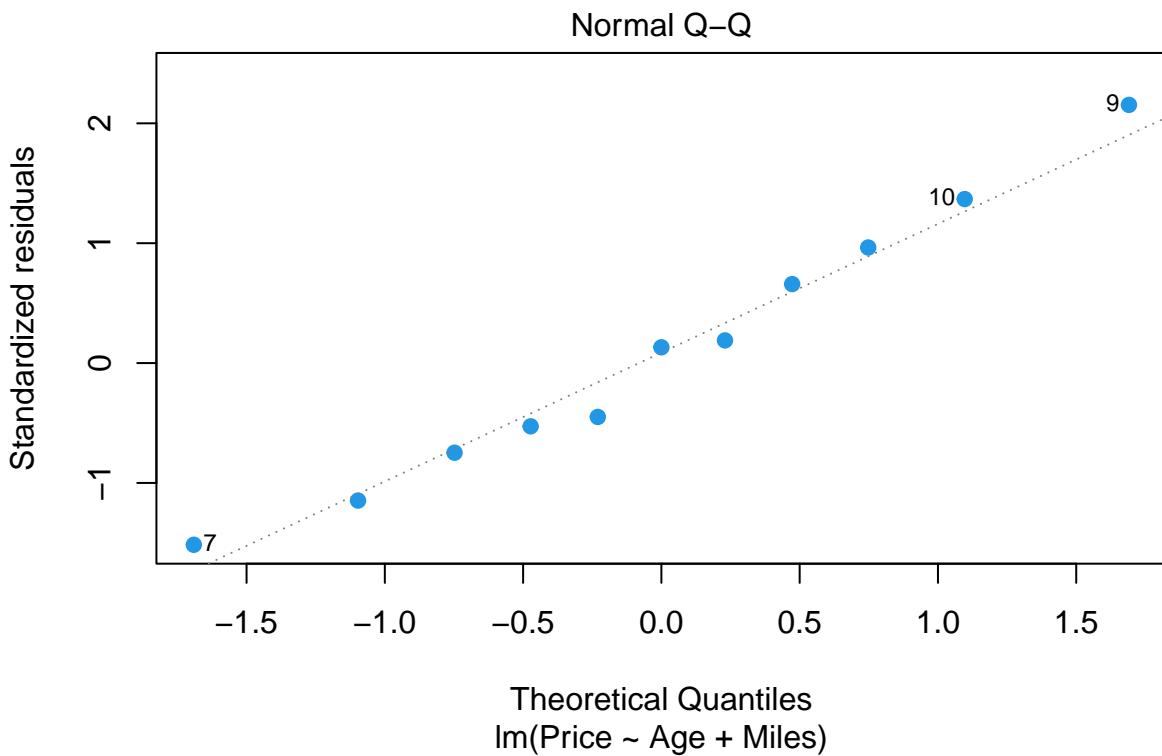
**Combine:** combining highly correlated variables.

**Advanced:** e.g. Principal Components Analysis, Partial Least Squares.

```
plot(reg, which=1, pch=19, col=4)
Regression in R (regression assumptions)
```



```
plot(reg, which=2, pch=19, col=4)
```



```
# install.packages("car")
library(car)
vif(reg)
```

```
##          Age      Miles
## 3.907129 3.907129
```

The value of  $VIF = 3.91$  indicates a moderate correlation between the age and the miles in the model, but this is not a major concern.



# Chapter 4

## Model Assessment and Selection

Assessment of model performance is extremely important in practice, since it guides the choice of machine learning algorithm or model, and gives us a measure of the quality of the ultimately chosen model.

It is important to note that there are in fact two separate goals here:

- *Model selection*: estimating the performance of different models in order to choose the best one.
- *Model assessment*: having chosen a final model, estimating its *prediction error* (*generalization error*) over an independent new data sample.

### 4.1 In-sample vs out-of-sample

If we are in a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts: a *training set*, a *validation/evaluation set*, and a *test set*.

- The training set is used to fit the model parameters.
- The validation set is used to estimate prediction error for model selection (including choosing the values of hyperparameters).
- The test set is used for assessment of the generalization error (also referred to as *test error*, is the prediction error over an independent test sample.) of the final chosen model.

The period that the training set and the validation set are used for the initial parameter estimation and model selection, is called *in-sample* period. And an *out-of-sample* period uses test set to evaluate final forecasting performance.

Ideally, the test set should be kept in a “vault” and be brought out only at the end of the data analysis. Suppose instead that we use the test-set repeatedly, choosing the model with smallest test-set error. Then the test set error of the final chosen model will underestimate the true test error, sometimes substantially.

It is difficult to give a general rule on how to choose the number of observations in each of the three parts. A typical split might be 50% for training, and 25% each for validation and

testing.



## 4.2 Cross-Validation

For the situations where there is insufficient data to split it into three parts. One could conduct  $k$ -fold Cross-validation on a single training set for both training and validation.

*k-fold Cross-validation* (CV) involves randomly dividing the set of  $n$  observations into  $k$  groups (or folds) of approximately equal size.

Then, for each group  $i = 1, \dots, k$ :

- Fold  $i$  is treated as a validation set, and the model is fit on the remaining  $k - 1$  folds.
- The performance metric,  $Performance_i$  (for example MSE), is then computed based on the observations of the held out fold  $i$ .

This process results in  $k$  estimates of the test performance,  $Performance_1, \dots, Performance_k$ . The  $k$ -fold CV estimate is computed by averaging these values.

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^k Performance_i$$

Figure 4.1 illustrates this nicely.

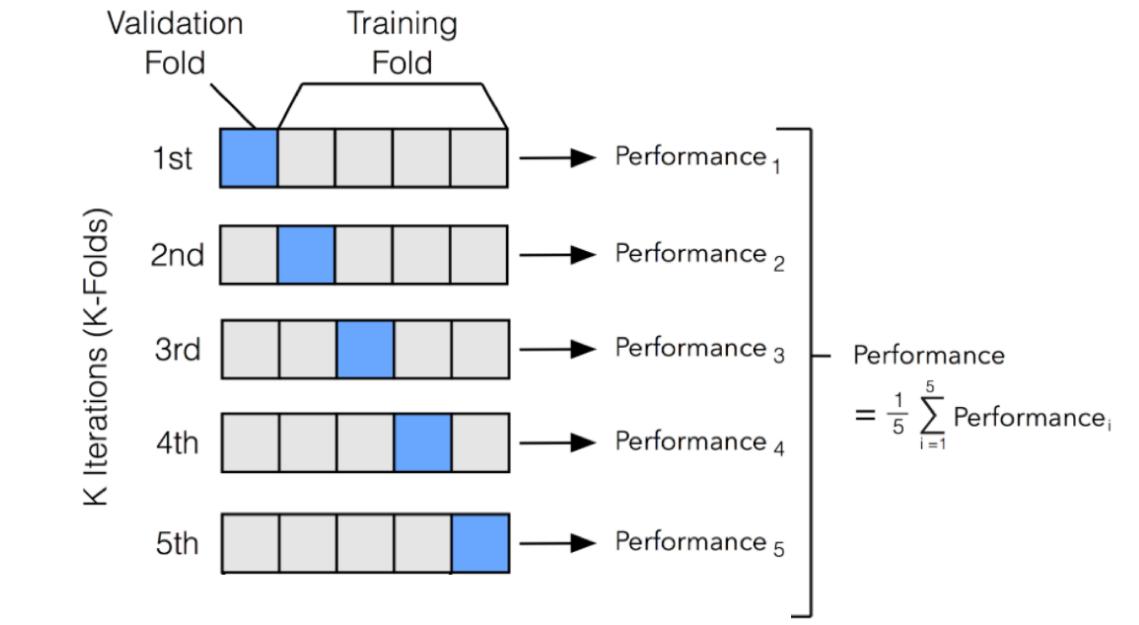
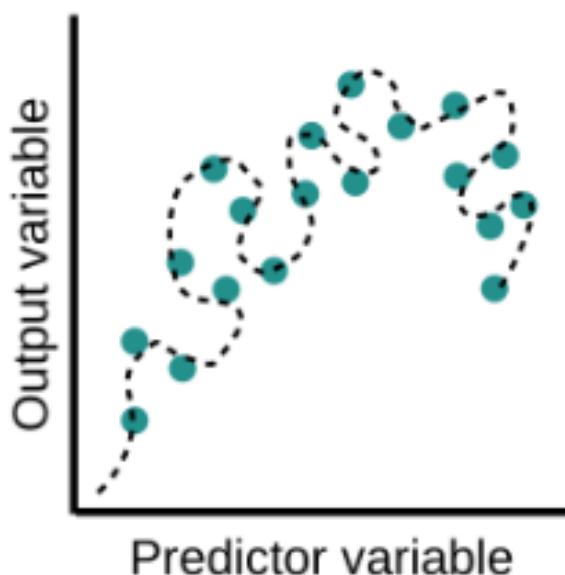
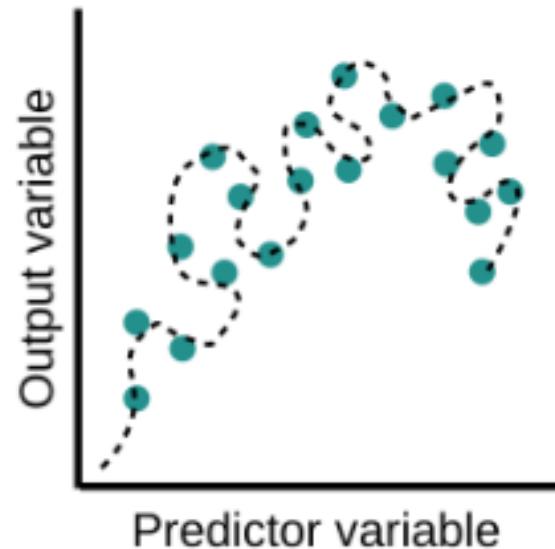


Figure 4.1: An illustration of  $k$ -fold CV with 5 folds.

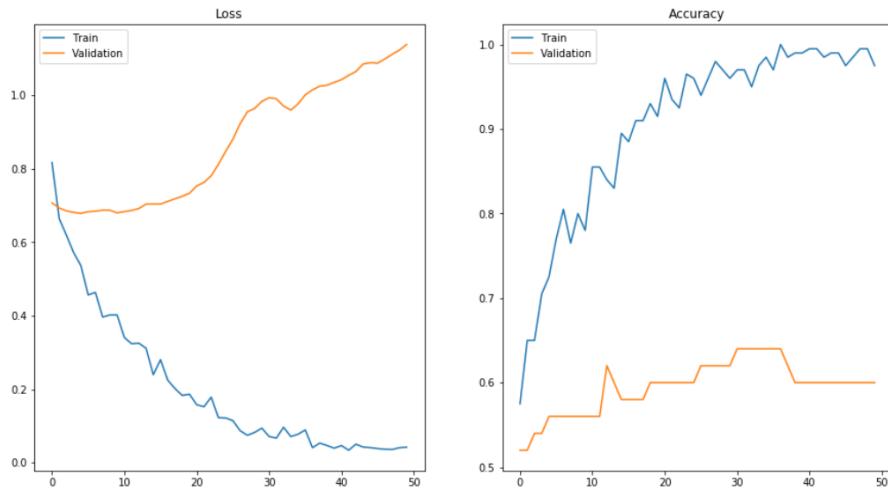
The  $CV_{(k)}$  as an estimate of the test performance can also be used for model selection. Note that to assess the performance of the final chosen model, one still need another independent test sample.

## 4.3 Overfitting vs Underfitting

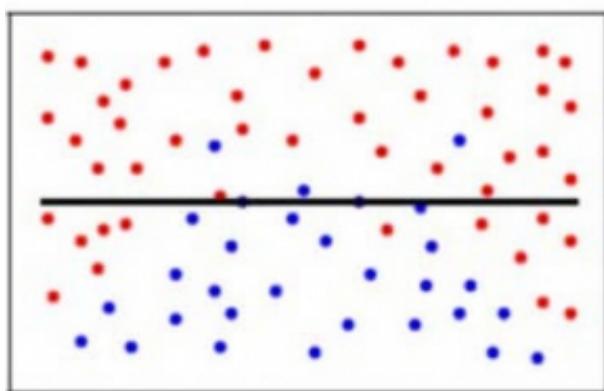
*Overfitting* is a common pitfall in machine learning modelling, in which a model tries to fit the training data entirely and ends up “memorizing” the data patterns and the noise/random fluctuations. These models fail to generalize and perform well in the case of unseen data scenarios, defeating the model’s purpose. That is why an overfit model results in poor test accuracy. Example of overfitting situation in classification and regression:

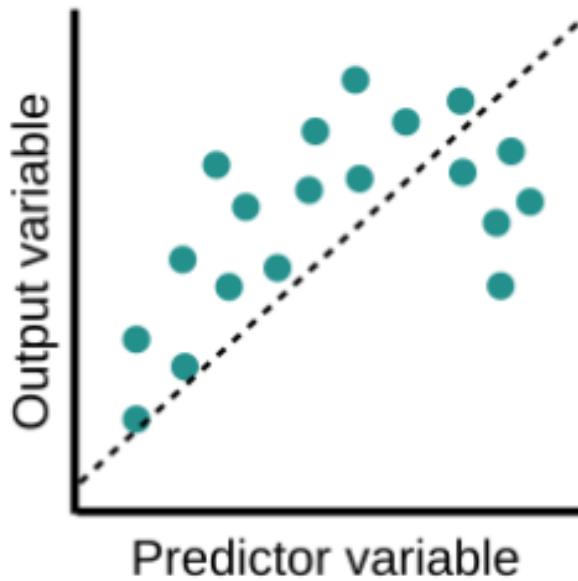


Detecting overfitting is only possible once we move out of the training phase, for example evaluate the model performance using the validation set.



*Underfitting* is another common pitfall in machine learning modelling, where the model cannot create a mapping between the input and the target variable that reflects the underlying system, for example due to under-observing the features. Underfitting often leads to a higher error in the training and unseen data samples. Example of overfitting situation in classification and regression:





Underfitting becomes obvious when the model is too simple and cannot represent a relationship between the input and the output. It is detected when the training error is very high and the model is unable to learn from the training data.

## 4.4 Bias Variance Trade-off

In supervised learning, the model performance can help us to identify or even quantify overfitting/underfitting. Often we use the difference between the actual values and predicted values to evaluate the model, such prediction error can in fact be decomposed into three parts:

- Bias: The difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.
- Variance: Variance is the variability of model prediction for a given data point or a value which tells us how uncertainty our model is. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.
- Noise: Irreducible error that we cannot eliminate.

Mathematically, assume the relationship between the response  $Y$  and the predictors  $X = (X_1, \dots, X_p)$  can be represented as:

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

where  $f$  is some fixed but unknown function of  $X$  and  $\epsilon$  is a random error term, which is independent of  $X$  and has mean zero.

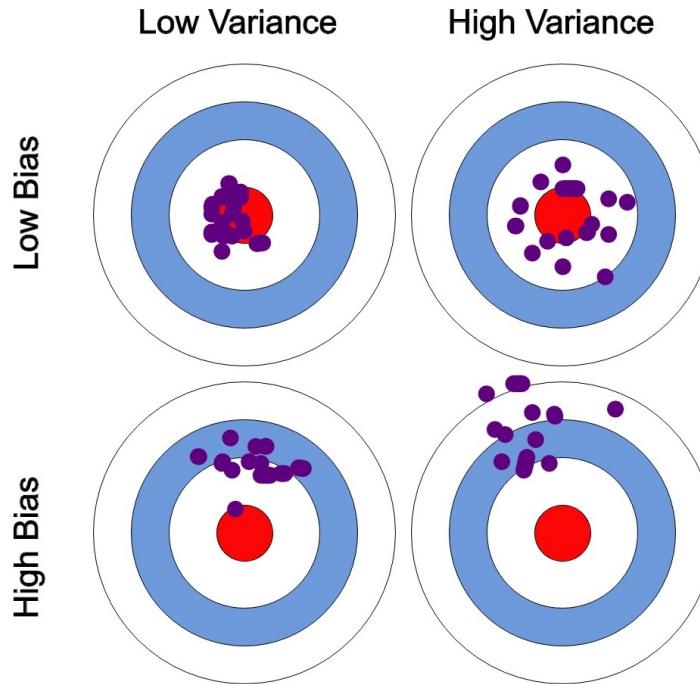
Consider building a model  $\hat{f}(X)$  of  $f(X)$  (for example a linear regression model), the expected

squared error (MSE) at a point  $x$  is

$$\begin{aligned}
 MSE &= E \left[ (y - \hat{f}(x))^2 \right] \\
 &= E \left[ (f(x) + \epsilon - \hat{f}(x))^2 \right] \\
 &= E \left[ (f(x) + \epsilon - \hat{f}(x) + E[\hat{f}(x)] - E[\hat{f}(x)])^2 \right] \\
 &= \dots \\
 &= (f(x) - E[\hat{f}(x)])^2 + E[\epsilon^2] + E[(E[\hat{f}(x)] - \hat{f}(x))^2] \\
 &= (f(x) - E[\hat{f}(x)])^2 + Var[\epsilon] + Var[\hat{f}(x)] \\
 &= Bias[\hat{f}(x)]^2 + Var[\epsilon] + Var[\hat{f}(x)] \\
 &= Bias^2 + Variance + Irreducible\ Error
 \end{aligned} \tag{4.1}$$

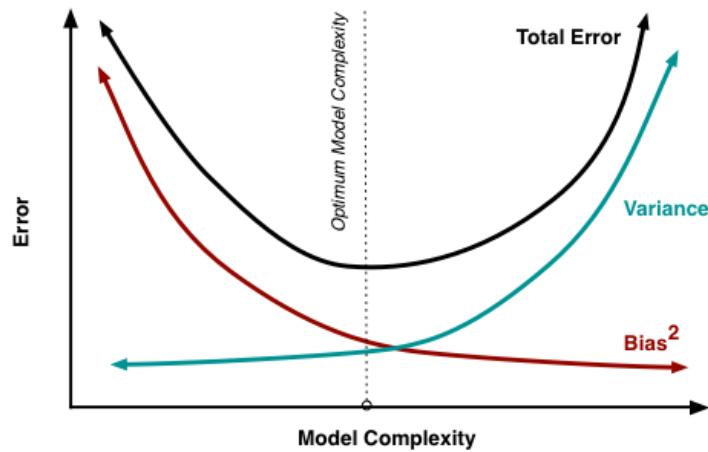
We can create a graphical visualization of bias and variance using a bulls-eye diagram. Imagine that the center of the target is a model that perfectly predicts the correct values. As we move away from the bulls-eye, our predictions get worse and worse.

We can plot four different cases representing combinations of both high and low bias and variance.

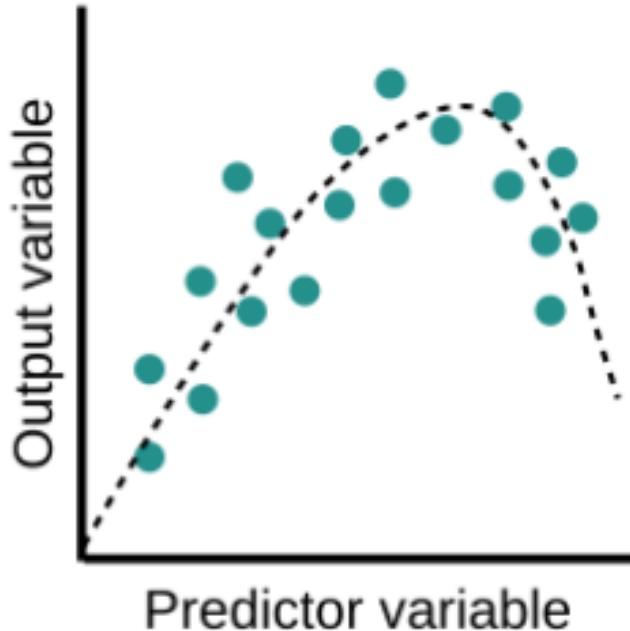
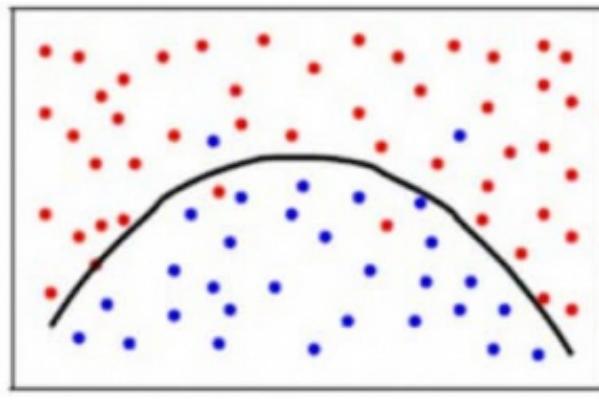


At its root, dealing with bias and variance is really about dealing with overfitting and underfitting. High bias and low variance are the most common indicators of underfitting. Similarly low bias and high variance are the most common indicators of overfitting. Bias is reduced and variance is increased in relation to model complexity. As more and more parameters

are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls.



An optimal balance of bias and variance would neither overfit nor underfit the model. Example of an optimal balanced situation in classification and regression:



## 4.5 Model selection for Linear regression

Consider the standard linear regression model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$

Here,  $y$  is the *response* variable, the  $x$ 's are the *predictor* variables and  $\epsilon$  is an *error* term.

Model selection involves identifying a subset of the  $p$  predictors  $x_1, \dots, x_p$  of size  $d$  that we believe to be related to the response. We then fit a *least squares linear regression model* using just the reduced set of variables.

For example, if we believe that only  $x_1$  and  $x_2$  are related to the response, then we may take  $d = 2$  and fit a model of the following form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where we have deemed variables  $x_3, \dots, x_p$  to be irrelevant.

*But how do we determine which variables are relevant?* We may just ‘know’ which variables are most informative for a particular response (for example, from previous data investigations or talking to an expert in the field), but often we need to investigate.

*How many possible linear models are we trying to choose from?* Well, we will always include the intercept term  $\beta_0$ . Then each variable  $X_1, \dots, X_p$  can either be included or not, hence we have  $2^p$  possible models to choose from.

**Question:** *how do we choose which variables should be included in our model?*

### Training Error

The *training error* of a linear model is evaluated via the *Residual Sum of Squares (RSS)*, which as we have seen is given by

$$RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2,$$

where

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_d x_{id}.$$

is the predicted response value at input  $x_i = (x_{i1}, \dots, x_{id})$ . Or simply scaling down the RSS leads to the Mean Squared Error (MSE):

$$MSE = \frac{RSS}{n}$$

where  $n$  is the size of the training sample. Unfortunately, neither of these metrics is appropriate when comparing models of different dimensionality (different number of predictors) because both *RSS* and *MSE* generally decrease as we include additional predictors to a linear model. In fact, in the extreme case of  $n = p$  both metrics will be 0! This does not mean that we have a “good” model, just that we have *overfitted* our linear model to perfectly adjust to the training data. Overfitted models will exhibit poor predictive performance (low training error but high prediction error). Our aim is to have simple, interpretable models with relatively small  $p$  (in relation to  $n$ ), which have good predictive performance.

## Validation

We could simply use an independent validation set (either by splitting the data into training and validation set or using cross-validation techniques) to evaluate the model performance and select the “best” model.

For example:

Let us assume a hypothetical scenario where the true model is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where  $\beta_0 = 1$ ,  $\beta_1 = 1.5$ ,  $\beta_2 = -0.5$  for some given values of the predictors  $x_1$  and  $x_2$ . Let’s set the errors to be normally distributed with zero mean and variance equal to one; i.e.,  $\epsilon \sim N(0, 1)$ . Also assume that we have three additional predictors  $x_3$ ,  $x_4$  and  $x_5$  which are irrelevant to  $y$  (in the sense that  $\beta_3 = \beta_4 = \beta_5 = 0$ ), but of course we do not know that beforehand.

We can plot the training MSE and prediction MSE against the number of predictors used in the model, such as is illustrated in Figure 4.2.

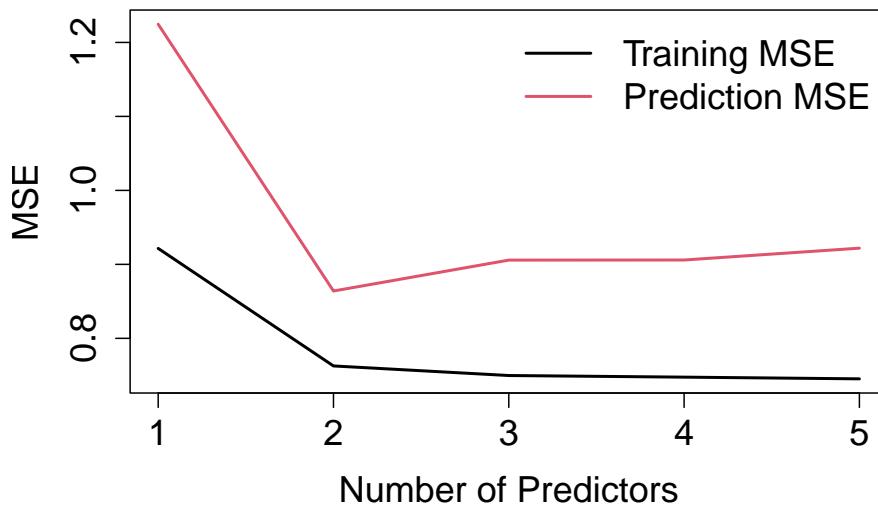


Figure 4.2: Training MSE and prediction MSE against number of predictors.

Notice the following:

- Training error: steady (negligible) decline after adding  $x_3$  – not really obvious how many predictors to include.
- Prediction error: increase after adding  $x_3$ , which indicates overfitting. Clearly here one would select the model with  $x_1$  and  $x_2$ !
- Prediction errors are larger than training errors – this is generally always the case! (see Q8 in Problem Sheet 1).

## 4.6 Model Selection Criteria

Given that in practice we may not wish to exclude part of the data for calculating a prediction error using validation, instead of using cross-validation techniques we can also *indirectly*

estimate prediction error by making an *adjustment* to the performance measure that accounts for overfitting. Here, we have several model selection criteria that are developed from such kind of adjustment.

## $C_p$

For a given model with  $d$  predictors (out of the available  $p$  predictors) Mallows'  $C_p$  criterion is

$$C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2),$$

where  $RSS$  is the residual sum of squares for the model of interest (with  $d$  predictors), and  $\hat{\sigma}^2 = RSS_p/(n - p - 1)$  is an estimate of the error variance for the full model with all  $p$  possible predictors included. As such, we use  $RSS_p$  to denote the Residual Sum of Squares for the full model with all  $p$  possible predictors included.

In practice, we choose the model which has the *minimum*  $C_p$  value: so we essentially *penalise* models of higher dimensionality (the larger  $d$  is the greater the penalty).

## AIC

For linear models (with normal errors, as is often assumed), Mallows'  $C_p$  is equivalent to the *Akaike Information Criterion (AIC)* (as the two are proportional), this being given by

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2).$$

## BIC

Another metric is the *Bayesian information criterion (BIC)*

$$BIC = \frac{1}{n\hat{\sigma}^2}(RSS + \log(n)d\hat{\sigma}^2),$$

where again the model with the minimum BIC value is selected

In comparison to  $C_p$  or AIC, where the penalty is  $2d\hat{\sigma}^2$ , the BIC penalty is  $\log(n)d\hat{\sigma}^2$ . This means that generally BIC has a heavier penalty (because  $\log(n) > 2$  for  $n > 7$ ), thus BIC selects models with fewer variables than  $C_p$  or AIC.

In general, all three criteria are based on rigorous theoretical asymptotic ( $n \rightarrow \infty$ ) justifications.

## Adjusted R-squared value

Another simple method which is not backed up by statistical theory, but that often works well in practice, is to simply adjust the  $R^2$  metric by taking into account the number of predictors as we defined before.

The adjusted  $R^2$  value for a model with  $d$  variables is calculated as follows

$$\text{Adjusted } R^2 = 1 - \frac{RSS/(n - d - 1)}{TSS/(n - 1)}.$$

In this case we choose the model with the maximum adjusted  $R^2$  value.

### Example

By plotting the various model selection criteria against the number of predictors (Figure 4.3), we can see that, in our example,  $C_p$ , AIC and BIC are in agreement (2 predictors). Adjusted  $R^2$  would select 3 predictors.

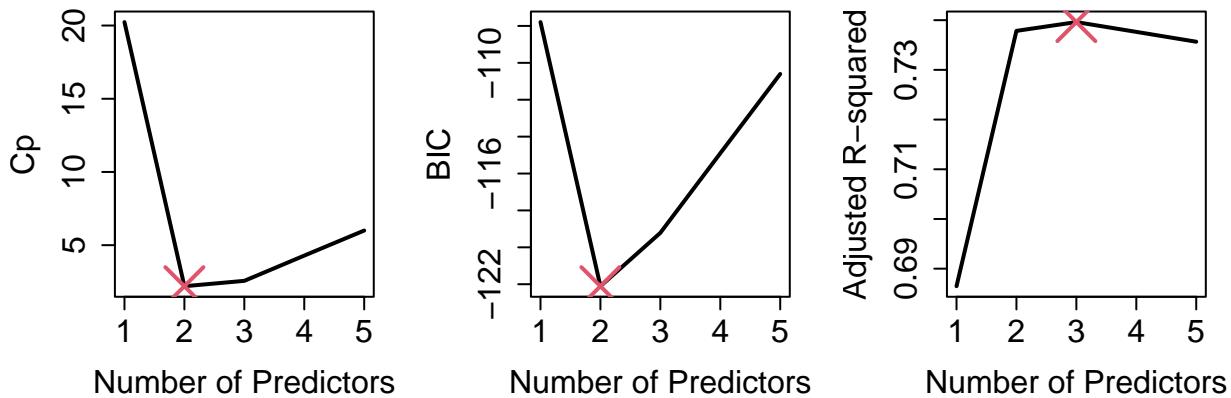


Figure 4.3: Various model selection criteria against number of predictors.

### Further points

- A shortcoming of the previous approaches is that they are not applicable for models that have more variables than the size of the sample ( $d > n$ ).
- Also, in the setting of penalised regression (which we will see later on in the course) deciding what  $d$  actually becomes a bit problematic.
- CV is an effective computational tool which can be used in all settings.



# Chapter 5

## Model Search Methods

In this section, we consider some methods for selecting subsets of predictors.

### 5.1 Best Subset Selection

To perform *best subset selection*, we fit a separate least squares regression for each possible combination of the  $p$  predictors. This is often broken up into stages, as follows:

1. Let  $M_0$  denote the *null model* which contains no predictors. This model simply predicts the *sample mean* of the response for each observation.
2. For  $k = 1, 2, \dots, p$ :
  - Fit all  $\binom{p}{k}$  models<sup>1</sup> that contain exactly  $k$  predictors.
  - Pick the best among these  $\binom{p}{k}$  models and call it  $M_k$ . Here *best* is defined as having the smallest RSS or largest  $R^2$ .
3. Select a single best model from among  $M_0, M_1, \dots, M_p$  using cross-validated prediction error,  $C_p$  (AIC), BIC, or adjusted  $R^2$ .

It is important to note that use of  $RSS$  or  $R^2$  in step 2 of the above algorithm is acceptable because the models all have an equal number of predictors. We can't use  $RSS$  in step 3 because, as already discussed,  $RSS$  decreases monotonically as the number of predictors included in the model increases.

Cross-validation could also be used in step 2, but would be computationally more time-consuming as it would ultimately involve fitting all  $2^p$  possible models  $K$  times, each time excluding the  $i$ th fold when training and using the  $i$ th fold to calculate a prediction error. By only using cross-validation at step 3, we only need to fit  $p + 1$  models  $K$  times.

**An Illustration:** Suppose we have three predictors  $X_1$ ,  $X_2$  and  $X_3$ . In this case best subset selection works as follows:

---

<sup>1</sup>Reminder:  $\binom{p}{k} = \frac{p!}{k!(p-k)!}$  is the binomial coefficient. It gives all the possible ways to extract a subset of  $k$  elements from a fixed set of  $p$  elements. For instance, if  $p = 3$  and  $k = 2$ , the number of models with 2 predictors is given by  $\binom{3}{2} = \frac{3!}{2!(3-2)!} = \frac{1 \cdot 2 \cdot 3}{1 \cdot 2 \cdot 1} = 3$ .

$M_0$  : intercept only (null model)

$$C_1 : \begin{cases} X_1 \\ X_2 \rightarrow \text{lowest training } RSS \text{ within } C_1 : \text{make this } M_1 \\ X_3 \end{cases}$$

$$C_2 : \begin{cases} X_1, X_2 \rightarrow \text{lowest training } RSS \text{ within } C_2 : \text{make this } M_2 \\ X_1, X_3 \\ X_2, X_3 \end{cases}$$

$M_3 : X_1, X_2, X_3$  (full model)

We then choose one model among  $M_0, M_1, M_2, M_3$  either via validation/cross-validation, or based on  $C_p$ , BIC, adjusted  $R^2$ .

## 5.2 Forwards Stepwise Selection

For computational reasons, best subset selection cannot be applied with very large  $p$  as it needs to consider all  $2^p$  models. This is true even without using cross-validation which involves fitting the coefficients of at least some of the models  $K$  times. Best subset selection may also suffer from statistical problems when  $p$  is large. The larger the search space, the higher the chance of finding models which look good on (overfit) the training data, even though they might not have any predictive power on future data.

*Forward stepwise selection* is a computationally efficient alternative to best subset selection. This is because it considers a much smaller set of models. Forward stepwise selection works by starting with the model containing no predictors, and then adding one predictor at a time until all of the predictors are in the model. In particular, at each step the variable that gives the greatest additional improvement to the fit is added to the model. More formally, the algorithm proceeds as follows:

1. Let  $M_0$  denote the null model which contains no predictors.
2. For  $k = 0, 1, \dots, p - 1$ :
  - Consider all  $p - k$  models that augment the predictors in  $M_k$  with one additional predictor.
  - Choose the best among these  $p - k$  models and call it  $M_{k+1}$ . Here *best* is defined as having the smallest RSS or largest  $R^2$ .
3. Select a single best model from among  $M_0, M_1, \dots, M_p$  using cross-validated prediction error,  $C_p$  (AIC), BIC, or adjusted  $R^2$ .

**An Illustration:** Again, consider three predictors  $X_1, X_2$  and  $X_3$ . Forward stepwise selection works as follows:

$M_0$  : intercept only (null model)

$C_1 : \begin{cases} X_1 \\ X_2 \rightarrow \text{lowest training } RSS \text{ within } C_1 : \text{ make this } M_1 \\ X_3 \end{cases}$

$C_2 : \begin{cases} X_1, X_2 \rightarrow \text{lowest training } RSS \text{ within } C_2 : \text{ make this } M_2 \\ X_2, X_3 \end{cases}$

$M_3 : X_1, X_2, X_3$  (full model)

At the end we choose again one model among  $M_0, M_1, M_2, M_3$  based on validation/cross-validation, or based on  $C_p$ , BIC, adjusted  $R^2$ .

### Some general comments:

- Best subset selection requires training  $2^p$  models, forward stepwise selection requires only  $1 + \frac{p(p+1)}{2}$  comparisons.
- So for  $p = 20$ : best subset selection considers 1,048,576 models, whereas forward stepwise selection considers 211 models!
- There is cost to pay though for the gain in scalability. Forward stepwise selection can be *sub-optimal* in the sense that the models selected are not the best subset models.
- This is because the guided forward stepwise search at each step depends on the previously selected predictor.
- As an example, in the illustration above we had  $p = 3$ . The best possible one-variable model contained  $X_2$ . It is quite possible that the best two-variable model contains the pair  $(X_1, X_3)$ , however, we did not consider this combination in the above algorithm.
- However, in practice best subset selection and forward stepwise selection perform quite similarly in terms of *predictive accuracy*.

## 5.3 Backwards Stepwise Selection

*Backwards stepwise selection* begins with the full model containing all  $p$  predictors, and iteratively removes the least useful predictor one-at-a-time.

1. Let  $M_p$  denote the *full model* which contains all predictors.
2. For  $k = p, p-1, \dots, 1$ :
  - Consider all  $k$  models that contain all but one of the predictors in  $M_k$ , for a total of  $k-1$  predictors.
  - Choose the best among these  $k$  models and call it  $M_{k-1}$ . Here *best* is defined as having the smallest RSS or largest  $R^2$ .
3. Select a single best model from among  $M_0, M_1, \dots, M_p$  using cross-validated prediction error,  $C_p$  (AIC), BIC, or adjusted  $R^2$ .

Just like the forward algorithm, backward elimination requires only  $1 + \frac{p(p+1)}{2}$  comparisons.

**An Illustration:** Again, consider three predictors  $X_1$ ,  $X_2$  and  $X_3$ . Backwards stepwise selection works as follows:

$$M_3 : X_1, X_2, X_3 \text{ (full model)}$$

$$C_2 : \begin{cases} X_1, X_2 \rightarrow \text{lowest training } RSS \text{ within } C_2 : \text{ make this } M_2 \\ X_1, X_3 \\ X_2, X_3 \end{cases}$$

$$C_1 : \begin{cases} X_1 \\ X_2 \rightarrow \text{lowest training } RSS \text{ within } C_1 : \text{ make this } M_1 \end{cases}$$

$$M_0 : \text{intercept only (null model)}$$

At the end we choose again one model among  $M_0, M_1, M_2, M_3$  based on cross-validation, or based on  $C_p$ , BIC, adjusted  $R^2$ .

### 5.3.1 Concluding Remarks

- Best subset, forward stepwise and backward stepwise selection approaches give similar but not identical results.
- Hybrid approaches that combine forward and backward steps also exist (but we will not cover these).
- Best subset and forward selection can be applied to some extent when  $n < p$ , but only up to the model that contains  $n - 1$  predictors. Backward selection cannot be applied when  $n < p$ .
- The following practical demonstration and the practical classes will involve applying these methods to real data sets.

## 5.4 Practical Demonstration

In this practical demonstration we will start with an analysis of the **Hitters** dataset included in package **ISLR**. This dataset consists of 322 records of baseball players. The response variable is the players' salary and the number of predictors is 19, including variables such as number of hits, years in the league and so forth.

### 5.4.1 Data handling

The first thing to do is to load the data by first installing package **ISLR** in R by typing `install.packages('ISLR')` (if you have already installed the package ignore this). Next we load the library and use `?Hitters` which will open the help window in RStudio with information about the dataset.

```
library(ISLR)
?Hitters
```

Incidentally, recall that we can view the help file for any function or dataset in R by typing `? followed by the function or dataset, as above for dataset Hitters or below for function head.`. You will need to make use of this to increase your understanding of R - I will not be able to explain every detail of every function throughout these tutorials (although I will explain a lot), and you indeed will wish to vary the commands I call to do your own investigations!

To get a clearer picture of how the data looks we can use commands `names()`, `head()` and `dim()`. The first returns the names of the 20 variables, the second returns the first 6 rows (type `?head` to see how to adjust this) and the third gives us the number of rows (sample size) and number of columns (1 response + 19 predictors).

```
names(Hitters)
```

```
## [1] "AtBat"      "Hits"       "HmRun"      "Runs"       "RBI"        "Walks"
## [7] "Years"       "CATBat"     "CHits"      "CHmRun"     "CRuns"      "CRBI"
## [13] "CWalks"     "League"     "Division"    "PutOuts"    "Assists"    "Errors"
## [19] "Salary"      "NewLeague"
```

```
head(Hitters)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun
## -Andy Allanson	293	66	1	30	29	14	1	293	66	1
## -Alan Ashby	315	81	7	24	38	39	14	3449	835	69
## -Alvin Davis	479	130	18	66	72	76	3	1624	457	63
## -Andre Dawson	496	141	20	65	78	37	11	5628	1575	225
## -Andres Galarraga	321	87	10	39	42	30	2	396	101	12
## -Alfredo Griffin	594	169	4	74	51	35	11	4408	1133	19
	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors		
## -Andy Allanson	30	29	14	A	E	446	33	20		
## -Alan Ashby	321	414	375	N	W	632	43	10		
## -Alvin Davis	224	266	263	A	W	880	82	14		
## -Andre Dawson	828	838	354	N	E	200	11	3		
## -Andres Galarraga	48	46	33	N	E	805	40	4		
## -Alfredo Griffin	501	336	194	A	W	282	421	25		
	Salary	NewLeague								
## -Andy Allanson	NA		A							
## -Alan Ashby	475.0		N							
## -Alvin Davis	480.0		A							
## -Andre Dawson	500.0		N							
## -Andres Galarraga	91.5		N							
## -Alfredo Griffin	750.0		A							

```
dim(Hitters)
```

```
## [1] 322 20
```

Prior to proceeding to any analysis we must check whether the data contains any missing values. To do this we can use a combination of the commands `sum()` and `is.na()`. We find that we have 59 missing `Salary` entries. To remove these 59 rows we make use of the command `na.omit()` and then double-check.

```

sum( is.na( Hitters ) ) # checks for missing data in the entire dataframe.

## [1] 59

sum( is.na( Hitters$Salary ) ) # checks for missing data in the response only.

## [1] 59

Hitters = na.omit( Hitters ) # removes entries to the dataframe with any data
                             # missing.

dim( Hitters )

## [1] 263 20

sum( is.na( Hitters ) ) # check that there is now no

## [1] 0

# missing data in the dataframe.

```

### 5.4.2 Best Subset Selection

Now we are ready to start with best subset selection. There are different packages in R which perform best subset as well as stepwise selections. We will use the function `regsubsets()` from library `leaps` (again `install.packages('leaps')` is needed if not already installed). `regsubsets()` essentially performs step 2 of the algorithm presented in Section 5.1, although yields results that make step 3 relatively straightforward. Note that the `.` part of `Salary~.` indicates that we wish to consider all possible predictors in the subset selection method. We will store the results from best subset in an object called `best` (any name would do) and summarize the results via `summary()`, which outputs the best models of different dimensionality.

```

library(leaps)
best = regsubsets(Salary~., Hitters)
summary(best)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., Hitters)
## 19 Variables (and intercept)
##          Forced in Forced out
## AtBat      FALSE      FALSE
## Hits       FALSE      FALSE
## HmRun      FALSE      FALSE
## Runs       FALSE      FALSE
## RBI        FALSE      FALSE
## Walks      FALSE      FALSE
## Years      FALSE      FALSE
## CAtBat     FALSE      FALSE
## CHits      FALSE      FALSE
## CHmRun     FALSE      FALSE
## CRuns      FALSE      FALSE

```

```

## CRBI      FALSE   FALSE
## CWalks    FALSE   FALSE
## LeagueN   FALSE   FALSE
## DivisionW FALSE   FALSE
## PutOuts   FALSE   FALSE
## Assists   FALSE   FALSE
## Errors    FALSE   FALSE
## NewLeagueN FALSE  FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 ) " "   " "   " "   " "   " "   " "   " "   " "   " "   " "   " "   "*" 
## 2  ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "   "*" 
## 3  ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "   "*" 
## 4  ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "   "*" 
## 5  ( 1 ) "*"  "*"  " "   " "   " "   " "   " "   " "   " "   " "   " "   "*" 
## 6  ( 1 ) "*"  "*"  " "   " "   " "   " "   "*"  " "   " "   " "   " "   "*" 
## 7  ( 1 ) " "   "*"  " "   " "   " "   " "   "*"  " "   "*"  " "   " "   " " 
## 8  ( 1 ) "*"  "*"  " "   " "   " "   " "   "*"  " "   " "   "*"  " "   "*"  " "
##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) " "   " "   " "   " "   " "   " "   " "
## 2  ( 1 ) " "   " "   " "   " "   " "   " "   " "
## 3  ( 1 ) " "   " "   " "   "*"  " "   " "   " "
## 4  ( 1 ) " "   " "   " "   "*"  " "   " "   " "
## 5  ( 1 ) " "   " "   " "   "*"  " "   " "   " "
## 6  ( 1 ) " "   " "   " "   "*"  " "   " "   " "
## 7  ( 1 ) " "   " "   " "   "*"  " "   " "   " "
## 8  ( 1 ) "*"  " "   " "   "*"  " "   " "   " "

```

The asterisks \* indicate variables which are included in model  $M_k$ ,  $k = 1, \dots, 19$  as discussed in the subset selection algorithm of Section 5.1. For instance, we see that for the model with one predictor that variable is CRBI, while for the model with four predictors the selected variables are Hits, CRBI, DivisionW and PutOuts. As we see the `summary()` command displayed output up to the best model with 8 predictors; this is the default option in `regsubsets()`. To change this we can use the extra argument `nvmax`. Below we re-run the command for all best models (19 in total). Now we also store the output of `summary()` in an object called `results` and further investigate what this contains via `names()`. As we can see there is a lot of useful information! Of particular use to us, we can see that it returns  $R^2$ , RSS, adjusted- $R^2$ ,  $C_p$  and BIC. We can easily extract these quantities; for instance below we see the  $R^2$  values for the 19 models.

```

best = regsubsets(Salary~., data = Hitters, nvmax = 19)
results = summary(best)
names(results)

## [1] "which"   "rsq"     "rss"     "adjr2"   "cp"      "bic"     "outmat"  "obj"

```

```
results$rsq
```

```
## [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
## [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
## [15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159
```

Lets store these quantities as separate objects. We can also stack them together into one matrix as done below and see the values.

```
RSS = results$rsq
```

```
r2 = results$rsq
```

```
Cp = results$cp
```

```
BIC = results$bic
```

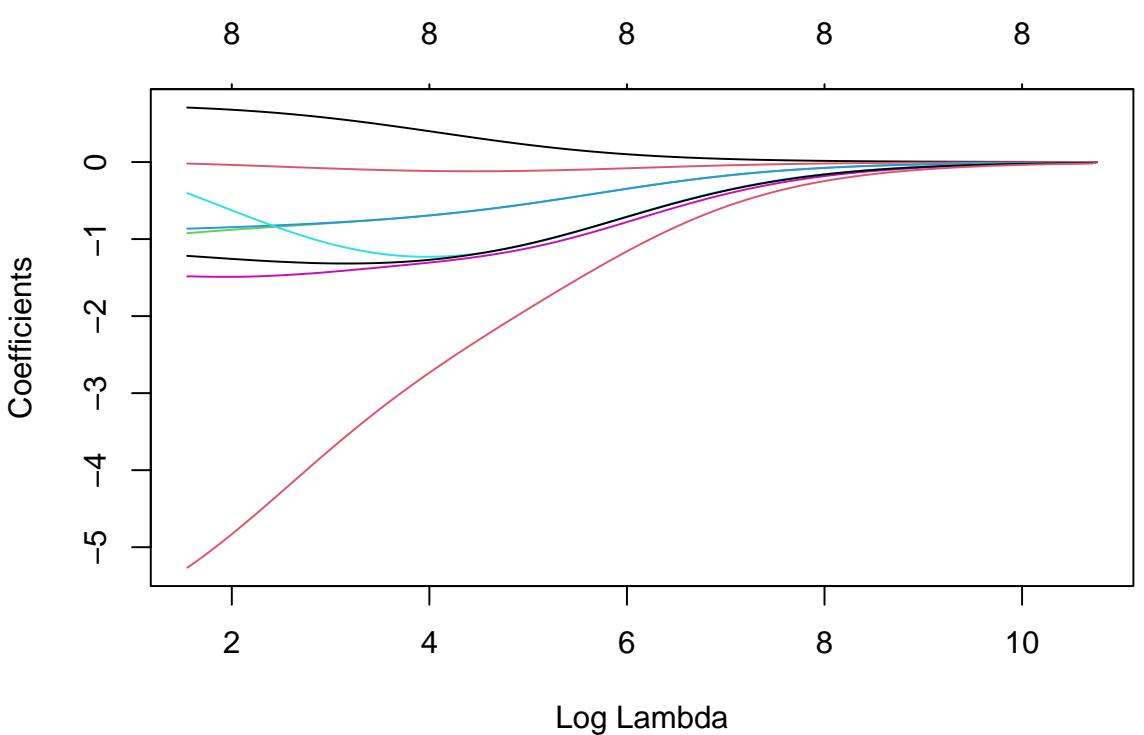
```
Adj_r2 = results$adjr2
```

```
cbind(RSS, r2, Cp, BIC, Adj_r2)
```

	RSS	r2	Cp	BIC	Adj_r2
## [1,]	36179679	0.3214501	104.281319	-90.84637	0.3188503
## [2,]	30646560	0.4252237	50.723090	-128.92622	0.4208024
## [3,]	29249297	0.4514294	38.693127	-135.62693	0.4450753
## [4,]	27970852	0.4754067	27.856220	-141.80892	0.4672734
## [5,]	27149899	0.4908036	21.613011	-144.07143	0.4808971
## [6,]	26194904	0.5087146	14.023870	-147.91690	0.4972001
## [7,]	25906548	0.5141227	13.128474	-145.25594	0.5007849
## [8,]	25136930	0.5285569	7.400719	-147.61525	0.5137083
## [9,]	24814051	0.5346124	6.158685	-145.44316	0.5180572
## [10,]	24500402	0.5404950	5.009317	-143.21651	0.5222606
## [11,]	24387345	0.5426153	5.874113	-138.86077	0.5225706
## [12,]	24333232	0.5436302	7.330766	-133.87283	0.5217245
## [13,]	24289148	0.5444570	8.888112	-128.77759	0.5206736
## [14,]	24248660	0.5452164	10.481576	-123.64420	0.5195431
## [15,]	24235177	0.5454692	12.346193	-118.21832	0.5178661
## [16,]	24219377	0.5457656	14.187546	-112.81768	0.5162219
## [17,]	24209447	0.5459518	16.087831	-107.35339	0.5144464
## [18,]	24201837	0.5460945	18.011425	-101.86391	0.5126097
## [19,]	24200700	0.5461159	20.000000	-96.30412	0.5106270

We know that RSS should steadily decrease and that  $R^2$  increase as we add predictors. Lets check that by plotting the values of RSS and  $R^2$ . We would like to have the two plots next to each other, so will make use of the command `par(mfrow())`, with the `c(1, 2)` indicating we require the plotting window to contain 1 row and 2 columns, before calling the command `plot()`.

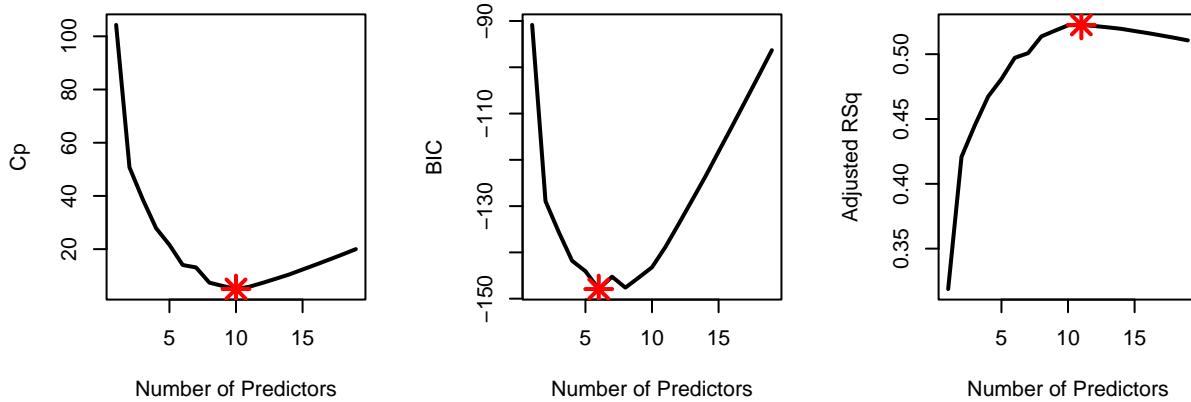
```
par(mfrow = c(1, 2))
plot(RSS, xlab = "Number of Predictors", ylab = "RSS",
     type = "l", lwd = 2)
plot(r2, xlab = "Number of Predictors", ylab = "R-square",
     type = "l", lwd = 2)
```



Results are as expected. Above the argument `type = 'l'` is used for the lines to appear (otherwise by default R would simply plot the points) and the argument `lwd` controls the thickness of the lines. Plots in R are very customizable (type `?par` to see all available options).

Now let us find how many predictors are included in the optimal models under  $C_p$ , BIC and adjusted- $R^2$  and produce some plots illustrating this information.

```
which.min(Cp)
## [1] 10
which.min(BIC)
## [1] 6
which.max(Adj_r2)
## [1] 11
par(mfrow = c(1, 3))
plot(Cp, xlab = "Number of Predictors", ylab = "Cp",
     type = 'l', lwd = 2)
points(10, Cp[10], col = "red", cex = 2, pch = 8, lwd = 2)
plot(BIC, xlab = "Number of Predictors", ylab = "BIC",
     type = 'l', lwd = 2)
points(6, BIC[6], col = "red", cex = 2, pch = 8, lwd = 2)
plot(Adj_r2, xlab = "Number of Predictors", ylab = "Adjusted RSq",
     type = "l", lwd = 2)
points(11, Adj_r2[11], col = "red", cex = 2, pch = 8, lwd = 2)
```

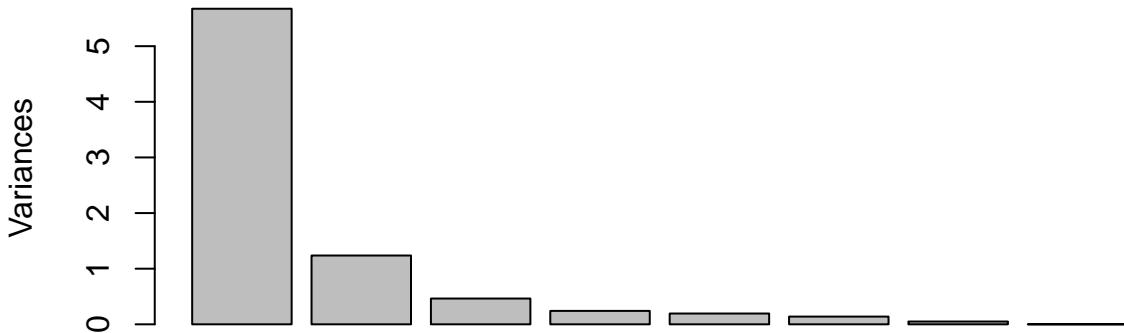


$C_p$  and adjusted- $R^2$  select 10 and 11 predictors respectively, while the optimal BIC model has only 6 predictors. In the code above we use the command `points()` to highlight the optimal model; the first two arguments of this command correspond to x-axis and y-axis coordinates. So, for example, we know that the optimal  $C_p$  model is the one with 10 predictors; therefore the first argument is set to 10 and the second argument is set to `Cp[10]` (i.e.,  $C_p$  at its minimum).

The `regsubsets()` in-built function for plotting the results is very convenient when the output from `summary()` is difficult to read due to there being many possible predictors. Below we see the visualisation for the BIC models. *Remark:* Because previously we used `par(mfrow(1,3))` we have to either close the plotting panel by typing `dev.off()` (this returns the window plot to the default state), or respecifying the window splitting using `par( mfrow = c(1,1) )`.

```
plot(best, scale = "bic")
```

**seatpos.pr**



The top row corresponds to the best model, while the bottom row to the worst model according to the chosen criterion. White squares correspond to variables that are excluded under each model.

Finally, we extract the model coefficients for any model we choose via the command `coef`. For instance, the best models under each approach are the following.

```
coef(best, 10) # Cp
```

##	(Intercept)	AtBat	Hits	Walks	CAtBat	CRuns
##	162.5354420	-2.1686501	6.9180175	5.7732246	-0.1300798	1.4082490

```

##          CRBI         CWalks      DivisionW       PutOuts       Assists
## 0.7743122 -0.8308264 -112.3800575  0.2973726  0.2831680
coef(best,6) # BIC

## (Intercept)      AtBat       Hits       Walks       CRBI      DivisionW
## 91.5117981   -1.8685892  7.6043976  3.6976468  0.6430169 -122.9515338
## PutOuts
## 0.2643076

coef(best,11) # adj-Rsq

## (Intercept)      AtBat       Hits       Walks      CAtBat      CRuns
## 135.7512195  -2.1277482  6.9236994  5.6202755 -0.1389914  1.4553310
## CRBI        CWalks      LeagueN      DivisionW      PutOuts      Assists
## 0.7852528  -0.8228559 43.1116152 -111.1460252  0.2894087  0.2688277

```

### 5.4.3 Forward Stepwise Selection

We can use the same function to also perform forward or backward selection by only changing one of its arguments. Forward selection can be implemented in the following way.

```

fwd = regsubsets(Salary~., data = Hitters, nvmax = 19,
                  method = "forward")
summary(fwd)

```

```

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
## 19 Variables (and intercept)
##          Forced in Forced out
## AtBat      FALSE      FALSE
## Hits       FALSE      FALSE
## HmRun      FALSE      FALSE
## Runs       FALSE      FALSE
## RBI        FALSE      FALSE
## Walks      FALSE      FALSE
## Years      FALSE      FALSE
## CAtBat     FALSE      FALSE
## CHits      FALSE      FALSE
## CHmRun     FALSE      FALSE
## CRuns      FALSE      FALSE
## CRBI       FALSE      FALSE
## CWalks     FALSE      FALSE
## LeagueN    FALSE      FALSE
## DivisionW  FALSE      FALSE
## PutOuts    FALSE      FALSE
## Assists    FALSE      FALSE
## Errors     FALSE      FALSE
## NewLeagueN FALSE      FALSE

```

```

## 1 subsets of each size up to 19
## Selection Algorithm: forward
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1 ( 1 ) " " " " " " " " " " " " " " " " " " " "
## 2 ( 1 ) " " "*" " " " " " " " " " " " " " " " "
## 3 ( 1 ) " " "*" " " " " " " " " " " " " " " " "
## 4 ( 1 ) " " "*" " " " " " " " " " " " " " " " "
## 5 ( 1 ) "*" "*" " " " " " " " " " " " " " " " "
## 6 ( 1 ) "*" "*" " " " " " " " " " " * " " " " "
## 7 ( 1 ) "*" "*" " " " " " " " " " " * " " " " "
## 8 ( 1 ) "*" "*" " " " " " " " " " " * " " " " "
## 9 ( 1 ) "*" "*" " " " " " " " " " " * " " " " "
## 10 ( 1 ) "*" "*" " " " " " " " " " " * " " " " "
## 11 ( 1 ) "*" "*" " " " " " " " " " " * " " " " "
## 12 ( 1 ) "*" "*" " " " " * " " " " " * " " " " "
## 13 ( 1 ) "*" "*" " " " " * " " " " " * " " " " "
## 14 ( 1 ) "*" "*" " * " " * " " " " " * " " " " "
## 15 ( 1 ) "*" "*" " * " " * " " " " " * " " " " "
## 16 ( 1 ) "*" "*" " * " " * " " * " " " * " " " "
## 17 ( 1 ) "*" "*" " * " " * " " * " " " * " " " "
## 18 ( 1 ) "*" "*" " * " " * " " * " " * " " " * "
## 19 ( 1 ) "*" "*" " * " " * " " * " " * " " * " " "
##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1 ( 1 ) " " " " " " " " " " " " "
## 2 ( 1 ) " " " " " " " " " " " " "
## 3 ( 1 ) " " " " " " * " " " " " " "
## 4 ( 1 ) " " " " " * " " " * " " " " "
## 5 ( 1 ) " " " " " * " " " * " " " " "
## 6 ( 1 ) " " " " " * " " " * " " " " "
## 7 ( 1 ) "*" " " " " * " " " * " " " " "
## 8 ( 1 ) "*" " " " " * " " " * " " " " "
## 9 ( 1 ) "*" " " " " * " " " * " " " " "
## 10 ( 1 ) "*" " " " " * " " " * " " " " "
## 11 ( 1 ) "*" " " " * " " " * " " " " "
## 12 ( 1 ) "*" " " " * " " " * " " " " "
## 13 ( 1 ) "*" " " " * " " " * " " " * " " "
## 14 ( 1 ) "*" " " " * " " " * " " " * " " "
## 15 ( 1 ) "*" " " " * " " " * " " " * " " "
## 16 ( 1 ) "*" " " " * " " " * " " " * " " "
## 17 ( 1 ) "*" " " " * " " " * " " " * " " "
## 18 ( 1 ) "*" " " " * " " " * " " " * " " "
## 19 ( 1 ) "*" " " " * " " " * " " " * " " "

```

The analysis will then be essentially identical to the previous analysis for best subset selection.

One interesting thing to check is whether the results from best subset and forward stepwise selection are in agreement (we know that this will not necessarily be the case). For example, let's have a look at the models with 6 and 7 predictors.

```

coef(best, 6)

## (Intercept)      AtBat      Hits      Walks      CRBI      DivisionW
## 91.5117981    -1.8685892   7.6043976   3.6976468   0.6430169  -122.9515338
## PutOuts
## 0.2643076

coef(fwd, 6)

## (Intercept)      AtBat      Hits      Walks      CRBI      DivisionW
## 91.5117981    -1.8685892   7.6043976   3.6976468   0.6430169  -122.9515338
## PutOuts
## 0.2643076

coef(best, 7)

## (Intercept)      Hits      Walks      CAtBat     CHits      CHmRun
## 79.4509472    1.2833513   3.2274264  -0.3752350  1.4957073   1.4420538
## DivisionW      PutOuts
## -129.9866432   0.2366813

coef(fwd, 7)

## (Intercept)      AtBat      Hits      Walks      CRBI      CWalks
## 109.7873062   -1.9588851   7.4498772   4.9131401   0.8537622  -0.3053070
## DivisionW      PutOuts
## -127.1223928   0.2533404

```

The results for the model with 6 predictors are in agreement (which is good to know because this would be one of the three models one would likely select), but as we see the results for the models with 7 predictors are different.

#### 5.4.4 Best Subset Selection: Validation

In this section, we consider applying best subset selection not based on model selection criteria but using the validation approach.

As we know, for validation (and cross-validation) we need to be able to predict from our linear models. Unfortunately, `regsubsets()` doesn't have its own built-in function for that (most similar types of R functions have). You are welcome to try writing your own code for this step, but it is beyond the scope of the course. For the purposes of performing subset selection by validation in this course, we will just make use of the function `predict.regsubsets` below which will do the predictions for us. The important aspects of the function are the arguments. `object` should be the result of a call to the function `regsubsets()` (for example the object `best` in Section 5.4.2 or `fwd` in Section 5.4.3), `newdata` should be a data frame of data at which we wish to predict at, and `id` indicates that we wish to use the resulting model from `regsubsets` with `id` number of predictors (so in this case could range from 1,2,...19). The output is the response prediction for each combination of inputs. Note that if you wish to use this function in your own code you will need to define it by copy and pasting this code segment.

```
predict.regsubsets = function(object, newdata, id, ...){
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id = id)
  xvars = names(coefi)
  mat[, xvars] %*% coefi
}
```

The validation approach is based on splitting the data once into a training sample and a validation or testing sample. We also need to decide how to split the data. A common approach is to use 2/3 of the sample for training and 1/3 of the sample for testing (although it does not have to be so).

Note the first line: `set.seed`. If we removed this command, then every time we ran the following code we could get different results (for example, variables selected) due to the effect of the random `sample` (do feel free to try this, or alternatively changing the seed number). `set.seed` ensures a particular random state for any particular integer value, thus making the code reproducible (i.e. the same every time).

```
set.seed(10)          # for the results to be reproducible
dim(Hitters)

## [1] 263 20

training.obs = sample(1:263, 175)
Hitters.train = Hitters[training.obs, ]
Hitters.test = Hitters[-training.obs, ]
dim(Hitters.train)

## [1] 175 20

dim(Hitters.test)

## [1] 88 20
```

Validation is then executed as follows. We first apply `regsubsets()` to the training set. By doing so note how we are only using the training data for both steps 2 and 3 of the algorithm in Section 5.1.

```
best.val = regsubsets(Salary~, data = Hitters.train, nvmax = 19)
```

Then we create an empty vector called `val.error` where we will store the 19 (because we have 19 models) validation MSEs. We then use a `for` loop inside which we (i) generate predictions using `predict.regsubsets()` and (ii) calculate the validation MSEs. At the end we just locate which model produces the lowest MSE. The code is the following.

```
val.error<-c()
for(i in 1:19){
  pred = predict.regsubsets(best.val, Hitters.test, i)
  val.error[i] = mean((Hitters.test$Salary - pred)^2)
}
```

```

val.error

## [1] 199324.8 184215.9 196949.2 192709.1 180806.0 179388.2 178929.9 176754.4
## [9] 177029.2 163013.2 165134.5 164986.7 166072.1 164983.9 164890.7 165144.4
## [17] 164983.7 164962.6 164959.9

which.min(val.error)

## [1] 10

```

In this instance we see that the model with 10 predictors is selected as the best. This was also the model selected by  $C_p$  in Section 5.4.2.

### 5.4.5 What about inference?

Note that the aim of subset selection is to find which predictors should be included within our model, not to identify the specific model (including coefficient values). Therefore, it is important to note that there is a final step to fitting a model by validation, and that is to train the *same* 10-predictor model to the entire data. This will yield different coefficient values to the current coefficients. The current coefficients should not be used because these were “learned” from the reduced training sample (that is, we didn’t use all of the available data).

For the purpose of fitting a specific model (with specified predictors) we use the command `lm()` in R.

```
coef(best.val, 10) # Check which variables to use in the lm.
```

```

## (Intercept)          AtBat          Hits          Runs          Walks         CAtBat
## -21.6323918     -0.7961450      4.5874937     -3.1211583      7.5830718     -0.1058408
## CRuns            CRBI           CWalks        DivisionW        PutOuts
## 1.9895304      0.3010303     -1.1736748    -121.3385526      0.1989660

ls10 = lm(Salary ~ AtBat + Hits + Runs + Walks + CAtBat + CRuns + CRBI
          + CWalks + Division + PutOuts, data = Hitters)

```

As we have seen the output from `regsubsets()` relates to the model selection procedure; i.e. we get the values of  $C_p$ , BIC, adjusted- $R^2$ . We can also extract model specific regression coefficients via the command `coef()`. However, the `lm` object above is required for quantities such as standard errors, confidence and prediction intervals etc., for example, recalling that `summary()` provides a summary, and `confint()` provides confidence intervals. In addition, we can now use this model for prediction purposes using the built-in `predict` function for linear models (which is distinct to the `predict.regsubsets` function we defined above).

```

summary(ls10)

##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Runs + Walks + CAtBat +
##     CRuns + CRBI + CWalks + Division + PutOuts, data = Hitters)
## 
```

```

## Residuals:
##      Min     1Q Median     3Q    Max
## -927.60 -177.70 -30.03 142.91 1986.30
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 144.30288   66.52508   2.169 0.031007 *
## AtBat       -1.84992   0.52681  -3.512 0.000528 ***
## Hits        7.64658   1.82552   4.189 0.0000388 ***
## Runs        -2.92542   2.33580  -1.252 0.211576
## Walks        6.28202   1.68895   3.719 0.000246 ***
## CAtBat      -0.12749   0.05751  -2.217 0.027517 *
## CRuns        1.45072   0.41373   3.506 0.000537 ***
## CRBI         0.66823   0.20074   3.329 0.001002 **
## CWalks      -0.80479   0.26367  -3.052 0.002515 **
## DivisionW   -117.14522  39.33350  -2.978 0.003182 **
## PutOuts      0.27111   0.07406   3.661 0.000307 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 312.8 on 252 degrees of freedom
## Multiple R-squared:  0.5375, Adjusted R-squared:  0.5191
## F-statistic: 29.29 on 10 and 252 DF,  p-value: < 0.00000000000000022
confint(ls10, 1:11)

##                               2.5 %      97.5 %
## (Intercept) 13.2868988 275.31885341
## AtBat       -2.8874302 -0.81240354
## Hits        4.0513571 11.24181209
## Runs        -7.5256017  1.67476709
## Walks        2.9557696  9.60827296
## CAtBat      -0.2407504 -0.01423684
## CRuns        0.6359135  2.26553592
## CRBI         0.2728907  1.06357743
## CWalks      -1.3240670 -0.28550387
## DivisionW  -194.6094923 -39.68094337
## PutOuts      0.1252518   0.41696960

test.data <- data.frame( "AtBat" = 322, "Hits" = 90, "HmRun" = 14,
                         "Runs" = 40, "RBI" = 22, "Walks" = 40,
                         "Years" = 4, "CAtBat" = 3000, "CHits" = 830,
                         "CHMRun" = 100, "CRuns" = 250, "CRBI" = 600,
                         "CWalks" = 300, "League" = "N", "Division" = "W",
                         "PutOuts" = 600, "Assists" = 60, "Errors" = 7,
                         "newLeague" = "N" )
predict( ls10, newdata = test.data )

```

```
##      1
## 556.3127
```

### 5.4.6 Variance of selection based on validation

When using validation and cross-validation for best subset selection (or any stepwise method) it is important to understand that results depend on the random splitting between training and testing samples. This is generally a drawback which is not discussed a lot, especially for the simple validation approach. The below code implements the validation approach presented previously but now starting from 50 *different random seeds*, storing the number of predictors in the selected model each time.

```
min.valid = c()

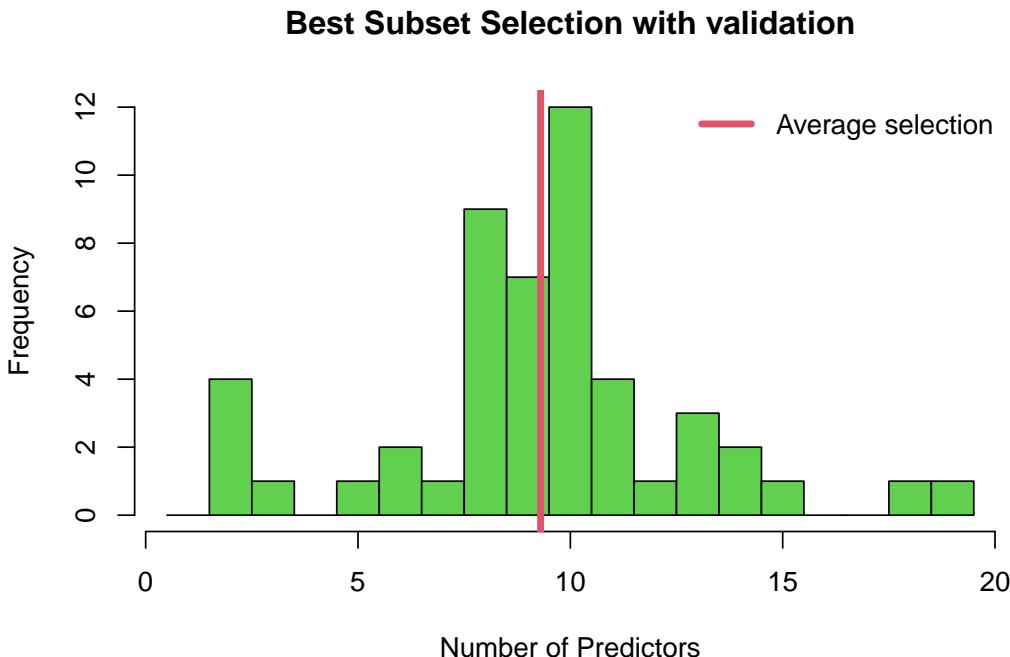
for(n in 1:50){
  set.seed(n)
  training.obs = sample(1:263, 175)
  Hitters.train = Hitters[training.obs, ]
  Hitters.test = Hitters[-training.obs, ]

  best.val = regsubsets(Salary~., data = Hitters.train, nvmax = 19)

  val.error<-c()
  for(i in 1:19){
    pred = predict.regsubsets(best.val, Hitters.test, i)
    val.error[i] = mean((Hitters.test$Salary - pred)^2)
  }
  val.error
  min.valid[n] = which.min(val.error)
}
```

The histogram below shows that the final selection of the best model can vary a lot, even in terms of the number of predictors! Note that in addition, the histogram doesn't tell the whole story because each time, say, a 10-predictor model is chosen, it may be choosing different predictors. In this case we can see that the validation method is most often choosing a model with 8, 9 or 10 predictors.

```
hist(min.valid, col = 3, breaks = seq( from = 0.5, to = 19.5, length = 20 ),
      xlab = 'Number of Predictors',
      main = 'Best Subset Selection with validation')
abline(v = mean(min.valid), col = 2, lwd = 4)
legend('topright', legend=c('Average selection'), bty = 'n', lty = 1,
      lwd = 4, col = 2)
```



### 5.4.7 Cross-validation

Now lets turn our attention to the  $K$ -fold cross-validation (CV) approach. This procedure is generally preferable to simple validation because it utilizes the entire sample for training as well as for testing.

In contrast to the application of validation in Section 5.4.4, where we used the training data for both steps 2 and 3 of the algorithm presented in Section 5.1, we here use the full dataset in the initial `regsubsets()` command (step 2), and cross-validation for step 3 only. Therefore step 2 proceeds as before:

```
best = regsubsets(Salary~., data = Hitters, nvmax = 19)
```

Since step 3 involves fitting models with specified predictors, we need to once again use the `lm()` function in R. Since this would involve manually specifying 19 models using `lm()`, we will instead demonstrate how to obtain a  $K$ -fold cross-validation error for just 3 of the models suggested by `regsubsets()` only.

In particular, we are going to answer the following question...which of the different models chosen under  $C_p$ , BIC and adjusted  $R^2$  respectively in Section 5.4.2 should we choose? Recall  $C_p$  yielded the model with 10 variables, BIC the one with 6, and adjusted  $R^2$  the one with 11. Therefore, we will calculate the  $K$ -fold cross-validation error of the models with 6, 10 and 11 variables suggested by `regsubsets()` only. Let's remind ourselves of the variables in those three models:

```
coef(best, 10) # Cp
```

	(Intercept)	AtBat	Hits	Walks	CAtBat	CRuns
##	162.5354420	-2.1686501	6.9180175	5.7732246	-0.1300798	1.4082490
##	CRBI	CWalks	DivisionW	PutOuts	Assists	
##	0.7743122	-0.8308264	-112.3800575	0.2973726	0.2831680	

```
coef(best,6) # BIC  
  
## (Intercept) AtBat Hits Walks CRBI DivisionW  
## 91.5117981 -1.8685892 7.6043976 3.6976468 0.6430169 -122.9515338  
## PutOuts  
## 0.2643076  
  
coef(best,11) # adj-Rsq  
  
## (Intercept) AtBat Hits Walks CAtBat CRUNS  
## 135.7512195 -2.1277482 6.9236994 5.6202755 -0.1389914 1.4553310  
## CRBI CWALKS LeagueN DivisionW PutOuts Assists  
## 0.7852528 -0.8228559 43.1116152 -111.1460252 0.2894087 0.2688277
```

Linear models based on the *whole training set* would be constructed as follows:

```
ls10 = lm(Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
          Division + PutOuts + Assists, data = Hitters)
ls6 = lm(Salary ~ AtBat + Hits + Walks + CRBI + Division + PutOuts,
         data = Hitters)
ls11 = lm(Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
          + League + Division + PutOuts + Assists, data = Hitters)
```

Now we turn to calculating  $K$ -fold cross-validation errors for these models. A first question is how many folds to use. Common choices in practice are 5 and 10, leading to *5-fold* and *10-fold* CV. In this example we will use 10 folds. Of course, often the sample size  $n$  will not be perfectly divisible by  $K$  (as in our case where  $n=263$  and  $K=10$ ). This does not matter, since it is not a problem if some folds have fewer observations.

Lets see how we can create folds in R. First we need to form an *indicator vector* with length equal to 263 containing the numbers 1,2,3,4,5,6,7,8,9,10 roughly the same number of times. This can be done by combining the commands `cut()` and `seq()` in R as below. The command `table()` then tabulates how many of each integer are present in the resulting vector.

```
table(folds)

## folds
##  1  2  3  4  5  6  7  8  9 10
## 27 26 26 26 27 26 26 26 26 27
```

It is then very important to *randomly re-shuffle* this vector, because we do not want the fold creation procedure to be *deterministic*. We can do this very easily in R using once again the command `sample()`.

```
set.seed(2)
folds = sample(folds)
folds

## [1] 8 10 8 7 3 5 3 6 9 2 3 4 5 9 7 2 10 5 7 6 2 10 8 6 4
## [26] 1 5 8 7 5 10 4 5 10 8 8 1 5 9 2 2 10 8 10 1 1 7 9 2 6
## [51] 5 8 3 4 8 2 1 6 8 10 5 7 2 3 7 7 1 1 7 3 9 8 4 3 6
## [76] 7 5 5 10 2 5 6 2 1 4 5 9 1 3 3 8 3 10 8 6 1 2 6 1 4
## [101] 9 6 4 10 9 1 9 7 9 8 4 8 6 1 10 2 10 10 7 9 3 9 4 4 5
## [126] 6 10 3 9 2 8 8 6 2 2 6 1 9 1 10 7 4 2 4 8 5 8 1 6 3
## [151] 1 10 10 1 3 3 3 7 3 4 2 9 4 6 8 9 7 2 1 7 4 3 5 7 8
## [176] 4 9 7 9 5 2 1 1 6 4 4 5 10 6 5 1 10 9 7 3 1 5 2 7 7
## [201] 4 6 7 10 3 6 5 10 4 9 9 5 7 7 2 6 9 5 5 3 2 3 5 5 9
## [226] 3 6 1 1 9 8 6 8 9 10 4 4 10 10 5 2 3 3 6 3 4 10 9 6 2
## [251] 1 8 6 7 8 4 2 8 1 10 2 7 4
```

Finally, we need to create a *matrix* this time with  $K=10$  rows and 3 columns. The validation MSEs of the folds for model `ls10` will be stored in the first column of this matrix, the validation MSEs of the folds for model `ls6` will be stored in the second column of this matrix, and the validation MSEs of the folds for model `ls11` will be stored in the final column.

```
cv.errors = matrix(NA, nrow = k, ncol = 3,
                   dimnames = list(NULL, c("ls10", "ls6", "ls11")))
cv.errors

##          ls10  ls6  ls11
## [1,]     NA   NA   NA
## [2,]     NA   NA   NA
## [3,]     NA   NA   NA
## [4,]     NA   NA   NA
## [5,]     NA   NA   NA
## [6,]     NA   NA   NA
## [7,]     NA   NA   NA
## [8,]     NA   NA   NA
## [9,]     NA   NA   NA
## [10,]    NA   NA   NA
```

Above `NA` means “not available”. The `NA` entries will be replaced with the MSEs as the loop below progresses. The part `dimnames = list(NULL, c("ls10", "ls6", "ls11"))` just assigns no names to the rows and names the columns `"ls10"`, `"ls6"` and `"ls11"` respectively.

Now for calculating the cross-validation errors we will need a *for* loop to loop over the folds. The important part is to understand what `data = Hitters[folds!=i, ]` is doing. In R `!=` means “not equal”. So for example when `i=1` we discard the rows of `Hitter` for which the corresponding `fold` vector equals 1. We then predict only for the rows for which the `fold` vector equals `i` (in `Hitters[folds==i, ]` and `Hitters$Salary[folds==i]`). We then store each validation MSE in the appropriate row and column of `cv.errors`.

```

for(i in 1:k){
  ls10 = lm(Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
             Division + PutOuts + Assists, data = Hitters[folds!=i, ] )
  ls6 = lm(Salary ~ AtBat + Hits + Walks + CRBI + Division + PutOuts,
            data = Hitters[folds!=i, ])
  ls11 = lm(Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
             + League + Division + PutOuts + Assists,
            data = Hitters[folds!=i, ])

  pred10 <- predict( ls10, newdata = Hitters[folds==i, ] )
  pred6 <- predict( ls6, newdata = Hitters[folds==i, ] )
  pred11 <- predict( ls11, newdata = Hitters[folds==i, ] )

  cv.errors[i,] = c( mean( (Hitters$Salary[folds==i]-pred10)^2),
                    mean( (Hitters$Salary[folds==i]-pred6)^2),
                    mean( (Hitters$Salary[folds==i]-pred11)^2) )
}

cv.errors

##          ls10        ls6        ls11
## [1,] 132495.25 127446.51 130581.86
## [2,] 111992.83 110608.25 109426.81
## [3,] 87822.97  98247.02  87132.03
## [4,] 113161.24 98812.43  112384.13
## [5,] 225490.54 231835.72 227007.73
## [6,] 83622.26  46935.72  85460.28
## [7,] 90430.09  95275.05  91963.45
## [8,] 100105.45 106254.47  98519.80
## [9,] 97701.55  97551.93  104063.92
## [10,] 48005.26  63602.55  46157.69

```

Finally, we can take the average of the MSEs in each column of the matrix to obtain the  $K$ -fold cross-validation error for each of the proposed models.

```
cv.mean.errors <- colMeans(cv.errors)
```

```
cv.mean.errors
```

```

##          ls10        ls6        ls11
## 109082.7 107657.0 109269.8

```

We may choose `ls6` as it has slightly smaller average cross-validation error, although in this case there is not that much in it. Since we also tend to favour models with fewer predictors

as they tend to be more interpretable, we would probably choose `lss6`.

# Chapter 6

## Shrinkage Methods

### 6.1 Motivation

#### Issue due to multicollinearity

Recall that multicollinearity refers to a situation when two or more predictor variables in the linear regression model are highly (linearly) correlated. The least square estimates will become “unstable”. Here we explore the issue via the following simulation study.

This is a simulation study because we simulate data based on a known model, add some random noise, and then investigate how different methods use this data to estimate the regression parameters. We can then evaluate the performance of the method based on how close the estimated parameter values are to the true values.

We assume a correct model of

$$y = x_1 - x_2 + \epsilon$$

so that we have true generating  $\beta_0 = 0$ ,  $\beta_1 = 1$  and  $\beta_2 = -1$ , and noise term  $\epsilon \sim \mathcal{N}(0, 1)$ . In other words, for any specified  $(x_1, x_2)$ , we evaluate the response  $x_1 - x_2$  before adding random noise.

Rather than sampling the training data independently, however, we assume a high level of collinearity between  $x_1$  and  $x_2$  by simulating  $x_1 \sim \mathcal{N}(0, 1)$  and  $x_2 \sim \mathcal{N}(0.9x_1, 0.2^2)$ . In other words,  $x_2$  is a scaling of  $x_1$  with just a small amount of random noise added - there is definitely a strong correlation between the values of those predictors!

Feel free to look through the following code to understand how the simulation study is working and the plots below are generated! Note that  $(X^T X)^{-1} X^T Y$  is the regression coefficient estimates for LS.

```
# Number of iterations for the simulation study.  
iter <- 50  
  
# Number of training points.  
n <- 10
```

```

# Empty matrix of NAs for the regression coefficients for LS
b_LS <- matrix( NA, nrow = 2, ncol = iter )

# For each iteration...
for( i in 1:iter ){

  # As we can see x2 is highly correlated with x1 - COLLINEARITY!
  x1 <- rnorm( n )
  x2 <- 0.9 * x1 + rnorm( n, 0, 0.2 )

  # Design matrix.
  x <- cbind( x1, x2 )

  # y = x1 - x2 + e, where error e is N(0,1)
  y <- x1 - x2 + rnorm( n )

  # LS regression components - the below calculation obtains them given the
  # training data and response generated in this iteration, and stores
  # those estimates in the relevant column of the matrix b_LS.
  b_LS[,i]<- solve( t(x) %*% x ) %*% t(x) %*% y

}

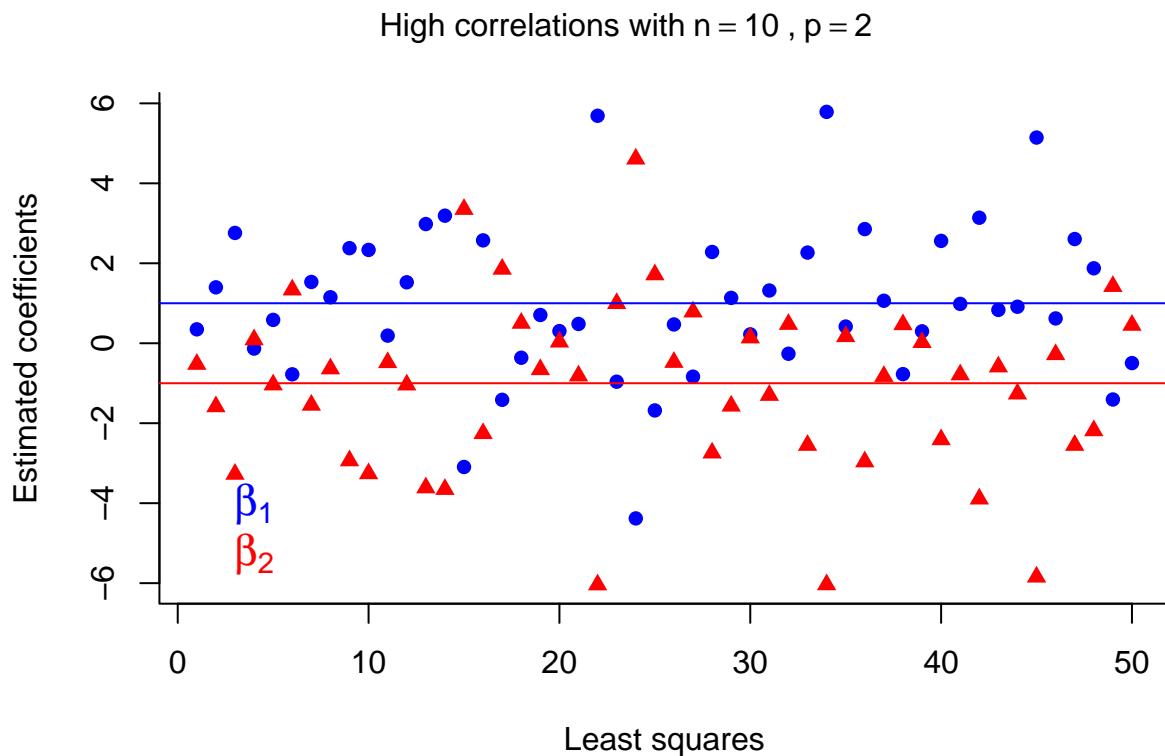
# Now we plot the results.

# Plot the LS estimates for beta_1.
plot( b_LS[1,], col = 'blue', pch = 16, ylim = c( min(b_LS), max(b_LS) ),
      ylab = 'Estimated coefficients', xlab = 'Least squares', bty ='l' )
# Add the LS estimates for beta_2 in a different colour.
points( b_LS[2,], col = 'red', pch = 17 )
# Add horizontal lines representing the true values of beta_1 and beta_2.
abline( a = 1, b = 0, col = 'blue' ) # Plot the true value of beta_1 = 1.
abline( a = -1, b = 0, col='red' ) # Plot the true value of beta_2 = -1.

# Add a legend to this plot.
legend( 'bottomleft', cex = 1.3, text.col=c('blue','red'),
        legend = c( expression( beta[1] ),expression( beta[2] ) ),
        bty = 'n' )

# Add a title over the top of both plots.
mtext( expression( 'High correlations with'~n == 10~, ' ~ p == 2 ),
       side = 3, line = -3, outer = TRUE )

```



### Issue due to $p$ is close to $n$

Another issue is when  $p$  is close to  $n$ . We investigate this case by running the simulation above again, but this time sampling both  $x_1$  and  $x_2$  from a standard normal distribution  $\mathcal{N}(0, 1)$  (making the two predictors independent) and lowering the number of training points to 3. I have removed the detailed comments this time - try to see (and understand!) how it is different to the simulation above.

```
iter <- 50
n <- 3

b_LS <- matrix( NA, nrow = 2, ncol = iter )

for( i in 1:iter ){
  x1 <- rnorm( n )
  x2 <- rnorm( n )
  x <- cbind( x1, x2 )
  y <- x1 - x2 + rnorm( n )
  b_LS[,i] <- solve( t(x) %*% x ) %*% t(x) %*% y
}

plot( b_LS[1,], col = 'blue', pch = 16, ylim = c( min(b_LS), max(b_LS) ),
      ylab = 'Estimated coefficients', xlab = 'Least squares', bty = 'l' )
```

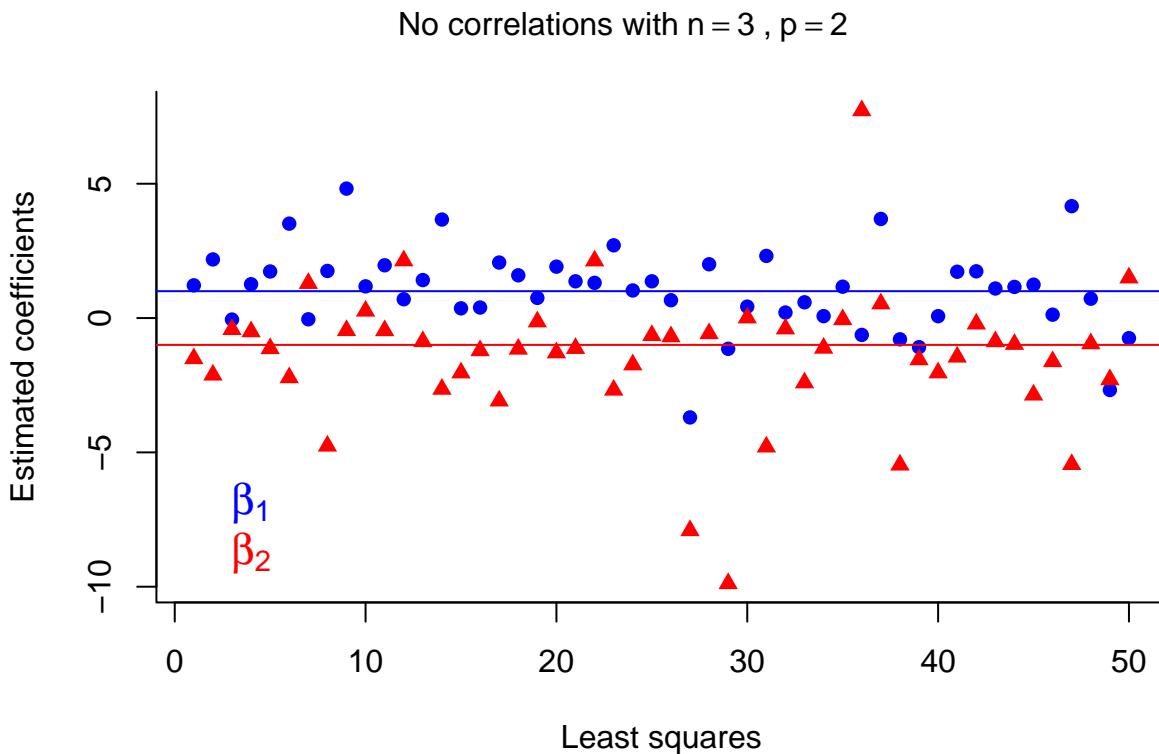
```

points( b_LS[2,], col = 'red', pch = 17 )
abline( a = 1, b = 0, col = 'blue' )
abline( a = -1, b = 0, col='red' )

# Add a legend to this plot.
legend( 'bottomleft', cex = 1.3, text.col=c('blue','red'),
        legend = c( expression( beta[1] ),expression( beta[2] ) ), bty = 'n' )

mtext( expression('No correlations with'~n == 3~, ' ~ p == 2),
       side = 3, line = -3, outer = TRUE )

```



LS estimates suffer again from high variance. Of course, having only  $n = 3$  training runs isn't ideal in any situation, but it illustrates how LS estimates are affected when  $p$  is close to  $n$ .

You may want to run the simulation above again but increase the number of training points and the number of predictors.

Many modern machine learning applications suffer from both of these problems we have just explored - that is,  $p$  can be large (and of similar magnitude to  $n$ ) and it is very likely that there will be strong correlations among the many predictors!

To address these issues, one may adopt shrinkage methods. In previous lectures, we have seen *discrete* model search methods, where one option is to pick the *best* model based on information criteria (e.g.  $C_p$ , BIC, etc.). In this case, we generally seek the minimum of

$$\frac{\text{model fit}}{\text{RSS}} + \text{penalty on model dimensionality}$$

across models of differing dimensionality (number of predictors).

Shrinkage methods offer a *continuous analogue* (which is much faster). In this case we train only the full model, but the *estimated regression coefficients* minimise a general solution of the form

$$\underbrace{\text{model fit}}_{\text{RSS}} + \text{penalty on size of coefficients.}$$

This approach is called *penalised* or *regularised* regression.

## 6.2 Ridge Regression

Recall that Least Squares (LS) produces estimates of  $\beta_0, \beta_1, \beta_2, \dots, \beta_p$  by minimising

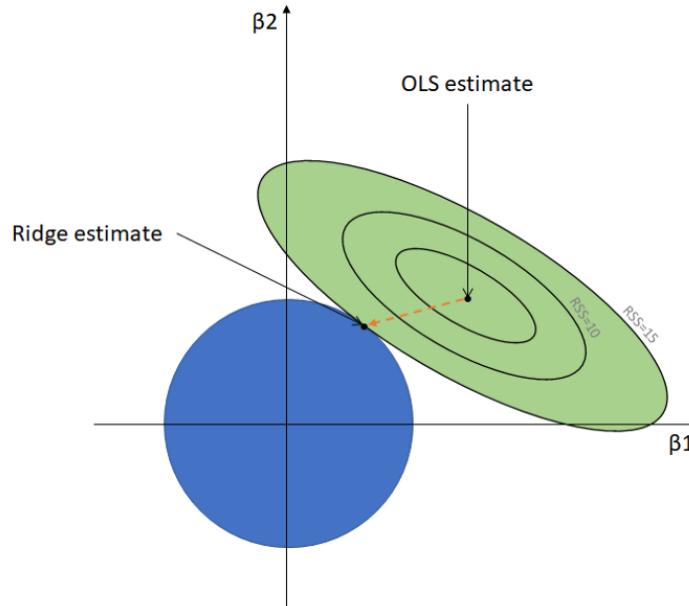
$$RSS = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

On the other hand, *ridge regression* minimises the cost function

$$\underbrace{\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2}_{\text{model fit}} + \underbrace{\lambda \sum_{j=1}^p \beta_j^2}_{\text{penalty}} = RSS + \lambda \sum_{j=1}^p \beta_j^2,$$

where  $\lambda \geq 0$  is a *tuning parameter*. The role of this parameter is crucial; it controls the *trade-off* between model fit and the size of the coefficients.

In fact, the estimates based on the ridge regression cost function is the solution to the least squares problem subject to the constraint  $\sum_{j=1}^p \beta_j^2 \leq c$  (Hint: Lagrange Multiplier). The following figure is the geometric interpretation of the ridge regression.



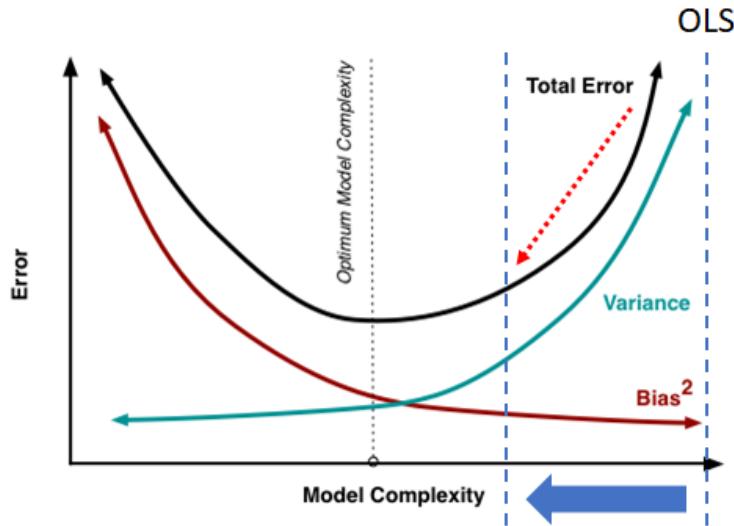
The green contours represent the RSS in terms of different values of parameters of a linear regression model. Each contour is a connection of spots where the RSS is the same, centered with the (Ordinary) Least Squares estimate where the RSS is the lowest. Note that the LS estimate is the point where it best fits the training set (low-bias).

The blue circle represents the constraint  $\sum_{j=1}^p \beta_j^2 \leq c$ . Unlike the LS estimates, the ridge estimates change as the size of the blue circle changes. It is simply where the circle meets the most outer contour. How ridge regression works is how we tune the size of the circle, which is via tuning  $\lambda$ .

From the ridge regression cost function, observe that:

- when  $\lambda = 0$  the penalty term has no effect and the ridge solution is the same as the LS solution.
- as  $\lambda$  gets *larger* naturally the penalty term gets larger; so in order to minimise the *entire* function (model fit + penalty) the regression coefficients will necessarily get *smaller*!
- Unlike LS, ridge regression does not produce one set of coefficients, it produces different sets of coefficients for different values of  $\lambda$ !
- As  $\lambda$  is increasing the coefficients are *shrunk* towards 0 (for really large  $\lambda$  the coefficients are almost 0).

Tuning  $\lambda$  is in fact balancing the bias and variance.



Another way to view the ridge regression is via vector representation. That is the ridge estimates  $\hat{\beta}_R = (\mathbf{X}^T \mathbf{X} + I)^{-1} \mathbf{X}^T \mathbf{Y}$  shifts the  $\mathbf{X}^T \mathbf{X}$  in the LS estimates  $\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$  by  $\lambda I$ . Note in the cases of “perfect” multicollinearity and  $n < p$ ,  $\mathbf{X}^T \mathbf{X}$  is singular (non-invertible) and shifting  $\mathbf{X}^T \mathbf{X}$  by  $\lambda I$  leads to an invertible matrix  $(\mathbf{X}^T \mathbf{X} + I)$ .

## Demonstration in multicollinearity issue

The following code conducts the same simulation study as in the motivation and adds ridge estimates for comparison. Note that  $(X^T X + \lambda I)^{-1} X^T Y$  are the regression coefficient estimates for ridge regression.

```

# Number of iterations for the simulation study.
iter <- 50

# Number of training points.
n <- 10

# lambda coefficient - feel free to explore this parameter!
lambda <- 0.3

# Empty matrix of NAs for the regression coefficients, one for LS
# and one for ridge.
b_LS <- matrix( NA, nrow = 2, ncol = iter )
b_ridge <- matrix( NA, nrow = 2, ncol = iter )

# For each iteration...
for( i in 1:iter ){

  # As we can see x2 is highly correlated with x1 - COLLINEARITY!
  x1 <- rnorm( n )
  x2 <- 0.9 * x1 + rnorm( n, 0, 0.2 )

  # Design matrix.
  x <- cbind( x1, x2 )

  # y = x1 - x2 + e, where error e is N(0,1)
  y <- x1 - x2 + rnorm( n )

  # LS regression components - the below calculation obtains them given the
  # training data and response generated in this iteration, and stores
  # those estimates in the relevant column of the matrix b_LS.
  b_LS[,i]<- solve( t(x) %*% x ) %*% t(x) %*% y

  # Ridge regression components - notice that the only difference relative
  # to the LS estimates above is the addition of a lambda x identity matrix
  # inside the inverse operator.
  b_ridge[,i]<- solve( t(x) %*% x + lambda * diag(2) ) %*% t(x) %*% y

}

# Now we plot the results. First, split the plotting window
# such that it has 1 row and 2 columns.
par(mfrow=c(1,2))

# Plot the LS estimates for beta_1.
plot( b_LS[1,], col = 'blue', pch = 16, ylim = c( min(b_LS), max(b_LS) ),
      ylab = 'Estimated coefficients', xlab = 'Least squares', bty = 'l' )

```

```

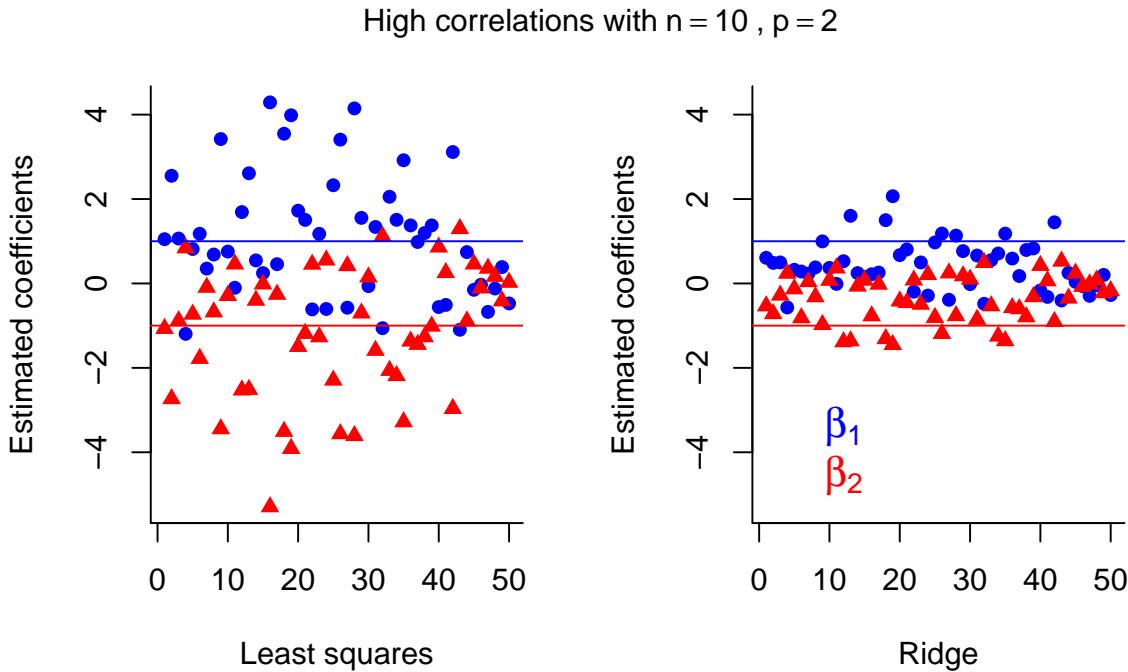
# Add the LS estimates for beta_2 in a different colour.
points( b_LS[2,], col = 'red', pch = 17 )
# Add horizontal lines representing the true values of beta_1 and beta_2.
abline( a = 1, b = 0, col = 'blue' ) # Plot the true value of beta_1 = 1.
abline( a = -1, b = 0, col='red' ) # Plot the true value of beta_2 = -1.

# Do the same for the ridge estimates.
plot( b_ridge[1,], col = 'blue', pch = 16, ylim = c(min(b_LS), max(b_LS)),
      ylab = 'Estimated coefficients', xlab = 'Ridge', bty = 'l' )
points( b_ridge[2,], col = 'red', pch = 17 )
abline( a = 1, b = 0, col = 'blue' )
abline( a = -1, b = 0, col = 'red' )

# Add a legend to this plot.
legend( 'bottomleft', cex = 1.3, text.col=c('blue','red'),
        legend = c( expression( beta[1] ),expression( beta[2] ) ),
        bty = 'n' )

# Add a title over the top of both plots.
mtext( expression( 'High correlations with'~n == 10~, ' ~ p == 2 ),
       side = 3, line = -3, outer = TRUE )

```



We can see that ridge regression introduces some bias but decreases the variance, which could result in better predictions.

### Demonstration in $p$ is close to $n$ issue

Similarly, the following code conducts the same simulation study as in the motivation for  $p$  is close to  $n$  issue and adds ridge estimates for comparison.

```

iter <- 50
n <- 3
lambda <- 0.3

b_LS <- matrix( NA, nrow = 2, ncol = iter )
b_ridge <- matrix( NA, nrow = 2, ncol = iter )

for( i in 1:iter ){
  x1 <- rnorm( n )
  x2 <- rnorm( n )
  x <- cbind( x1, x2 )
  y <- x1 - x2 + rnorm( n )
  b_LS[,i]<- solve( t(x) %*% x ) %*% t(x) %*% y
  b_ridge[,i]<- solve( t(x) %*% x + lambda * diag(2) ) %*% t(x) %*% y
}

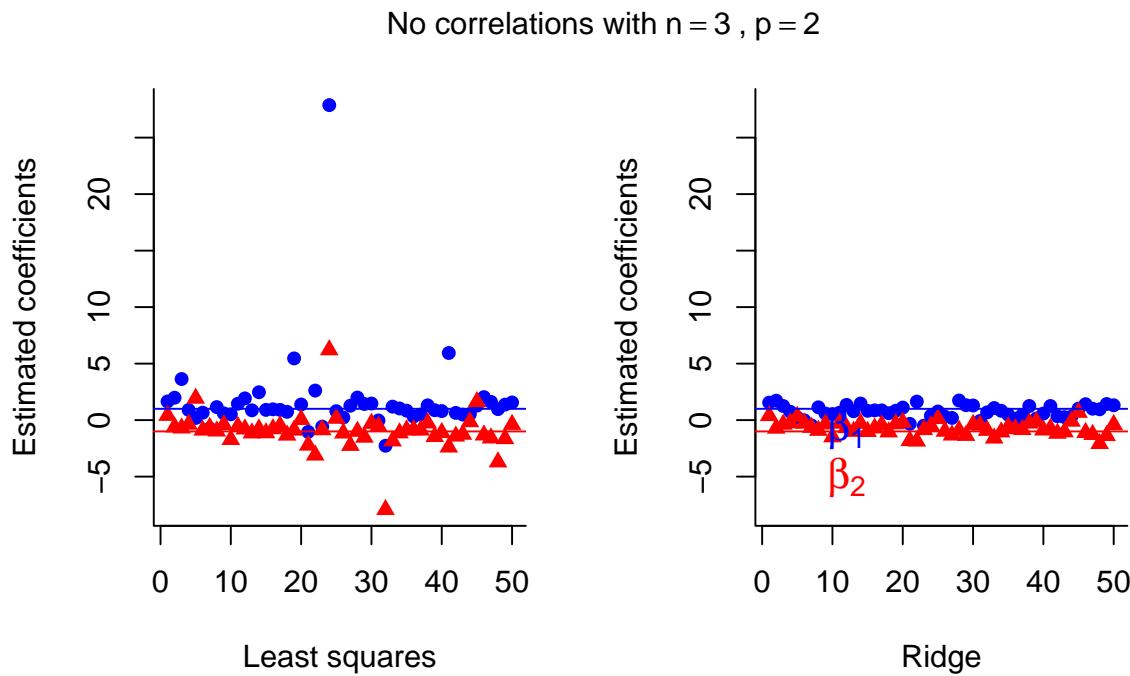
par(mfrow=c(1,2))
plot( b_LS[1,], col = 'blue', pch = 16, ylim = c( min(b_LS), max(b_LS) ),
      ylab = 'Estimated coefficients', xlab = 'Least squares', bty ='l' )
points( b_LS[2,], col = 'red', pch = 17 )
abline( a = 1, b = 0, col = 'blue' )
abline( a = -1, b = 0, col='red' )

plot( b_ridge[1,], col = 'blue', pch = 16, ylim = c(min(b_LS), max(b_LS)),
      ylab = 'Estimated coefficients', xlab = 'Ridge', bty ='l' )
points( b_ridge[2,], col = 'red', pch = 17 )
abline( a = 1, b = 0, col = 'blue' )
abline( a = -1, b = 0, col = 'red' )

# Add a legend to this plot.
legend( 'bottomleft', cex = 1.3, text.col=c('blue','red'),
        legend = c( expression( beta[1] ),expression( beta[2] ) ), bty = 'n' )

mtext( expression('No correlations with'~n == 3~, ' ~ p == 2),
       side = 3, line = -3, outer = TRUE )

```



Ridge estimates are again more stable and can be an improvement over LS.

## Scalability and Feature Selection

- Ridge regression is applicable for any number of predictors even when  $n < p$ , remember that LS solutions do not exist in this case. It is also much faster than the model-search based methods since we train only one model.
- Strictly speaking, ridge regression does not perform feature selection (since the coefficients are almost never exactly 0), but it can give us a good idea of which predictors are not influential and can be combined with post-hoc analysis based on ranking the absolute values of the coefficients.

## Scaling

- The LS estimates are scale invariant: multiplying a feature  $X_j$  by some constant  $c$  will scale down  $\hat{\beta}_j$  by  $1/c$ , so  $X_j \hat{\beta}_j$  will remain unaffected.
- In ridge regression (and any shrinkage method) the scaling of the features matters! If a relevant feature is in a *smaller scale* (that is, the numbers are smaller, e.g. if you use metres as opposed to millimetres) in relation to other features it will have a *larger coefficient* which constitutes a greater proportion of the penalty term and will thus *shrink proportionally faster to zero*.
- So, we want the features to be on the *same scale*.
- In practice, we *standardise*:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\sum \frac{1}{n}(x_{ij} - \bar{x}_j)^2}},$$

so that all predictors have variance equal to one.

## Practical Demonstration

We will continue analysing the `Hitters` dataset included in package `ISLR`, this time applying the method of ridge regression.

We perform the usual steps of removing missing data, as seen in Section 5.4.

```
library(ISLR)
Hitters = na.omit(Hitters)
dim(Hitters)
```

```
## [1] 263 20
```

```
sum(is.na(Hitters))
```

```
## [1] 0
```

To conduct ridge regression, the first thing to do is to load up the package `glmnet` (remember to run the command `install.packages('glmnet')` the first time).

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

Initially, we will make use of function `glmnet()` which implements ridge regression without cross-validation, but it does give a range of solutions over a grid of  $\lambda$  values. The grid is produced automatically - we do not need to specify anything. Of course, if we want to we can define the grid manually. Let's have a look first at the help document of `glmnet()`.

```
?glmnet
```

A first important thing to observe is that `glmnet()` requires a different syntax than `regsubsets()` for declaring the response variable and the predictors. Now the response should be a vector and the predictor variables must be stacked column-wise as a matrix. This is easy to do for the response. For the predictors we have to make use of the `model.matrix()` command. Let us define as `y` the response and as `x` the matrix of predictors.

```
y = Hitters$Salary
x = model.matrix(Salary~., Hitters)[,-1] # Here we exclude the first column
# because it corresponds to the
# intercept.
```

```
head(x)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun
## -Alan Ashby	315	81	7	24	38	39	14	3449	835	69
## -Alvin Davis	479	130	18	66	72	76	3	1624	457	63
## -Andre Dawson	496	141	20	65	78	37	11	5628	1575	225
## -Andres Galarraga	321	87	10	39	42	30	2	396	101	12
## -Alfredo Griffin	594	169	4	74	51	35	11	4408	1133	19
## -Al Newman	185	37	1	23	8	21	2	214	42	1
##	CRuns	CRBI	CWalks	LeagueN	DivisionW	PutOuts	Assists	Errors		

```

## -Alan Ashby      321  414   375     1      1    632    43    10
## -Alvin Davis    224  266   263     0      1    880    82    14
## -Andre Dawson    828  838   354     1      0    200    11     3
## -Andres Galarraga  48   46    33     1      0    805    40     4
## -Alfredo Griffin  501  336   194     0      1    282   421    25
## -Al Newman       30    9    24     1      0     76   127     7
##                               NewLeagueN
## -Alan Ashby          1
## -Alvin Davis         0
## -Andre Dawson        1
## -Andres Galarraga    1
## -Alfredo Griffin     0
## -Al Newman           0

```

See how the command `model.matrix()` automatically transformed the categorical variables in `Hitters` with names `League`, `Division` and `NewLeague` to dummy variables with names `LeagueN`, `DivisionW` and `NewLeagueN`. Other important things that we see in the help document are that `glmnet()` uses a grid of 100 values for  $\lambda$  (`nlambda = 100`) and that parameter `alpha` is used to select between ridge implementation and lasso implementation. The default option is `alpha = 1` which runs lasso (see Chapter 6.3) - for ridge we have to set `alpha = 0`. Finally we see the default option `standardize = TRUE`, which means we do not need to standardize the predictor variables manually.

Now we are ready to run our first ridge regression!

```
ridge = glmnet(x, y, alpha = 0)
```

Using the command `names()` we can find what is returned by the function. The most important components are "beta" and "lambda". We can easily extract each using the `$` symbol.

```
names(ridge)
```

```

## [1] "a0"        "beta"       "df"         "dim"        "lambda"     "dev.ratio"
## [7] "nulldev"   "npasses"    "jerr"       "offset"     "call"       "nobs"

ridge$lambda

## [1] 255282.09651 232603.53864 211939.68135 193111.54419 175956.04684
## [6] 160324.59659 146081.80131 133104.29670 121279.67783 110505.52551
## [11] 100688.51919 91743.62866 83593.37754 76167.17227 69400.69061
## [16] 63235.32454 57617.67259 52499.07736 47835.20402 43585.65633
## [21] 39713.62675 36185.57761 32970.95062 30041.90224 27373.06245
## [26] 24941.31502 22725.59734 20706.71790 18867.19015 17191.08098
## [31] 15663.87273 14272.33745 13004.42232 11849.14529 10796.49988
## [36] 9837.36858 8963.44387 8167.15622 7441.60857 6780.51658
## [41] 6178.15417 5629.30398 5129.21213 4673.54706 4258.36202
## [46] 3880.06088 3535.36696 3221.29471 2935.12376 2674.37546
## [51] 2436.79131 2220.31349 2023.06696 1843.34326 1679.58573
## [56] 1530.37596 1394.42158 1270.54501 1157.67329 1054.82879

```



```
## CRBI          0.0016548119  
## CWalks        0.0017488195  
## LeagueN       -0.0250669217  
## DivisionW    -0.3663713386  
## PutOuts       0.0010198028  
## Assists        0.0001663687  
## Errors         -0.0008021006  
## NewLeagueN   -0.0038293583
```

As we see above `glmnet()` defines the grid starting with large values of  $\lambda$ . This is because this helps speed up the optimisation procedure. Also, `glmnet()` performs some internal checks on the predictor variables in order to define the grid values. Therefore, it is generally better to let `glmnet()` define the grid automatically, rather than manually defining a grid. We further see that as expected `ridge$beta` had 19 rows (number of predictors) and 100 columns (length of the grid). Above the first 3 columns are displayed. Notice that `ridge$beta` does not return the estimates of the intercepts. We can use the command `coef()` for this, which returns all betas.

```
coef(ridge)[,1:3]
```

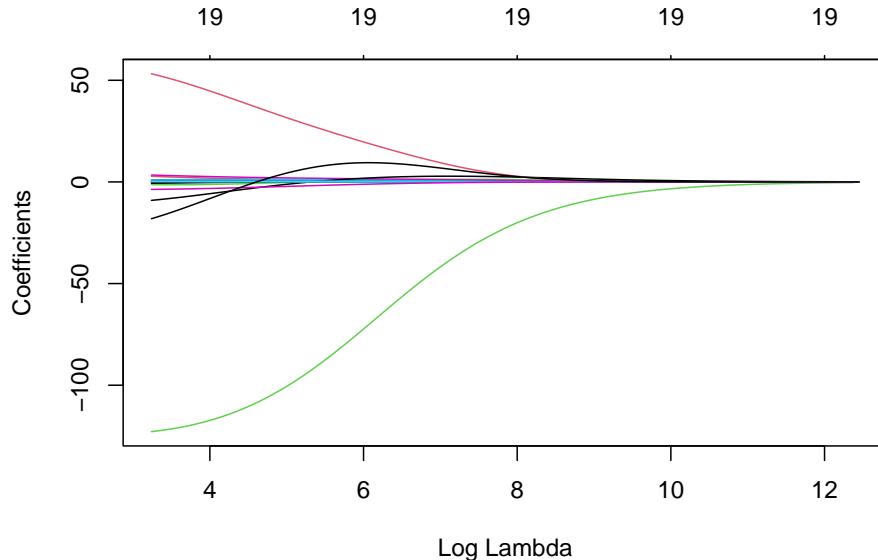
```

## RBI           0.0163950163
## Walks         0.0193237654
## Years         0.0787191800
## CAtBat        0.0002170325
## CHits          0.0007992257
## CHmRun         0.0060260057
## CRuns          0.0016034328
## CRBI           0.0016548119
## CWalks         0.0017488195
## LeagueN        -0.0250669217
## DivisionW      -0.3663713386
## PutOuts         0.0010198028
## Assists         0.0001663687
## Errors          -0.0008021006
## NewLeagueN     -0.0038293583

```

We can produce a plot similar to the ones seen previously via the following simple code. On the left we have the regularisation paths based on  $\log \lambda$ . Specifying `xvar` is important, as the default is to plot the  $\ell_1$ -norm on the  $x$ -axis (see Chapter 6.3). Don't worry about the row of 19's along the top of the plot for now, except to note that this is the number of predictor variables. The reason for their presence will become clear in Chapter 6.3.

```
plot(ridge, xvar = 'lambda')
```



## 6.3 Lasso Regression

### Disadvantage of Ridge Regression

- Unlike model search methods which select models that include subsets of predictors, ridge regression will include *all p* predictors.
- Even for those predictors that are irrelevant to the response, their coefficients will only get close to zero but never exactly zero!

- Even if we could perform a post-hoc analysis to reduce the number of predictors, ideally we would like a method that sets these coefficients equal to zero *automatically*.

## The Lasso

Similar to ridge regression it penalises the size of the coefficients, but instead of the *squares* it penalises the *absolute values*. The lasso regression estimates minimise the quantity

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = RSS + \lambda \sum_{j=1}^p |\beta_j|$$

The only difference is that the  $\beta_j^2$  term in the ridge regression penalty has been replaced by  $|\beta_j|$  in the lasso penalty.

Lasso offers the same benefits as ridge:

- it introduces some bias but decreases the variance, so it improves predictive performance.
- it is extremely scalable and can cope with  $n < p$  problems.

As with ridge regression, the lasso shrinks the coefficient estimates towards zero. However, in the case of the lasso, the  $\ell_1$  penalty has the effect of forcing some of the coefficient estimates to be *exactly equal to zero* when the tuning parameter  $\lambda$  is sufficiently large. Hence, much like best subset selection, the lasso performs *variable selection*. As a result, models generated from the lasso are generally much easier to interpret than those produced by ridge regression. We say that the lasso yields *sparse* models, that is, models that involve only a subset of the variables. As with ridge regression, the choice of  $\lambda$  is critical, but we defer discussion of this until Chapter 6.4.

## Ridge vs. Lasso

Here we make some comparisons between ridge regression and lasso regression.

### Ridge and Lasso with $\ell$ -norms

We have talked a little bit about the  $\ell$ -norms, but here we clarify the difference between ridge and lasso using them.

Lets just denote  $\beta = (\beta_1, \dots, \beta_p)$ . Then a more compact representation of ridge and lasso minimisation is the following:

- Ridge:  $\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \|\beta\|_2^2$ , where  $\|\beta\|_2 = \sqrt{\beta_1^2 + \dots + \beta_p^2}$  is the  $\ell_2$ -norm of  $\beta$ .
- Lasso:  $\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \|\beta\|_1$ , where  $\|\beta\|_1 = |\beta_1| + \dots + |\beta_p|$  is the  $\ell_1$ -norm of  $\beta$ .

That is why we say that ridge uses  $\ell_2$ -penalisation, while lasso uses  $\ell_1$ -penalisation.

### Ridge and Lasso as Constrained Optimisation Problems

- Ridge: minimise  $\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$  subject to  $\sum_{j=1}^p \beta_j^2 \leq c$ .
- Lasso: minimise  $\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$  subject to  $\sum_{j=1}^p |\beta_j| \leq c$ .
- The words *subject to* above essentially imposes the corresponding *constraints*.
- *Threshold*  $c$  may be considered as a *budget*: large  $c$  allows coefficients to be large.
- There is a *one-to-one* correspondence between penalty parameter  $\lambda$  and threshold  $c$ , meaning for any given regression problem, given a specific  $\lambda$  there always exists a corresponding  $c$ , and vice-versa.
- When  $\lambda$  is *large*, then  $c$  is *small*, and vice-versa!
- The ridge constraint is *smooth*, while the lasso constraint is *edgy*. This is nicely illustrated in Figure 6.1.  $\beta_1$  and  $\beta_2$  are not allowed to go outside of the red areas.

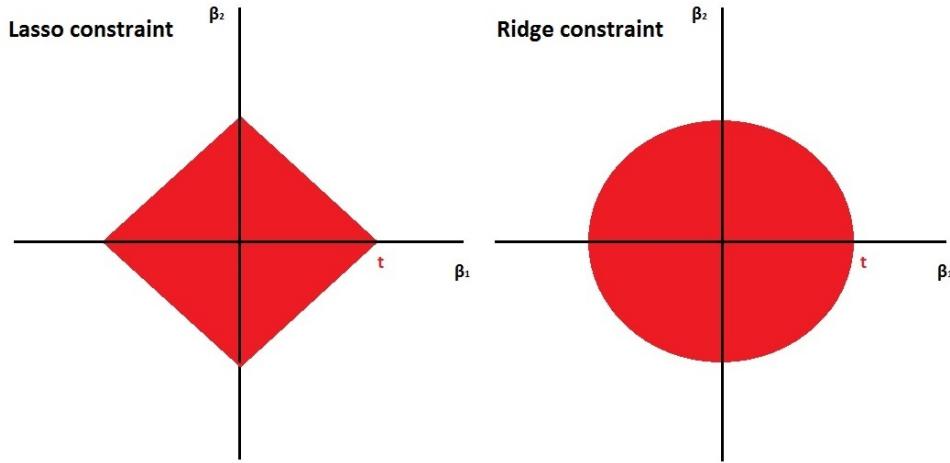


Figure 6.1: Left: the constraint  $|\beta_1| + |\beta_2| \leq c$ , Right: the constraint  $\beta_1^2 + \beta_2^2 \leq c$ .

It is this difference between the smooth and edgy constraints that result in the lasso having coefficient estimates that are exactly zero and ridge not. We illustrate this even further in Figure 6.2. The least squares solution is marked as  $\hat{\beta}$ , while the blue diamond and circle represent the lasso and ridge regression constraints (as in Figure 6.1). If  $c$  is sufficiently large (making  $c$  larger results in the diamond and circle respectively getting larger), then the constraint regions will contain  $\hat{\beta}$ , and so the ridge and lasso estimates will be the same as the least squares estimates.

The contours that are centered around  $\hat{\beta}$  represent regions of constant RSS. As the ellipses expand away from the least squares coefficient estimates, the RSS increases. The lasso and ridge regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region.

Since ridge regression has a circular constraint with no sharp points, this intersection will generally NOT occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero. However, the lasso constraint has *corners* at each of the axes, and

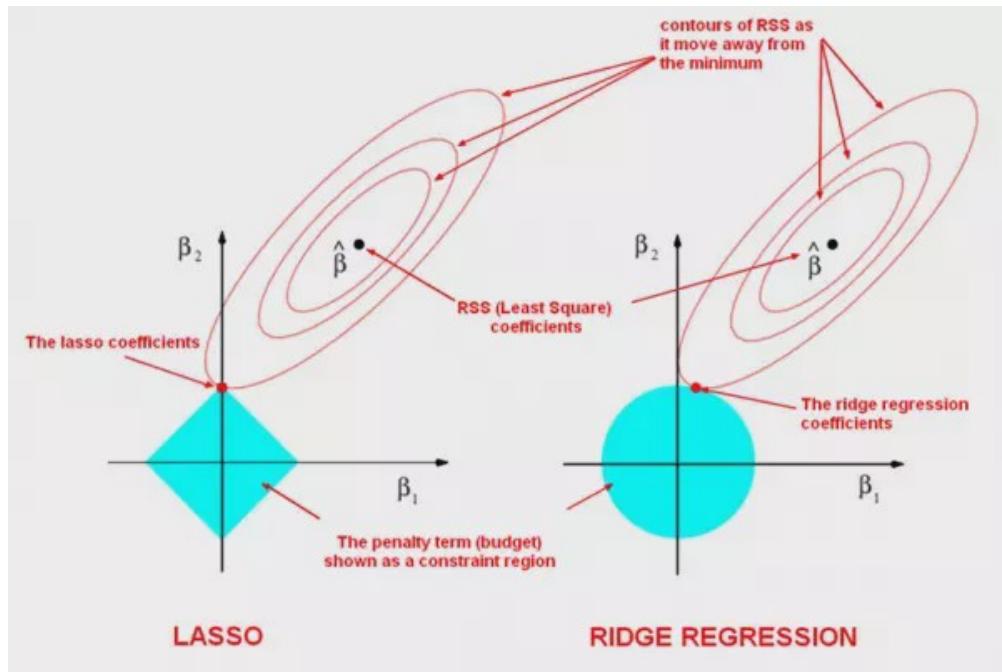


Figure 6.2: Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions  $|\beta_1|+|\beta_2|\leq c$  and  $\beta_1^2+\beta_2^2\leq c$  respectively. The red ellipses are the contours of the Residual Sum of Squares, with the values along each contour being the same, and outer contours having a higher value (least squares  $\hat{\beta}$  has minimal RSS).

so the ellipse will often intersect the constraint region at an axis (in fact in the higher dimensional case, the chance of the hyper-ellipse hitting the corners of the hyper-diamond gets much higher). When this occurs, one of the coefficient estimates will equal zero. In Figure 6.2, the intersection occurs at  $\beta_1 = 0$ , and so the resulting model will only include  $\beta_2$ .

### Different $\lambda$ 's

- Both ridge and lasso are *convex optimisation* problems Boyd and Vandenberghe [2009].
- In optimisation, convexity is a desired property.
- In fact, the ridge solution exists in *closed form* (meaning we have a formula for it).
- Lasso does not have a closed form solution, but nowadays there exist very efficient optimisation algorithms for the lasso solution.
- Package `glmnet` in R implements *coordinate descent* (very fast) – same time to solve ridge and lasso.
- Due to clever optimisation tricks, finding the solutions for multiple  $\lambda$ 's takes the *same time* as finding the solution for one single  $\lambda$  value. This is very useful: it implies that we can obtain the solutions for a grid of values of  $\lambda$  very fast and then pick a specific  $\lambda$  that is optimal in regard to some objective function (see Chapter 6.4).

### Predictive Performance

So, lasso has a clear advantage over ridge - it can set coefficients equal to zero!

This must mean that it is also better for prediction, right? Well, not necessarily. Things are a bit more complicated than that...it all comes down to the nature of the actual underlying mechanism. Let's explain this rather cryptic answer...

Generally...

- When the actual data-generating mechanism is *sparse lasso has the advantage*.
- When the actual data-generating mechanism is *dense ridge has the advantage*.

**Sparse mechanisms:** Few predictors are relevant to the response → good setting for lasso regression.

**Dense mechanisms:** A lot of predictors are relevant to the response → good setting for ridge regression.

Figure 6.3 roughly illustrates how prediction mean square error may depend on the density of the mechanism (number of predictors).

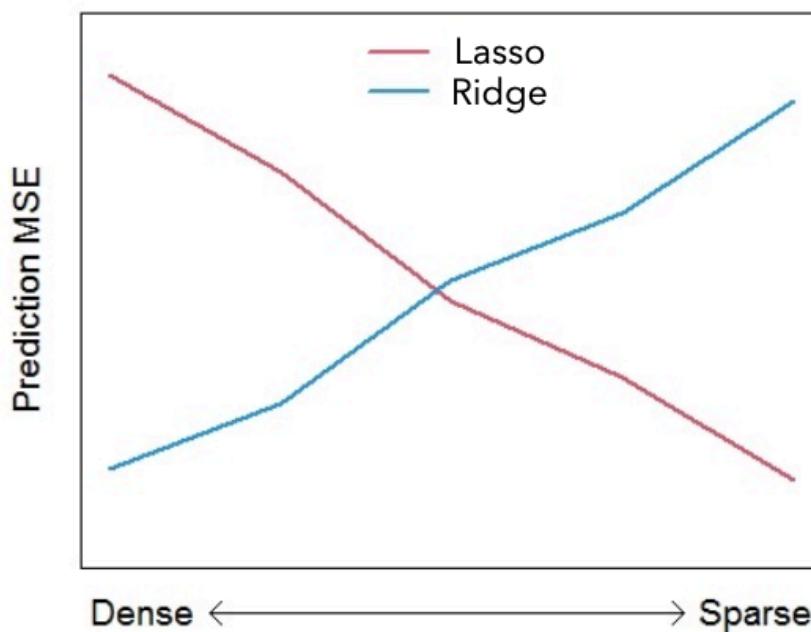


Figure 6.3: Rough illustration of the effect of mechanism density on prediction Mean Squared Error.

This figure really is a rough *extreme* illustration! The actual performance will also depend upon:

- Signal strength (the magnitude of the effects of the relevant variables).
- The correlation structure among predictors.
- Sample size  $n$  vs. number of predictors  $p$ .

**So...which is better?**

- In general, none of the two shrinkage methods will dominate in terms of predictive performance under all settings.

- Lasso performs better when few predictors have a substantial effect on the response variable.
- Ridge performs better when a lot of predictors have a substantial effect on the response.
- Keep in mind: *a few* and *a lot* are always relative to the total number of available predictors.
- Overall: in most applications, lasso is more robust.

**Question:** What should I do if I have to choose between ridge and lasso?

**Answer:** Keep a part of the data for testing purposes only and compare the predictive performance of the two methods.

## Practical Demonstration

We will continue analysing the `Hitters` dataset included in package `ISLR`, this time applying lasso regression.

We perform the usual steps of removing missing data as we have seen in previous practical demonstrations.

```
library(ISLR)
Hitters = na.omit(Hitters)
dim(Hitters)

## [1] 263 20
sum(is.na(Hitters))

## [1] 0
```

We will now use `glmnet()` to implement lasso regression.

```
library(glmnet)
```

Having learned how to implement ridge regression in previous section the analysis is very easy. To run lasso all we have to do is remove the option `alpha = 0`. We do not need to specify explicitly `alpha = 1` because this is the default option in `glmnet()`.

In this part again we call the response `y` and extract the matrix of predictors, which we call `x` using the command `model.matrix()`.

```
y = Hitters$Salary
x = model.matrix(Salary~., Hitters) [,-1] # Here we exclude the first column
# because it corresponds to the
# intercept.
```

```
head(x)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun
## -Alan Ashby	315	81	7	24	38	39	14	3449	835	69
## -Alvin Davis	479	130	18	66	72	76	3	1624	457	63
## -Andre Dawson	496	141	20	65	78	37	11	5628	1575	225
## -Andres Galarraga	321	87	10	39	42	30	2	396	101	12
## -Alfredo Griffin	594	169	4	74	51	35	11	4408	1133	19

```

## -Al Newman      185   37     1   23    8   21    2   214   42    1
##                               CRuns  CRBI  CWalks LeagueN DivisionW PutOuts Assists Errors
## -Alan Ashby      321   414    375     1       1    632    43    10
## -Alvin Davis     224   266    263     0       1    880    82    14
## -Andre Dawson    828   838    354     1       0    200    11     3
## -Andres Galarraga 48    46     33     1       0    805    40     4
## -Alfredo Griffin 501   336    194     0       1    282   421    25
## -Al Newman       30    9     24     1       0     76   127     7
##                               NewLeagueN
## -Alan Ashby          1
## -Alvin Davis          0
## -Andre Dawson          1
## -Andres Galarraga        1
## -Alfredo Griffin        0
## -Al Newman             0

```

We then run lasso regression.

```
lasso = glmnet(x, y)
```

lambda and beta are still the most important components of the output.

```
names(lasso)
```

```

## [1] "a0"        "beta"       "df"         "dim"        "lambda"     "dev.ratio"
## [7] "nulldev"   "npasses"    "jerr"       "offset"     "call"       "nobs"
lasso$lambda

## [1] 255.2820965 232.6035386 211.9396813 193.1115442 175.9560468 160.3245966
## [7] 146.0818013 133.1042967 121.2796778 110.5055255 100.6885192 91.7436287
## [13] 83.5933775  76.1671723  69.4006906  63.2353245  57.6176726  52.4990774
## [19] 47.8352040  43.5856563  39.7136268  36.1855776  32.9709506  30.0419022
## [25] 27.3730624  24.9413150  22.7255973  20.7067179  18.8671902  17.1910810
## [31] 15.6638727  14.2723374  13.0044223  11.8491453  10.7964999  9.8373686
## [37] 8.9634439   8.1671562   7.4416086   6.7805166   6.1781542   5.6293040
## [43] 5.1292121   4.6735471   4.2583620   3.8800609   3.5353670   3.2212947
## [49] 2.9351238   2.6743755   2.4367913   2.2203135   2.0230670   1.8433433
## [55] 1.6795857   1.5303760   1.3944216   1.2705450   1.1576733   1.0548288
## [61] 0.9611207   0.8757374   0.7979393   0.7270526   0.6624632   0.6036118
## [67] 0.5499886   0.5011291   0.4566102   0.4160462   0.3790858   0.3454089
## [73] 0.3147237   0.2867645   0.2612891   0.2380769   0.2169268   0.1976557
## [79] 0.1800965   0.1640972

```

```
dim(lasso$beta)
```

```
## [1] 19 80
```

```
lasso$beta[,1:3] # this returns only the predictor coefficients
```

```
## 19 x 3 sparse Matrix of class "dgCMatrix"
```

```

##          s0        s1        s2
## AtBat      . .
## Hits       . .
## HmRun      . .
## Runs       . .
## RBI        . .
## Walks      . .
## Years      . .
## CAtBat     . .
## CHits      . .
## CHmRun     . .
## CRuns      . .    0.01515244
## CRBI       . .07026614 0.11961370
## CWalks     . .
## LeagueN    . .
## DivisionW . .
## PutOuts    . .
## Assists    . .
## Errors     . .
## NewLeagueN . .

coef(lasso)[,1:3] # this returns intercept + predictor coefficients

```

```

## 20 x 3 sparse Matrix of class "dgCMatrix"
##          s0        s1        s2
## (Intercept) 535.9259 512.70866859 490.92996088
## AtBat      . .
## Hits       . .
## HmRun      . .
## Runs       . .
## RBI        . .
## Walks      . .
## Years      . .
## CAtBat     . .
## CHits      . .
## CHmRun     . .
## CRuns      . .    0.01515244
## CRBI       . .07026614 0.11961370
## CWalks     . .
## LeagueN    . .
## DivisionW . .
## PutOuts    . .
## Assists    . .
## Errors     . .
## NewLeagueN . .

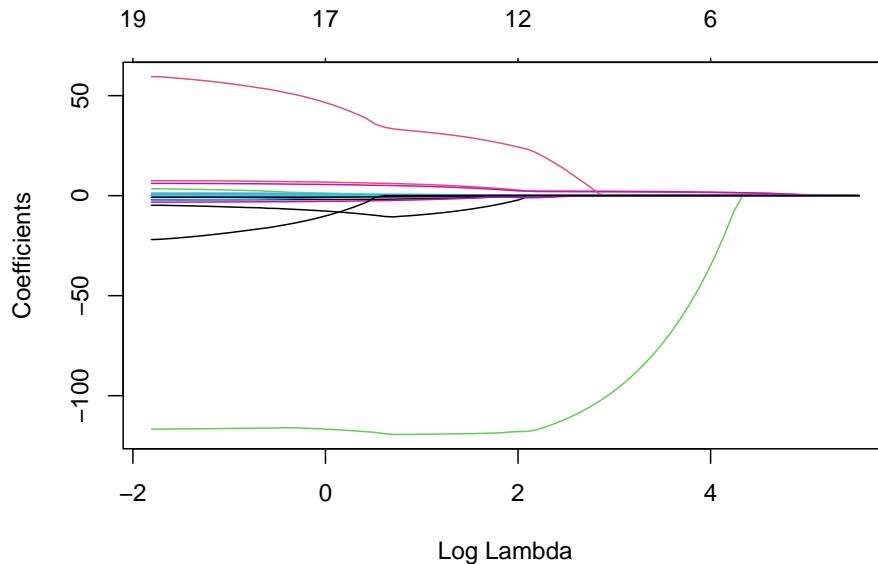
```

We see that `lasso$beta` has 19 rows (number of predictors) and 80 columns. (Note the number of  $\lambda$  is less than the default setting, 100 (The reason lies in the stopping criteria of

the algorithm. According to the default internal settings, the computations stop if either the fractional change in deviance (relates to the log-likelihoods) down the path is less than  $10^5$  or the fraction of explained deviance (similar to the concept of  $R^2$ ) reaches 0.999). Using `coef` instead includes the intercept term. The key difference this time (in comparison to the application to ridge regression is that some of the elements of the output are replaced with a “.”, indicating that the corresponding predictor is not included in the lasso regression model with that value of  $\lambda$ .

We now produce a plot of regularisation (Regularisation refers to the technique that discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.) paths based on  $\log \lambda$ . The numbers along the top now correspond to the number of predictors in the model for the corresponding (log)  $\lambda$  values. Recall the row of 19's across the top of the corresponding plot for ridge regression - this is because for ridge we always include all predictors.

```
plot(lasso, xvar = 'lambda')
```



## 6.4 Choosing $\lambda$

### Importance of $\lambda$

- As we have seen, the penalty parameter  $\lambda$  is of crucial importance in penalised regression.
- For  $\lambda = 0$  we essentially just get the LS estimates of the full model.
- For very large  $\lambda$ : all ridge estimates become extremely small, while all lasso estimates are exactly zero!

We require a principled way to fine-tune  $\lambda$  in order to get optimal results.

### Selection Criteria?

- In Chapter 4.6, we used the selection criteria  $C_p$ , BIC and adjusted- $R^2$  as criteria for choosing particular models during model search methods.

- Perhaps we can do that here, for instance, by defining a grid of values for  $\lambda$  and calculating the corresponding  $C_p$  under each model, before then finding the model with the corresponding  $\lambda$  that yields the lowest  $C_p$  value?
  - Not really - the problem is that all these techniques depend on model dimensionality (number of predictors).
  - With model-search methods we choose a model with  $d$  predictors, so it is clear that the model's dimensionality is  $d$ .
  - Shrinkage methods, however, penalise dimensionality in a different way - the very notion of model dimensionality becomes obscure somehow - does a constrained predictor count towards dimensionality in the same way as an unconstrained one?

## Cross-Validation

The only viable strategy is  $K$ -fold cross-validation. The steps are the following:

1. Choose the number of folds  $K$ .
  2. Split the data accordingly into training and testing sets.
  3. Define a grid of values for  $\lambda$ .
  4. For each  $\lambda$  calculate the validation Mean Squared Error (MSE) within each fold.
  5. For each  $\lambda$  calculate the overall cross-validation MSE.
  6. Locate under which  $\lambda$  cross-validation MSE is minimised. This value of  $\lambda$  is known as the *minimum\_CV*  $\lambda$ .

Seems difficult, but fortunately `glmnet` in R will do all of these things for us automatically! We will explore using cross-validation for choosing  $\lambda$  further via practical demonstration.

## Practical Demonstration

We will continue analysing the `Hitters` dataset included in package `ISLR`.

Perform the usual steps of removing missing data.

```
library(ISLR)
Hitters = na.omit(Hitters)
dim(Hitters)

## [1] 263 20
sum(is.na(Hitters))

## [1] 0
```

## Ridge regression

In this part again we call the response  $y$  and extract the matrix of predictors, which we call  $x$  using the command `model.matrix()`.

```
y = Hitters$Salary  
x = model.matrix(Salary~., Hitters) [,-1] # Here we exclude the first column  
# because it corresponds to the
```

```

# intercept term.

head(x)

##          AtBat  Hits  HmRun  Runs  RBI  Walks  Years  CAtBat  CHits  CHmRun
## -Alan Ashby     315    81      7    24    38     39    14    3449    835      69
## -Alvin Davis    479   130     18    66    72     76     3    1624    457      63
## -Andre Dawson   496   141     20    65    78     37    11    5628   1575     225
## -Andres Galarraga 321    87     10    39    42     30     2    396    101      12
## -Alfredo Griffin 594   169      4    74    51     35    11   4408   1133      19
## -Al Newman      185    37      1    23     8     21     2    214     42      1
##          CRuns  CRBI  CWalks LeagueN DivisionW PutOuts Assists Errors
## -Alan Ashby     321   414     375      1      1    632     43     10
## -Alvin Davis    224   266     263      0      1    880     82     14
## -Andre Dawson   828   838     354      1      0    200     11      3
## -Andres Galarraga 48    46     33      1      0    805     40      4
## -Alfredo Griffin 501   336     194      0      1    282     421     25
## -Al Newman      30     9     24      1      0     76    127      7
##          NewLeagueN
## -Alan Ashby      1
## -Alvin Davis     0
## -Andre Dawson     1
## -Andres Galarraga 1
## -Alfredo Griffin 0
## -Al Newman       0

```

For implementing cross-validation (CV) in order to select  $\lambda$  we will have to use the function `cv.glmnet()` from library `glmnet`. After loading library `glmnet`, we have a look at the help document by typing `?cv.glmnet`. We see the argument `nfold` = 10, so the default option is 10-fold CV (this is something we can change if we want to).

To perform cross-validation for selection of  $\lambda$  we simply run the following:

```

library(glmnet)

set.seed(1)
ridge.cv = cv.glmnet(x, y, alpha=0)

```

We extract the names of the output components as follows:

```

names( ridge.cv )

## [1] "lambda"      "cvm"        "cvsd"        "cvup"        "cvlo"
## [6] "nzero"        "call"        "name"        "glmnet.fit"  "lambda.min"
## [11] "lambda.1se"   "index"

```

`lambda.min` provides the optimal value of  $\lambda$  as suggested by the cross-validation MSEs.

```

ridge.cv$lambda.min

```

```

## [1] 25.52821

```

`lambda.1se` provides a different value of  $\lambda$ , known as the *1-standard error (1-se)  $\lambda$* . This is the maximum value that  $\lambda$  can take while still falling within the one standard error interval of the minimum\_CV  $\lambda$ .

```
ridge.cv$lambda.1se
```

```
## [1] 1843.343
```

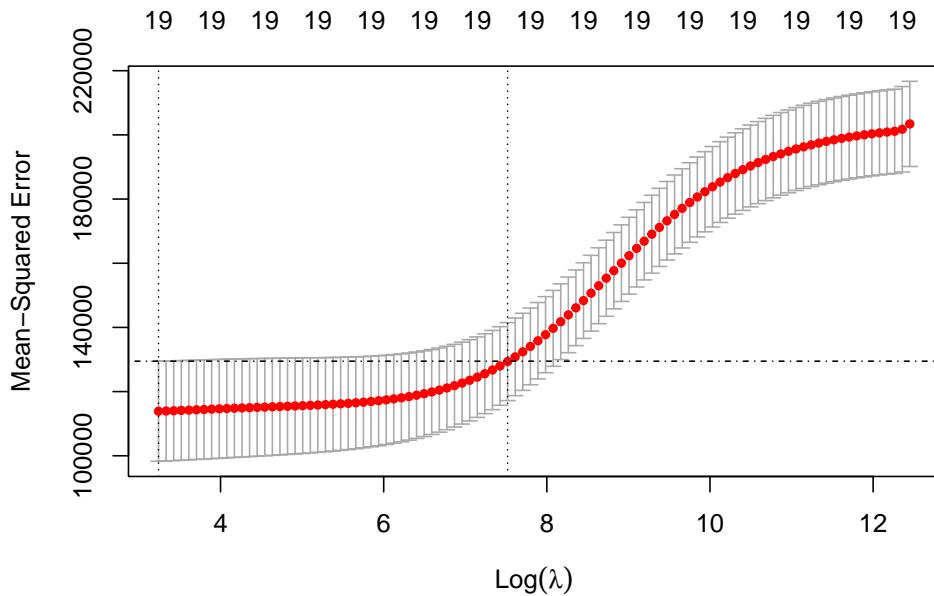
We see that in this case the 1-se  $\lambda$  is much larger than the min-CV  $\lambda$ . This means that we expect the coefficients under 1-se  $\lambda$  to be much smaller. We can see that this is indeed the case by looking at the corresponding coefficients from the two models, rounding them to 3 decimal places.

```
round(cbind(
  coef(ridge.cv, s = 'lambda.min'),
  coef(ridge.cv, s = 'lambda.1se')),
  digits = 3 )
```

```
## 20 x 2 sparse Matrix of class "dgCMatrix"
##           s1      s1
## (Intercept) 81.127 159.797
## AtBat       -0.682  0.102
## Hits        2.772  0.447
## HmRun       -1.366  1.289
## Runs         1.015  0.703
## RBI          0.713  0.687
## Walks        3.379  0.926
## Years       -9.067  2.715
## CAtBat      -0.001  0.009
## CHits        0.136  0.034
## CHmRun       0.698  0.254
## CRuns        0.296  0.069
## CRBI         0.257  0.071
## CWalks       -0.279  0.066
## LeagueN      53.213  5.396
## DivisionW   -122.834 -29.097
## PutOuts       0.264  0.068
## Assists       0.170  0.009
## Errors        -3.686 -0.236
## NewLeagueN   -18.105  4.458
```

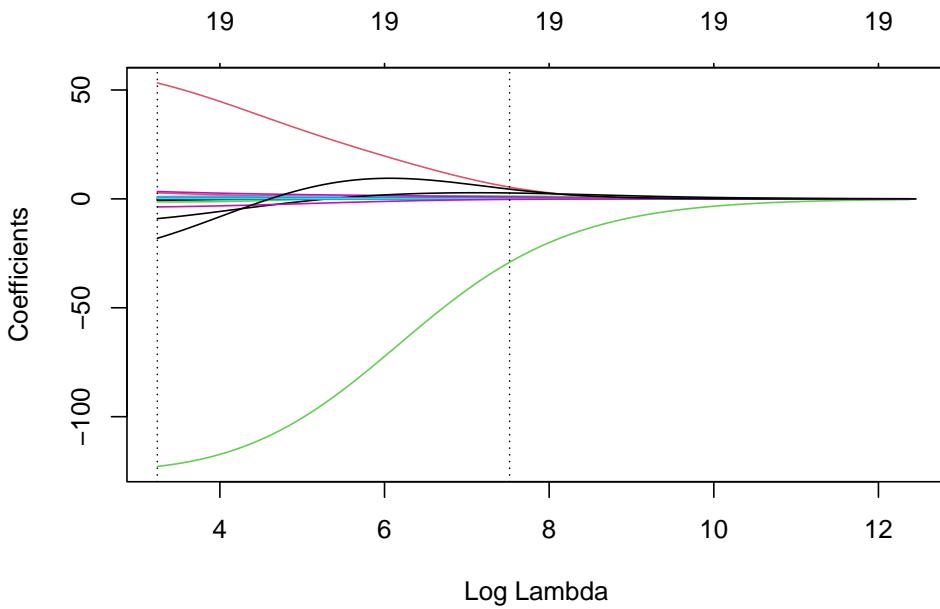
Let's plot the results based on the output of `ridge.cv` using `plot`. What is plotted is the estimated CV MSE for each value of (log)  $\lambda$  on the  $x$ -axis. The dotted line on the far left indicates the value of  $\lambda$  which minimises CV error. The dotted line roughly in the middle of the  $x$ -axis indicates the 1-standard-error  $\lambda$  - recall that this is the maximum value that  $\lambda$  can take while still falling within the one standard error interval of the minimum-CV  $\lambda$ . The second line of code has manually added a dot-dash horizontal line at the upper end of the 1-standard deviation interval of the MSE at the minimum-CV  $\lambda$  to illustrate this point further.

```
plot(ridge.cv)
abline( h = ridge.cv$cvup[ridge.cv$index[1]], lty = 4 )
```



We can also plot the ridge regularisation paths and add the minimum-CV and 1-se  $\lambda$  values to the plot.

```
ridge = glmnet(x, y, alpha = 0)
plot(ridge, xvar = 'lambda')
abline(v = log(ridge.cv$lambda.min), lty = 3) # careful to use the log here
# and below
abline(v = log(ridge.cv$lambda.1se), lty = 3)
```



**Important note:** Any method/technique that relies on validation/cross-validation is subject to variability. If we re-run this code under a different random seed results will change.

## Sparsifying the ridge coefficients?

One thing mentioned in Chapter 6.2 is that one can potentially consider some kind of *post-hoc* analysis in order to set some of the ridge coefficients equal to zero. A very simplistic approach is to just examine the ranking of the absolute coefficients and decide to use a “*cutting-threshold*”. We can use a combination of the commands `sort()` and `abs()` for this.

```
beta.1se = coef(ridge.cv, s = 'lambda.1se')[-1]
rank.coef = sort(abs(beta.1se), decreasing = TRUE)
rank.coef

## [1] 29.096663826 5.396487460 4.457548079 2.714623469 1.289060569
## [6] 0.925962429 0.702915318 0.686866069 0.446840519 0.253594871
## [11] 0.235989099 0.102483884 0.071334608 0.068874010 0.067805863
## [16] 0.066114944 0.034359576 0.009201998 0.008746278
```

The last two coefficients are quite small, so we could potentially set these two equal to zero (setting a threshold of say, 0.01). Equivalently we may set a larger threshold of say, 1, in which case we would remove far more variables from the model. We could check whether this sparsification would offer any benefits in terms of predictive performance by, for example, using  $K$ -fold cross-validation to evaluate the standard ridge regression fit (using `glmnet` with `alpha = 0`) with the  $i$ th fold excluded, and predicting the output for the  $i$ th fold using `predict` (see `?predict.glmnet` for details).

## Lasso regression

As we did with ridge, we can use the function `cv.glmnet()` to find the min-CV  $\lambda$  and the 1 standard error (1-se)  $\lambda$  under lasso. The code is the following.

```
set.seed(1)
lasso.cv = cv.glmnet(x, y)
lasso.cv$lambda.min

## [1] 1.843343
lasso.cv$lambda.1se

## [1] 69.40069
```

We see that in this case the 1-se  $\lambda$  is much larger than the min-CV  $\lambda$ . This means that we expect the coefficients under 1-se  $\lambda$  to be much smaller (maybe exactly zero). Let’s have a look at the corresponding coefficients from the two models, rounding them to 3 decimal places.

```
round(cbind(
  coef(lasso.cv, s = 'lambda.min'),
  coef(lasso.cv, s = 'lambda.1se')),
3)

## 20 x 2 sparse Matrix of class "dgCMatrix"
##           s1      s1
## (Intercept) 142.878 127.957
```

```

## AtBat      -1.793   .
## Hits       6.188   1.423
## HmRun      0.233   .
## Runs       .
## RBI        .
## Walks      5.148   1.582
## Years     -10.393  .
## CAtBat    -0.004   .
## CHits      .
## CHmRun     0.585   .
## CRuns      0.764   0.160
## CRBI       0.388   0.337
## CWalks     -0.630   .
## LeagueN    34.227  .
## DivisionW -118.981 -8.062
## PutOuts    0.279   0.084
## Assists    0.224   .
## Errors     -2.436   .
## NewLeagueN .

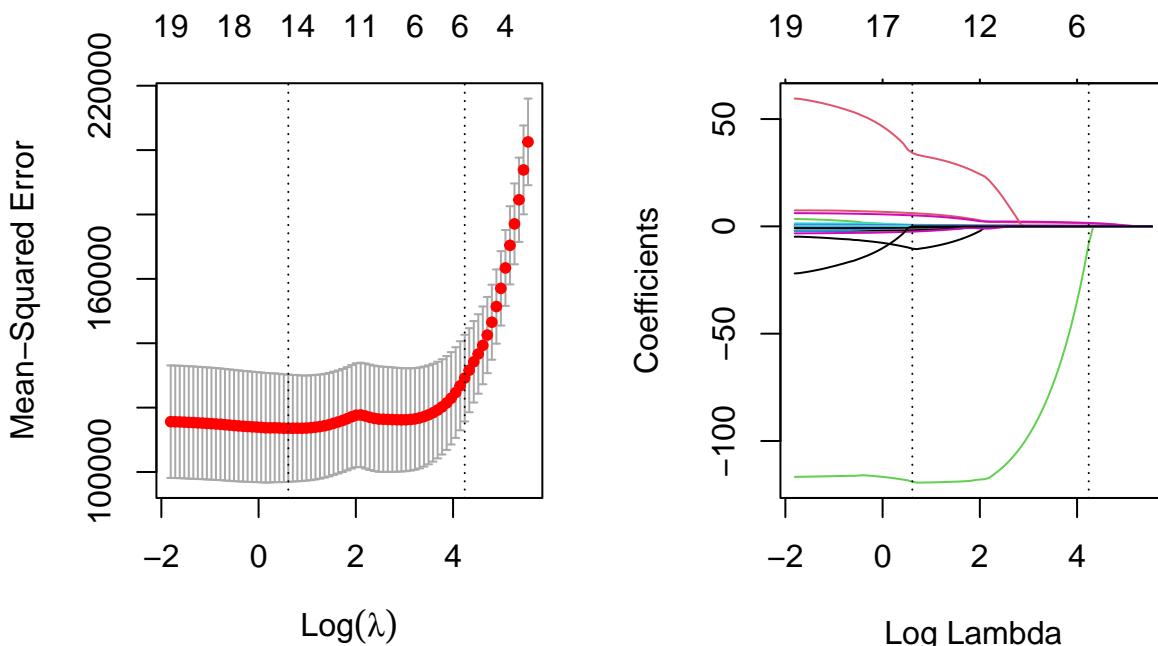
```

Finally, let's plot the results of the object `lasso.cv` which we have created, and compare them to that of the `lasso` object generated in previous section.

```

par(mfrow=c(1,2))
plot(lasso.cv)
lasso = glmnet(x, y)
plot(lasso, xvar = 'lambda')
abline(v = log(lasso.cv$lambda.min), lty = 3) # careful to use the log here
                                                # and below
abline(v = log(lasso.cv$lambda.1se), lty = 3)

```



Notice now how the numbers across the top of the left-hand plot decrease as  $\lambda$  increases. This is indicating how many predictors are included for the corresponding  $\lambda$  value.

## Comparing Predictive Performance for Different $\lambda$

We can investigate which of the two different  $\lambda$ -values (minimum-CV and 1-se) may be preferable in terms of predictive performance using the following code. Here, we split the data into a training and test set. We then use cross-validation to find minimum-CV  $\lambda$  and 1-se  $\lambda$ , and use the resulting model to predict the response at the test data, comparing it to the actual response values by MSE. We then repeat this procedure many times using different splits in the data.

```

repetitions = 50
mse.1 = c()
mse.2 = c()

set.seed(1)
for(i in 1:repetitions){

  # Step (i) data splitting
  training.obs = sample(1:263, 175)
  y.train = Hitters$Salary[training.obs]
  x.train = model.matrix(Salary~., Hitters[training.obs, ])[,-1]
  y.test = Hitters$Salary[-training.obs]
  x.test = model.matrix(Salary~., Hitters[-training.obs, ])[,-1]

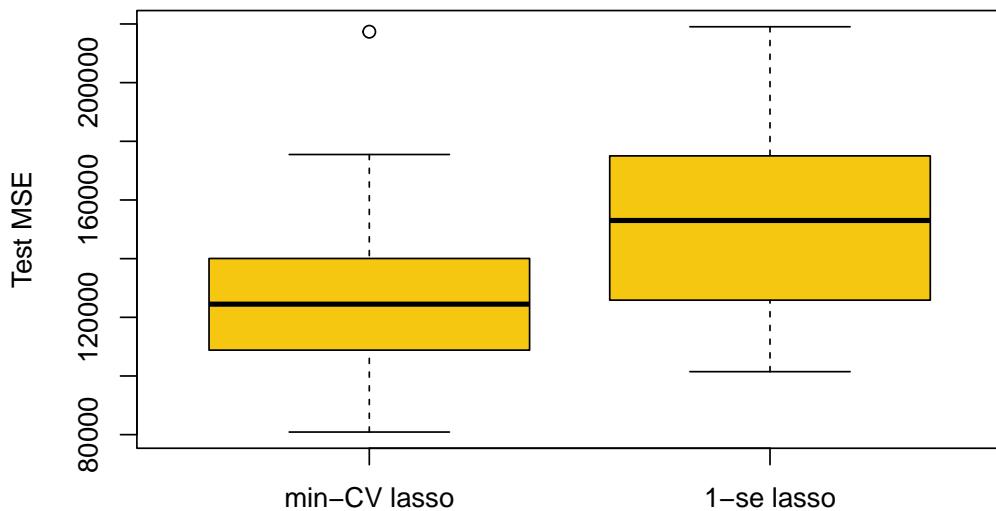
  # Step (ii) training phase
  lasso.train = cv.glmnet(x.train, y.train)

  # Step (iii) generating predictions
  predict.1 = predict(lasso.train, x.test, s = 'lambda.min')
  predict.2 = predict(lasso.train, x.test, s = 'lambda.1se')

  # Step (iv) evaluating predictive performance
  mse.1[i] = mean((y.test-predict.1)^2)
  mse.2[i] = mean((y.test-predict.2)^2)
}

par(mfrow=c(1,1))
boxplot(mse.1, mse.2, names = c('min-CV lasso','1-se lasso'),
        ylab = 'Test MSE', col = 7)

```



The approach based on min-CV is better, which makes sense as it minimised the cross-validation error (in `cv.glmnet`) in the first place. The 1-se  $\lambda$  can be of benefit in some cases for selecting models with fewer predictors that may be more interpretable. However, in applications where the 1-se  $\lambda$  is much larger than the min-CV  $\lambda$  (like here) we in general expect results as above.



# Chapter 7

## Principal Component Analysis

### 7.1 Reminder

So far, we have focused on linear models of the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$

and argued in favour of selecting *sparser* models which are *simpler to interpret* and also have *better predictive performance* in comparison to the model that includes all  $p$  predictors.

Specifically, we saw

- *Model-search methods*: discrete procedures which require full or partial enumeration of all possible models and select one best model.
- *Coefficient shrinkage methods*: continuous procedures which shrink the coefficients of the full model to zero (penalised regression).

Note that both are *supervised learning* approaches and effectively reduce the *dimensionality* of the problem.

There is another alternative *unsupervised learning* approach for dimension reduction, where only the predictors are of the interest. Here comes the principal component analysis.

### 7.2 Linear approximation

Consider that there is a  $q$ -dimensional random vector

$$X \sim (m, \Sigma),$$

where

$$m = E(X) = \begin{bmatrix} m_1 \\ \vdots \\ m_q \end{bmatrix}$$

and

$$\Sigma = \text{Var}(X) = \begin{bmatrix} \Sigma_{11} & \dots & \Sigma_{1q} \\ \vdots & \ddots & \vdots \\ \Sigma_{q1} & \dots & \Sigma_{qq} \end{bmatrix}$$

If we were to approximate  $X$  through a single straight line, what is the “best” line?

One approach to solve this problem dates back to Pearson [1901]: Minimize the expected squared distances between  $X$  and their projections  $X'$  onto the line. We make in the following use of geometrical considerations which are provided in Figure 7.1.

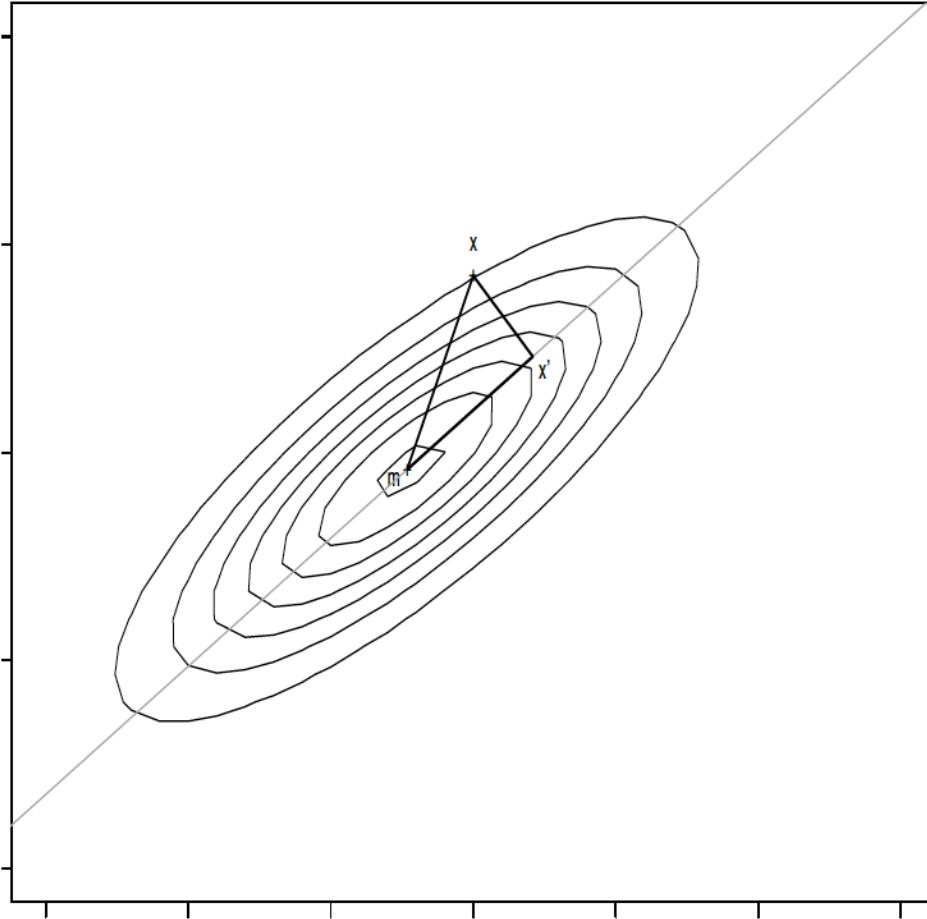


Figure 7.1: Geometrical considerations for PCA

By Pythagoras,

$$E(\overline{XX'}^2) = E(\overline{mX}^2) - E(\overline{mX'}^2)$$

where  $\overline{XX'}$  denotes the length of the line segment connecting  $X$  and  $X'$ , i.e.  $\overline{XX'} = \|X - X'\|$ . As  $E(\overline{mX}^2)$  does not depend on the fitted line, minimizing  $E(\overline{XX'}^2)$  is the same as maximizing  $E(\overline{mX'}^2)$ . Investigating this quantity further, let  $\gamma$  be a unit vector pointing in the direction of the vector  $X' - m$  and let  $t \in \mathbb{R}$  be a scalar such that the projected point is

given by

$$X' = m + t\gamma \in \mathbb{R}^q.$$

The Euclidean distance between  $m$  and  $X'$ ,  $\overline{mX'}$ , equals  $|t|$ . We have by definition

$$\cos(\gamma, X - m) = \frac{t}{\|X - m\|}$$

and, using the properties of the dot product gives

$$\gamma^T(X - m) = \|\gamma\| \|X - m\| \cos(\gamma, X - m) = t.$$

To maximizing  $E(\overline{mX'})^2$ , we have

$$\begin{aligned} E(\overline{mX'})^2 &= E(\overline{mX'} \overline{mX'}) = E(\gamma^T(X - m)(X - m)^T\gamma) = \\ &= \text{Var}(\gamma^T(X - m)) = \text{Var}(\gamma^T X) = \gamma^T \text{Var}(X) \gamma = \gamma^T \Sigma \gamma. \end{aligned}$$

Hence, we need to maximize  $\gamma^T \Sigma \gamma$  under the constraint  $\|\gamma\|^2 = \gamma^T \gamma = 1$ . This is a constrained maximization problem which can be addressed through the use of a Lagrange multiplier. Define

$$P(\gamma) = \gamma^T \Sigma \gamma - \lambda(\gamma^T \gamma - 1).$$

Then from Chapter 12 Mardia et al. [1979]

$$\frac{\partial P}{\partial \gamma} = 2\Sigma\gamma - 2\lambda\gamma,$$

and setting this equal to zero yields

$$\Sigma\gamma = \lambda\gamma \tag{7.1}$$

So,  $\gamma$  must be an eigenvector of  $\Sigma$ . But which one? (To fix terms, order the  $q$  eigenvalues of  $\Sigma \in \mathbb{R}^{q \times q}$  such that  $\lambda_1 \geq \dots \geq \lambda_q$ , and let the orthogonal vectors  $\gamma_1, \dots, \gamma_q$  denote the corresponding eigenvectors.) Multiplying ((7.1)) with  $\gamma^T$ , one has

$$\begin{aligned} \gamma^T \Sigma \gamma &= \lambda \gamma^T \gamma \\ \text{Var}(\gamma^T X) &= \lambda \end{aligned}$$

The left hand side is now exactly what we want to maximize. Maximizing  $\text{Var}(\gamma^T X)$  means then that we need to choose the eigenvector  $\gamma_1$  corresponding to the largest eigenvalue  $\lambda_1$ .

Some notes:

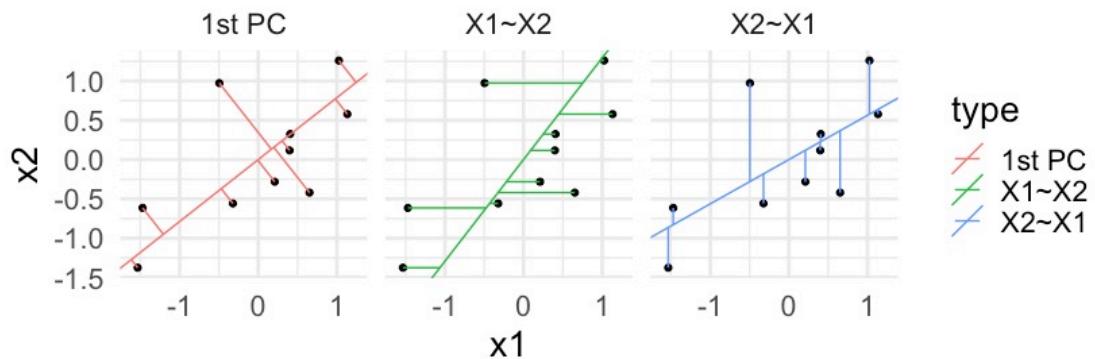
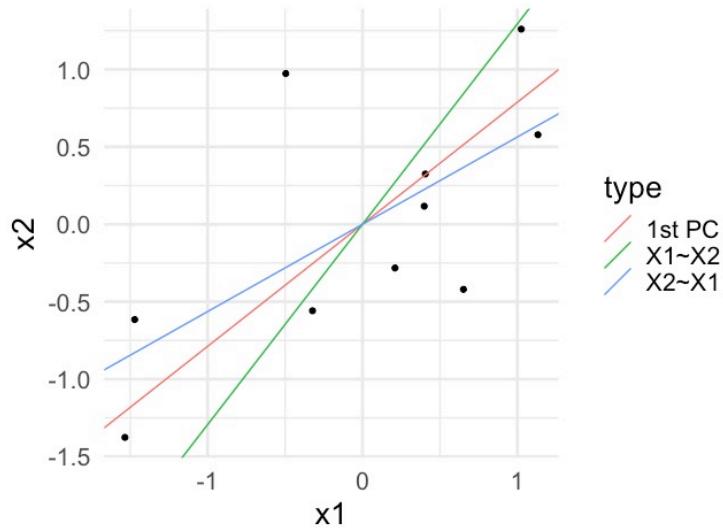
- The new random variable  $\gamma_1^T X$  is that linear combination of  $X = (X_1, \dots, X_q)^T$  with maximal variance, and is called the *first principal component* (PC) of  $X$ . The line

$$g_1(t) = m + t\gamma_1, \quad (t \in \mathbb{R})$$

is the corresponding *first principal component line*.

- Similarly, we define higher-order PC's: The  $j$ -th eigenvector  $\gamma_j$  maximizes  $\text{Var}(\gamma^T X)$  over all  $\gamma$  which are orthogonal to  $\gamma_1, \dots, \gamma_{j-1}$ . The  $j$ -th PC is given by  $\gamma_j^T X$ , and  $g_j(t) = m + t\gamma_j$  is the corresponding  $j$ -th PC line (See Krzanowski [2000]).
- For data  $Z$ , we need to replace  $m$  by  $\hat{m}$ , and  $\Sigma$  by some estimate  $\hat{\Sigma}$  (ML or sample), yielding  $\hat{\lambda}_1, \dots, \hat{\lambda}_q, \hat{\gamma}_1, \dots, \hat{\gamma}_q$ .
- From now on for the rest of this section, we will omit all “hats” on  $\Sigma$ ,  $m$ ,  $\lambda$ , etc, meaning that the formulas hold for either the random vector or the corresponding estimate, depending on context.

Note that principal components minimize the (sum of squared) *orthogonal* distances between data  $X_1, \dots, X_n$  and the line, unlike *vertical* distances as in the regression context. For example:



## 7.3 Decomposing variance

Recall that, for the  $j$ -th eigenvector  $\gamma_j$  of  $\Sigma$ , one has

$$\Sigma \gamma_j = \lambda_j \gamma_j \quad j = 1, \dots, q$$

For each  $j$ , either side of this equation is a  $q \times 1$  vector. We can bind these  $q$  vectors together to form  $q \times q$  matrices:

$$\begin{aligned} (\Sigma \gamma_1, \dots, \Sigma \gamma_q) &= (\lambda_1 \gamma_1, \dots, \lambda_1 \gamma_q) \\ \Sigma (\gamma_1, \dots, \gamma_q) &= (\gamma_1, \dots, \gamma_q) \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_q \end{pmatrix} \\ \Sigma \Gamma &= \Gamma \Lambda \\ \Sigma &= \Gamma \Lambda \Gamma^{-1} = \Gamma \Lambda \Gamma^T \end{aligned} \tag{7.2}$$

This decomposition is called the *eigen decomposition* of  $\Sigma$ . Some software packages use this in order to find the  $\lambda'_j$ s and  $\gamma'_j$ s.

Now, we have

$$\lambda_j = \text{Var}(\gamma_j^T X)$$

so the  $\lambda_j$  provide some decomposition of variance. So, each  $\lambda_j$  takes the share of something - but the share of what? Let's therefore compute their sum:

$$\lambda_1 + \dots + \lambda_q = \text{Tr}(\Lambda) = \text{Tr}(\Gamma^T \Sigma \Gamma) = \text{Tr}(\Gamma \Gamma^T \Sigma) = \text{Tr}(\Sigma) \equiv \text{TV}(X)$$

The trace of the variance matrix is called the *total variance*. Therefore

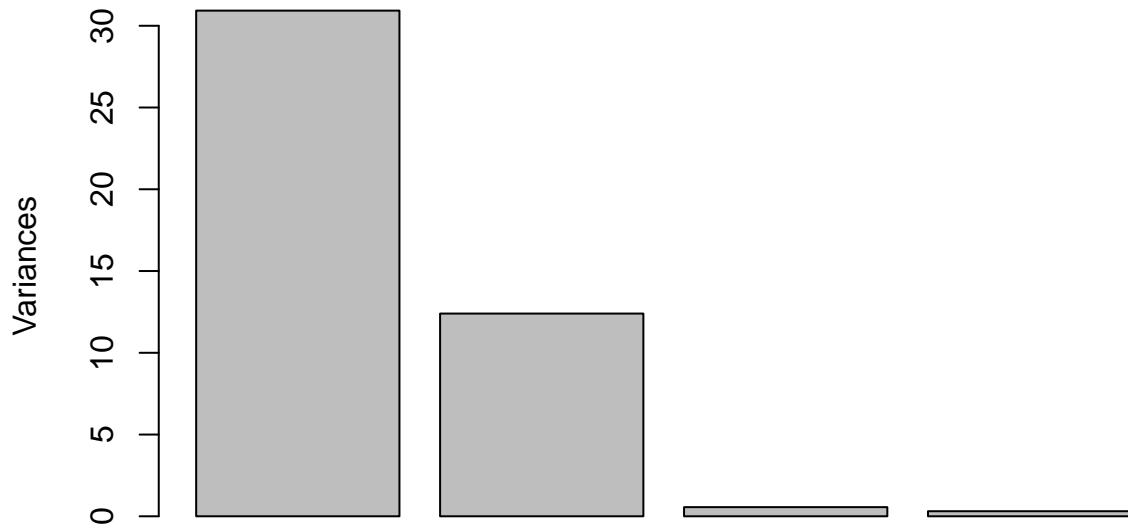
$$\frac{\lambda_j}{\lambda_1 + \dots + \lambda_q} = \frac{\text{Var}(\gamma_j^T X)}{\text{TV}(X)}$$

is the proportion of total variance explained by the  $j$ -th principal component. A simple graphical tool illustrating this decomposition is the *scree plot*, which plots  $\lambda_j$  vs.  $j$ .

### Practical Example

For the 4-dimensional space spanned by the four variables “TAX”, “DLIC”, “INC”, “ROAD” of the fuel consumption data set, we obtain a scree plot in R as follows:

```
fuelcons <- read.table("https://www.maths.dur.ac.uk/users/hailiang.du/data/FUEL.dat", he
fuel <- fuelcons[,c("TAX", "DLIC", "INC", "ROAD")]
fuel.pr <- prcomp(fuel)
plot(fuel.pr)
```

**fuel.pr**

Can we create it by hand? Let's look at the output of the `prcomp` function:

```
fuel.pr
```

```
## Standard deviations (1, ..., p=4):
## [1] 5.5610991 3.5210898 0.7482262 0.5562322
##
## Rotation (n x k) = (4 x 4):
##          PC1        PC2        PC3        PC4
## TAX -0.04687723  0.15579496 -0.96634218  0.19928183
## DLIC  0.99686606 -0.05351563 -0.05128624  0.02763787
## INC   0.01607022 -0.01033709 -0.20428171 -0.97872564
## ROAD -0.06166301 -0.98628452 -0.14772101  0.04023709
```

Here, the item `Standard deviations` contains the (ordered) values

$$SD[\gamma_j^T X] = \sqrt{Var[\gamma_j^T X]} = \sqrt{\lambda_j},$$

and the item `Rotation` is the same as  $\Gamma$ . Both items can be extracted directly from the `prcomp` object (here: `fuel.pr`) through the components `$.sdev` and `$.rot`, respectively. Hence, we get the  $\lambda_j$  via

```
fuel.pr$.sdev^2
```

```
## [1] 30.9258227 12.3980734  0.5598424  0.3093942
```

which is immediately confirmed to be the same as

```
eigen(var(fuel))$values
```

```
## [1] 30.9258227 12.3980734  0.5598424  0.3093942
```

and the proportions of variance explained are given by

```

fuel.pr$sdev^2/ sum(fuel.pr$sdev^2)

## [1] 0.699787972 0.280542985 0.012668086 0.007000957

```

One may also want to look at the built-in R summary output.

```

summary(fuel.pr)

## Importance of components:
##                 PC1      PC2      PC3      PC4
## Standard deviation 5.5611 3.5211 0.74823 0.5562
## Proportion of Variance 0.6998 0.2805 0.01267 0.0070
## Cumulative Proportion 0.6998 0.9803 0.99300 1.0000

```

We would conclude from the scree plot and the summary output that  $d = 2$  principal components capture the essential information provided by the data cloud.

## 7.4 Scaling

If the variables operate on different scales/units, the decomposition of variance may not be meaningful as it depends on the units chosen. It is a problem for the `fuel` data as all units are different, and our decomposition of variance may just reflect which variables are recorded on smaller or larger units! Assume, for instance, that INC had been measured in \\$ (instead of 1000\\$), or the proportions DLIC had been measured on the interval [0,1] (rather than in percent). This would have had a massive impact on  $\Sigma$  in the corresponding directions, and, hence, on the principal components found for this data set (Try this!).

To avoid problems of this type, standardize all variables by dividing through their standard deviation, i.e. set  $\tilde{X}_j = X_j/SD(X_j)$ , and then apply PCA to  $\tilde{X} = (\tilde{X}_1, \dots, \tilde{X}_q)^T$ . Also, observe that for eigenvalues  $\lambda_1, \dots, \lambda_q$  of the variance matrix of  $\tilde{X}$ ,

$$\lambda_1 + \dots + \lambda_q = \dots = \text{Tr}(Var[\tilde{X}]) = 1 + \dots + 1 = q$$

i.e. the eigenvalues of the correlation matrix always add up to the dimension of the random vector.

### Practical Example - Scaling for fuel data

```

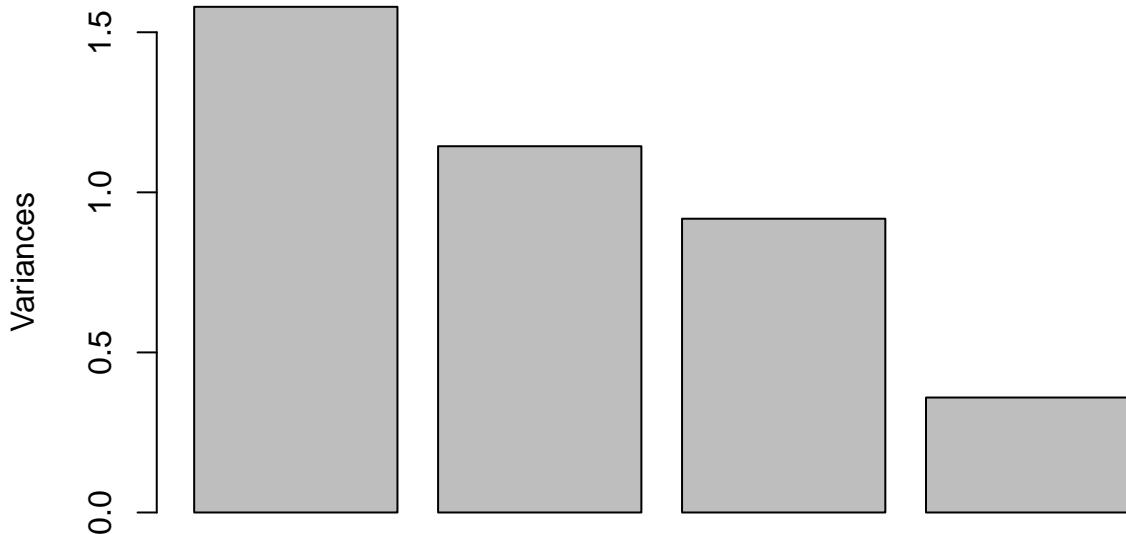
fuel.pr1 <- prcomp(fuel, scale=TRUE)
fuel.pr1

## Standard deviations (1, ..., p=4):
## [1] 1.2567608 1.0695586 0.9578833 0.5992131
##
## Rotation (n x k) = (4 x 4):
##                 PC1          PC2          PC3          PC4
## TAX    0.7101892 -0.08133737  0.1902941  0.6729069
## DLIC   -0.3185570 -0.68187354 -0.5219691  0.4013952
## INC    -0.1241055 -0.61747664  0.7603559 -0.1586799
## ROAD   -0.6154271  0.38360828  0.3364450  0.6007486

```

```
plot(fuel.pr1)
```

### fuel.pr1



```
sum(fuel.pr1$sdev^2)
```

```
## [1] 4
```

Note carefully that the values given at **Standard deviations**: are  $SD(\gamma_j^T \tilde{X})$ ; **NOT**  $SD(\tilde{X}_j)$ . We also verify that  $\sum \lambda_j = 4$ .

## 7.5 Data compression (Dimension reduction)

Let  $x_1, \dots, x_n$  be sampled from the  $q$ -dim. r.v.  $X \sim (m, \Sigma)$ , yielding a data matrix (or ‘data frame’)

$$\begin{pmatrix} x_1^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1q} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nq} \end{pmatrix} \in \mathbb{R}^{n \times q}.$$

Assume a PCA has been carried out as usual, yielding  $m, \Sigma, \gamma_1, \dots, \gamma_q, \lambda_1, \dots, \lambda_q$ . Suppose we wish to *compress* the data towards a smaller dimension  $d \leq q$  (for instance, in order to have fewer predictors or save storage space and computing time, etc.). How to choose  $d$ ? There are different conventions (all more or less heuristic) on how to extract such information from a scree plot (of  $\lambda_j$  versus  $j$ ), but the general idea is the following: The scree plot usually can be split into a left-hand part (which contains the components which capture significant variation in the data) and a right-hand part (which just contains noise or “scree”). So, one cuts off where this breakpoint, also called “knee”, is deemed to be situated.

How does the actual compression step work? *Data compression* means *projection*. We project all data points  $x_i$ ,  $i = 1, \dots, n$  onto the  $d$ -dim subspace spanned by the  $d$  largest principal components:

$$f : \mathbb{R}^q \longrightarrow \mathbb{R}^d, x_i \mapsto (\gamma_1, \dots, \gamma_d)^T (x_i - m), \quad i = 1, \dots, n$$

(the  $f(x_i) \equiv t_i$  are called *principal component scores*).

If desired, scores can be mapped back to the data space:

$$g : \mathbb{R}^d \longrightarrow \mathbb{R}^q, t_i \mapsto m + (\gamma_1, \dots, \gamma_d)t_i, \quad i = 1, \dots, n$$

which leads to ‘reconstructed’ values  $r_i \equiv (g \circ f)(x_i)$ . Obviously, the original data will not be exactly reconstructed (as we have dismissed some information), unless  $d = q$ . In the latter case, using the orthogonality of  $\Gamma$ ,

$$\begin{aligned} (g \circ f)(x_i) &= g(f(x_i)) = m + (\gamma_1, \dots, \gamma_d)(\gamma_1, \dots, \gamma_d)^T(x_i - m) \\ &= m + \Gamma\Gamma^T(x_i - m) = m + x_i - m = x_i \end{aligned}$$

The following R code implements the above procedure. We consider firstly just a single data point, and proceed with compression/reconstruction of the full `fuel` data set, each using  $d = 2$ , without scaling.

```
# Work firstly with a single observation. Say, case 1:
x1 <- fuel[1,]
mode(x1)           # wrong data type: lists can't be used for computation

## [1] "list"

x1<- as.numeric(x1)
mode(x1)           # that does it.

## [1] "numeric"

m <- colMeans(fuel)
t1 <- t(fuel.pr$rot[,c(1,2)]) %*% (x1-m)    # Score for case 1
r1 <- m+ fuel.pr$rot[,c(1,2)]%*% t1      # Reconstruction for case 1

x1

## [1]  9.000 52.500  3.571  1.976

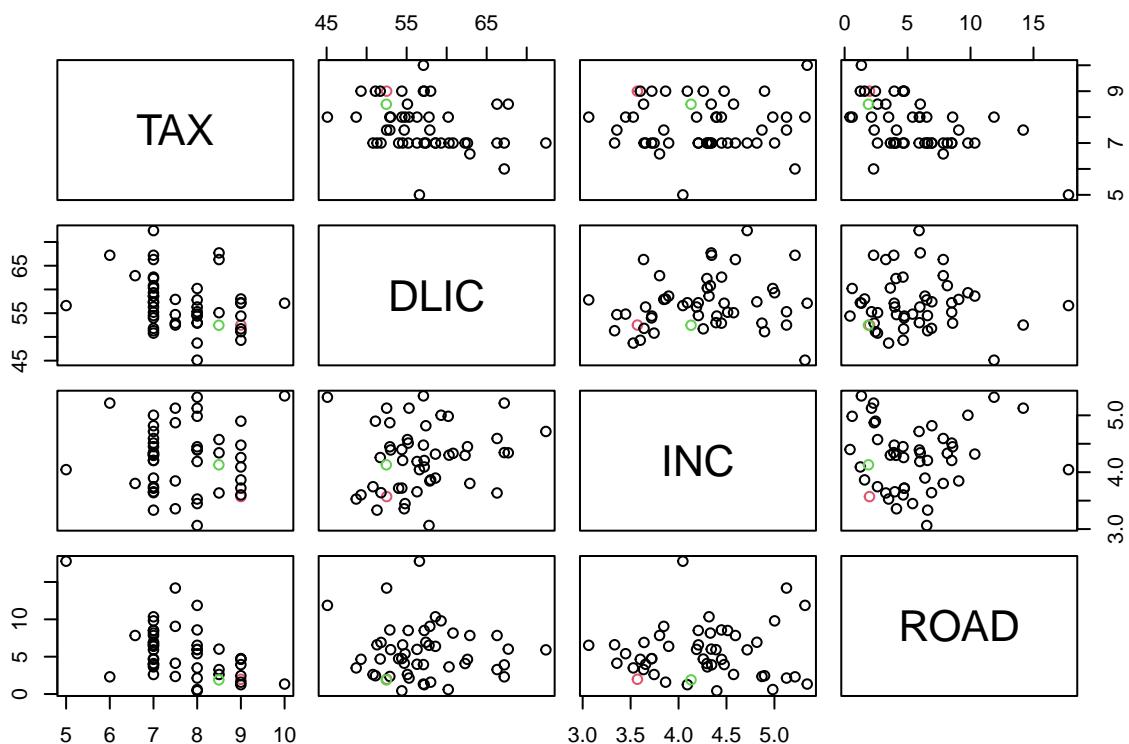
t1

##          [,1]
## PC1 -4.370997
## PC2  3.997192

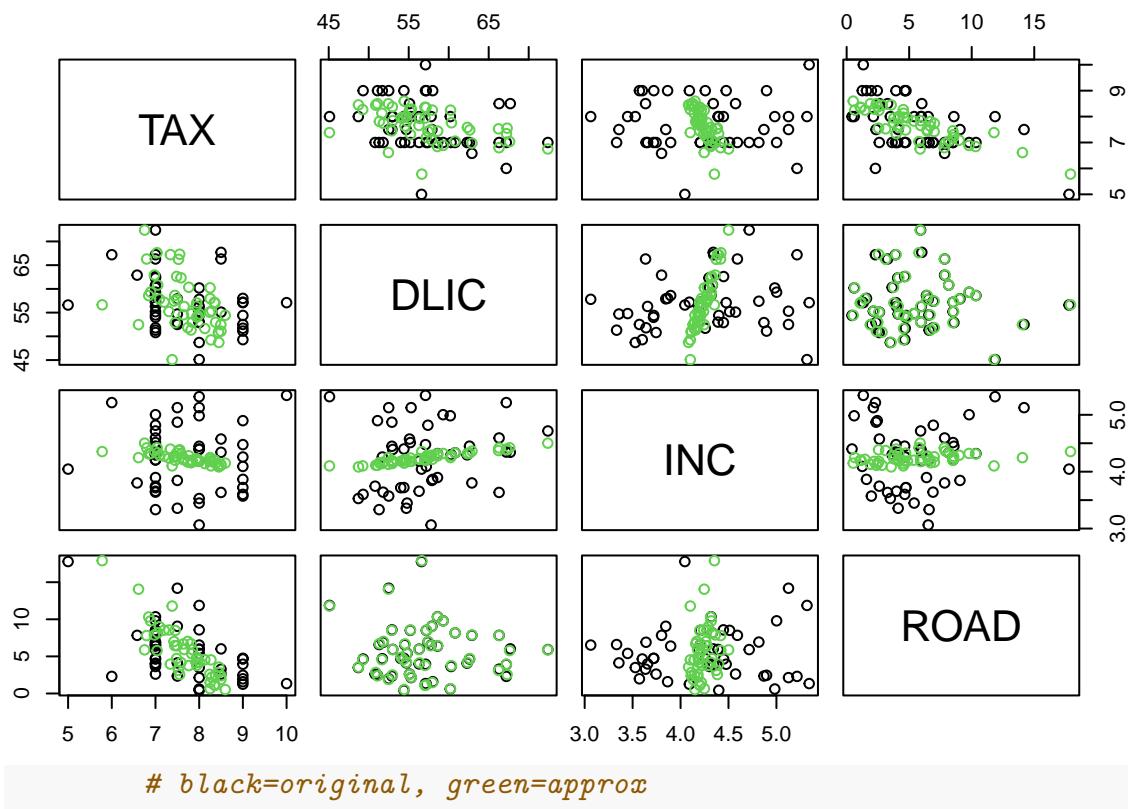
r1

##          [,1]
## TAX   8.495976
## DLIC  52.462122
## INC   4.130271
## ROAD  1.892577
```

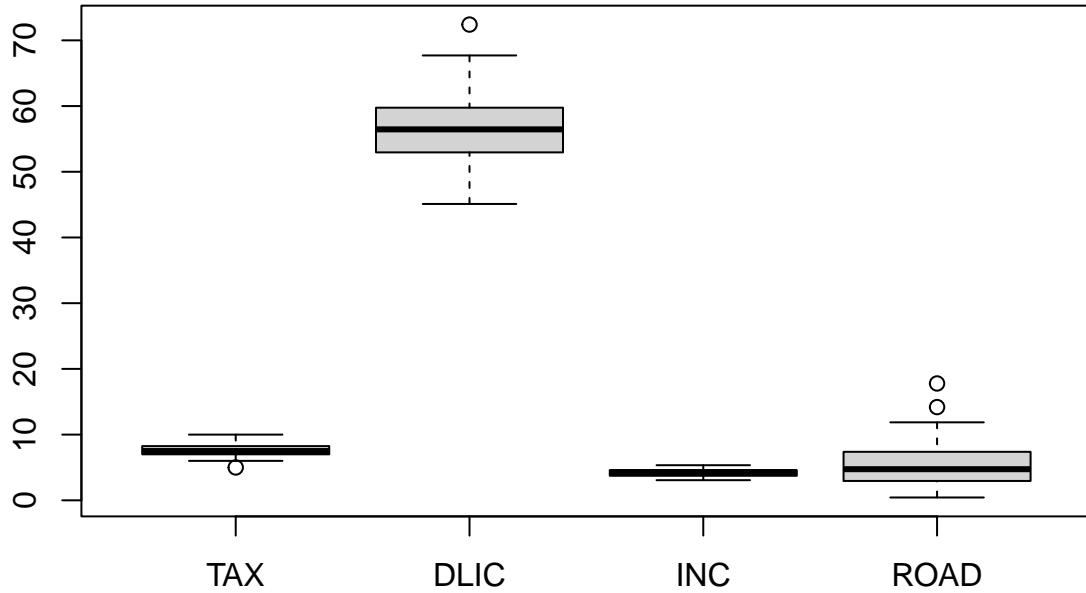
```
plot(rbind(fuel,t(r1)), col=c(2, rep(1,47),3))# red=original, green=approx
```



```
# For the full data matrix, we make use of the scores delivered by prcomp:  
  
# fuel.pr$x # is the same as:  
# t(fuel.pr$rot[,]) %*% (t(fuel)-m)  
  
T <- t(fuel.pr$x[,c(1,2)]) # All scores  
R <- t(m + fuel.pr$rot[,c(1,2)]%*% T) # All reconstructed values  
  
plot(rbind(fuel, R), col=c(rep(1,48),rep(3,48)))
```



```
boxplot(fuel)
```

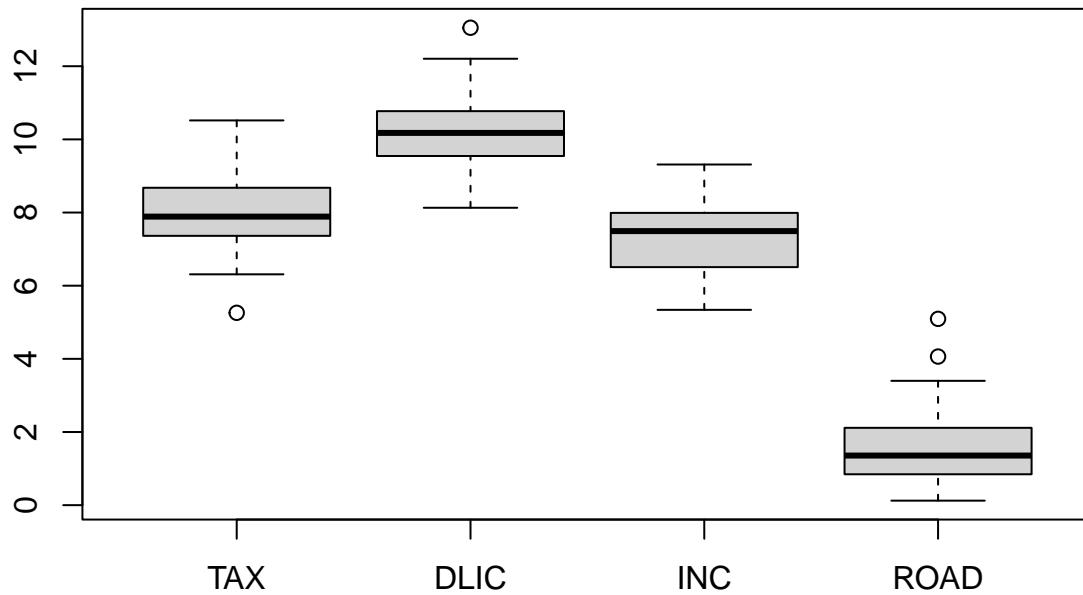


Obviously, PCA favors ROAD and DLIC which have the largest variance. Hence, when “compressing” the data we have essentially eliminated the “less variable” variables, TAX and INC. Note that the results would have changed dramatically if, for instance, INC would have been measured in \$ (instead of 1000\$).

To eliminate the effect of the units, we repeat the same analysis using scaled data.

```
s <- apply(fuel, 2, sd)      # calculates the column standard deviations
fuel.s <- sweep(fuel, 2, s, "/")  # divides all columns by their standard deviations
```

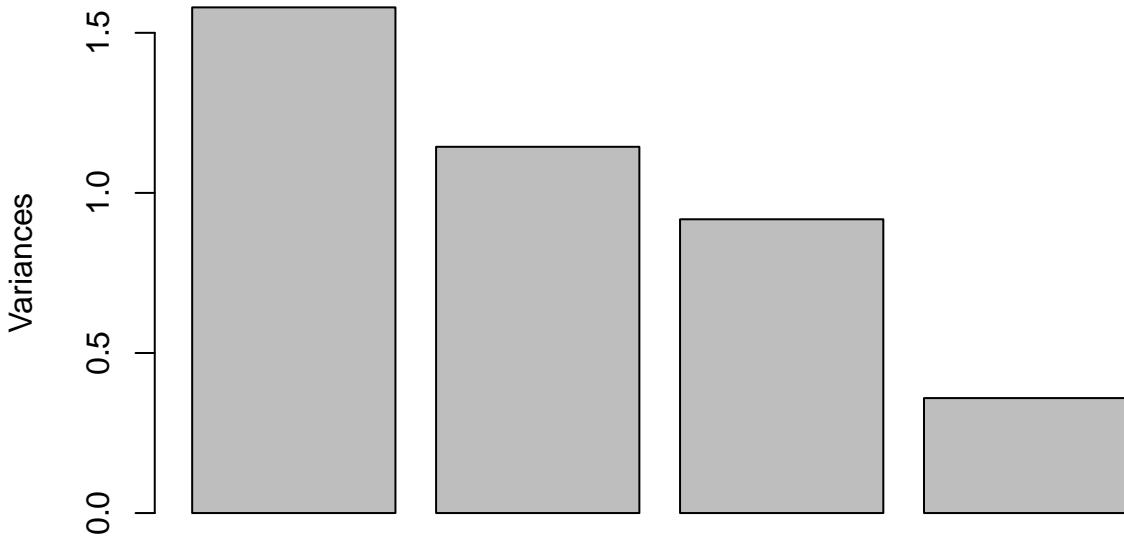
```
boxplot(fuel.s)
```



```
fuel.pr1 <- prcomp(fuel.s)
# (equivalent to fuel.pr1 in the example in previous section).
```

```
plot(fuel.pr1) # suggests d=3
```

**fuel.pr1**

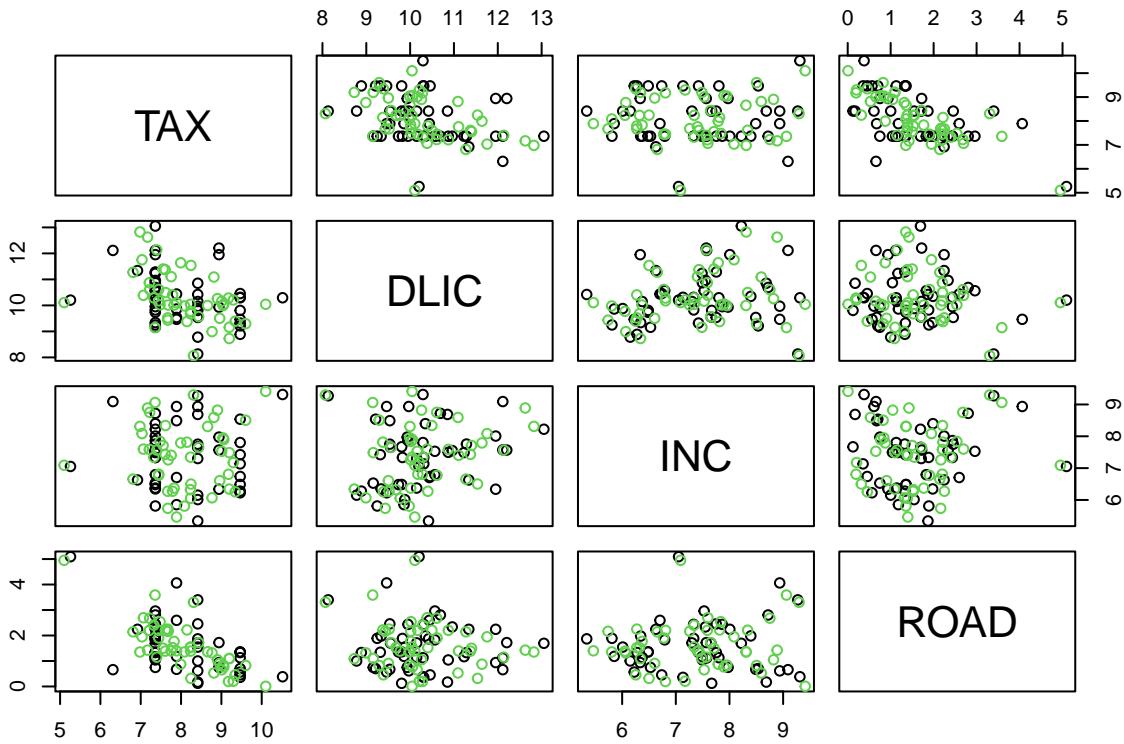


```
ms <- colMeans(fuel.s) # calculates means of scaled data set
```

```
Ts <- t(fuel.pr1$x[,c(1,2,3)]) # 3d-scores for scaled data
```

```
Rs <- t(ms + fuel.pr1$rot[,c(1,2,3)]%*% Ts) # Reconstructed scaled data
```

```
plot(rbind(fuel.s, Rs), col=c(rep(1,48),rep(3,48)))# black=original, green=approx
```



## 7.6 Principal Component Regression

### Principal Component Regression

Consider a linear regression problem with the model form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$

Assume that  $p$  is quite large, which may be impractical in practice. In such a case, may consider reducing the number of “predictors”, prior to regression, via PCA. That is instead of utilizing the  $p$  original predictors, this approach utilises  $d$  transformed variables (principal components) which are linear combinations of the original predictors, where  $d < p$ , then fit a regression model using the principal components. This technique, which combines the worlds of unsupervised and supervised learning, is known as *principal component regression* (PCR).

**Mathematically:**

1. Define  $d < p$  linear transformations  $Z_k, k = 1, \dots, d$  of  $X_1, X_2, \dots, X_p$  as

$$Z_k = \sum_{j=1}^p \phi_{jk} X_j,$$

when applying PCA,  $\phi_{1k}, \dots, \phi_{pk}$  is simply the  $\gamma_k$  obtained from eigen decomposition of the sample variance of data  $X$ .

2. Fit a linear regression model of the form

$$y_i = \theta_0 + \sum_{k=1}^d \theta_k z_{ik}, \quad i = 1, \dots, n.$$

We have **dimension reduction**: Instead of estimating  $p + 1$  coefficients we estimate  $d + 1$ ! Interestingly, with some re-ordering we get:

$$\sum_{k=1}^d \theta_k z_{ik} = \sum_{k=1}^d \theta_k \left( \sum_{j=1}^p \phi_{jk} x_{ij} \right) = \sum_{j=1}^p \left( \sum_{k=1}^d \theta_k \phi_{jk} \right) x_{ij} = \sum_{j=1}^p \beta_j x_{ij}$$

The key idea is that often a small number of principal components suffice to explain most of the variability of the data, as well as the relationship with the response. In other words, we *assume that the directions in which  $X_1, \dots, X_p$  show the most variation are the directions that are associated with  $Y$* . This assumption is not guaranteed, but it is often reasonable enough to give good results.

Another nice property of the new transformed variables (principal components) is that they are *uncorrelated* this would give a solution to the problem of *multicollinearity*.

## How Many Components?

- Once again we have a *tuning problem*.
- In ridge and lasso the tuning parameter ( $\lambda$ ) was continuous - in PCR the tuning parameter is the number of components ( $d$ ), which is discrete.
- Some methods calculate the total variance of  $X$  explained as we add further components. When the incremental increase is negligible, we stop. One such popular method is the *scree plot* (we have seen it in the previous session).
- Alternatively, we can use (yes, you guessed it...) cross-validation!

## What about Interpretability?

- One drawback of PCR is that it lacks interpretability, because the estimates  $\hat{\theta}_k$  for  $k = 1, \dots, d$  are the coefficients of the principal components, which have no meaningful interpretation.
- Well, that is partially true, but there is one thing we can do: once we fit LS on the transformed variables we can use the transformation to obtain the corresponding estimates of the original predictors

$$\hat{\beta}_j = \sum_{k=1}^d \hat{\theta}_k \phi_{jk},$$

for  $j = 1, \dots, p$

- However, for  $d < p$  these will not correspond to the LS estimates of the full model. In fact, the PCR estimates get *shrunk* in a discrete step-wise fashion as  $d$  decreases (this is why PCR is still a shrinkage method).

## PCR - a Summary

**Advantages:**

- Similar to ridge and lasso, it can result in improved predictive performance in comparison to Least Squares by resolving problems caused by multicollinearity and/or large  $p$ .
- It is a simple two-step procedure which first reduces dimensionality from  $p + 1$  to  $d + 1$  and then utilizes LS.

### Disadvantages:

- It does not perform feature selection.
- The issue of interpretability.
- It relies on the assumption that the directions in which the predictors show the *most variation* are the directions which are *predictive* of the response. This is not always the case. Sometimes, it is the last principal components which are associated with the response! Another dimension reduction method, *Partial Least Squares*, is more effective in such settings, but we do not have time in this course to cover this approach.

### Final Comments:

- In general, PCR will tend to do well in cases when the first few principal components are sufficient to capture most of the variation in the predictors as well as the relationship with the response.
- We note that even though PCR provides a simple way to perform regression using  $d < p$  predictors, it is *not* a feature selection method. This is because each of the  $d$  principal components used in the regression is a linear combination of all  $p$  of the original features.
- In general, dimension-reduction based regression methods are not very popular nowadays, mainly due to the fact that they do not offer any significant advantage over penalised regression methods.
- However, PCA is a very important tool in *unsupervised learning* where it is used extensively in a variety of applications!

## Practical Demonstration

For PCR, we will make use of the package `pls`, so we have to install this before first use by executing `install.packages('pls')`. The main command is called `pcr()`. The argument `scale=TRUE` standardises the columns of the predictor matrix, while argument `validation = 'CV'` picks the optimal number of principal components via cross-validation. The command `summary()` provides information on CV error and variance explained (RMSEP - Root Mean Squared Error (of Prediction)). Note that `adjCV` is an adjusted estimate of CV error accounting for bias. Under TRAINING, we can see the proportion of the variance of training data explained by the  $q$  largest PCs,  $q = 1, \dots, p$ .

```
library(pls) # for performing PCR

## 
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
## 
##     loadings
```

```

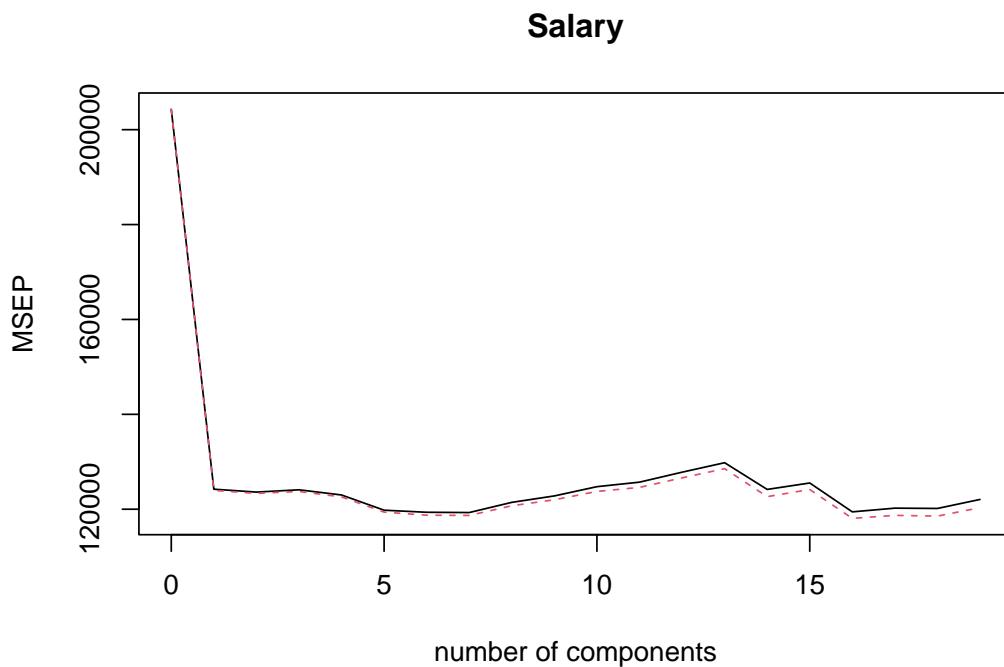
library(ISLR) # for Hitters dataset
set.seed(1)
pqr.fit = pqr( Salary ~ ., data = Hitters, scale = TRUE, validation = "CV" )
summary(pqr.fit)

## Data:      X dimension: 263 19
## Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##          (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
## CV          452     352.5   351.6   352.3   350.7   346.1   345.5
## adjCV       452     352.1   351.2   351.8   350.1   345.5   344.6
##          7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps
## CV          345.4   348.5   350.4   353.2   354.5   357.5   360.3
## adjCV       344.5   347.5   349.3   351.8   353.0   355.8   358.5
##          14 comps 15 comps 16 comps 17 comps 18 comps 19 comps
## CV          352.4   354.3   345.6   346.7   346.6   349.4
## adjCV       350.2   352.3   343.6   344.5   344.3   346.9
##
## TRAINING: % variance explained
##          1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps
## X          38.31   60.16   70.84   79.03   84.29   88.63   92.26   94.96
## Salary     40.63   41.58   42.17   43.22   44.90   46.48   46.69   46.75
##          9 comps 10 comps 11 comps 12 comps 13 comps 14 comps 15 comps
## X          96.28   97.26   97.98   98.65   99.15   99.47   99.75
## Salary     46.86   47.76   47.82   47.85   48.10   50.40   50.55
##          16 comps 17 comps 18 comps 19 comps
## X          99.89   99.97   99.99   100.00
## Salary     53.01   53.85   54.61   54.61

```

We can use the command `validationplot()` to plot the validation error. The argument `val.type = 'MSEP'` specifies to select Mean Squared Error (of Prediction), other options are Root Mean Squared Error (of Prediction), RMSEP and  $R^2$ ; type `?validationplot` for help.

```
validationplot( pqr.fit, val.type = 'MSEP' )
```



From the plot above it is not obvious which PCR model minimises the CV error. We would like to extract this information automatically, but it turns out that is not so easy with `pcr()`! One way to do this is via the code below (try to understand what it does!).

```
min.pcr = which.min( MSEP( pcr.fit )$val[1,1,] ) - 1
min.pcr
```

```
## 7 comps
##      7
```

So, it is the model with 7 principal components that minimises CV. Now that we know which model it is we can find the corresponding estimates in terms of the original  $\hat{\beta}$ 's using the command `coef()` we have used before. Also we can generate predictions via `predict()` (below the first 6 predictions are displayed).

```
coef(pcr.fit, ncomp = min.pcr)
```

```
## , , 7 comps
##
##          Salary
## AtBat     27.005477
## Hits      28.531195
## HmRun     4.031036
## Runs      29.464202
## RBI       18.974255
## Walks     47.658639
## Years     24.125975
## CAtBat   30.831690
## CHits    32.111585
## CHmRun   21.811584
## CRuns    34.054133
```

```

## CRBI      28.901388
## CWalks    37.990794
## LeagueN   9.021954
## DivisionW -66.069150
## PutOuts   74.483241
## Assists   -3.654576
## Errors    -6.004836
## NewLeagueN 11.401041

head( predict( pcr.fit, ncomp = min.pcr ) )

## , , 7 comps
##
##                               Salary
## -Alan Ashby      568.7940
## -Alvin Davis     670.3840
## -Andre Dawson    921.6077
## -Andres Galarraga 495.8124
## -Alfredo Griffin  560.3198
## -Al Newman       135.5378

```

## Extension: Regularisation Paths for PCR

What about regularisation paths with `pcr()`? This is again tricky because there is no built-in command. The below piece of code can be used to produce regularisation paths. The regression coefficients from all models using 1 up to 19 principal components are stored in `pcr.fit$coefficients`, but this is an object of class `list` which is not very convenient to work with. Therefore, we create a matrix called `coef.mat` which has 19 rows and 19 columns. We then store the coefficients in `pcr.fit$coefficients` to the columns of `coef.mat`, starting from the model that has 1 principal component, using a `for` loop. We then use `plot()` to plot the path of the first row of `coef.mat`, followed by a `for` loop calling the command `lines()` to superimpose the paths of the rest of the rows of `coef.mat`.

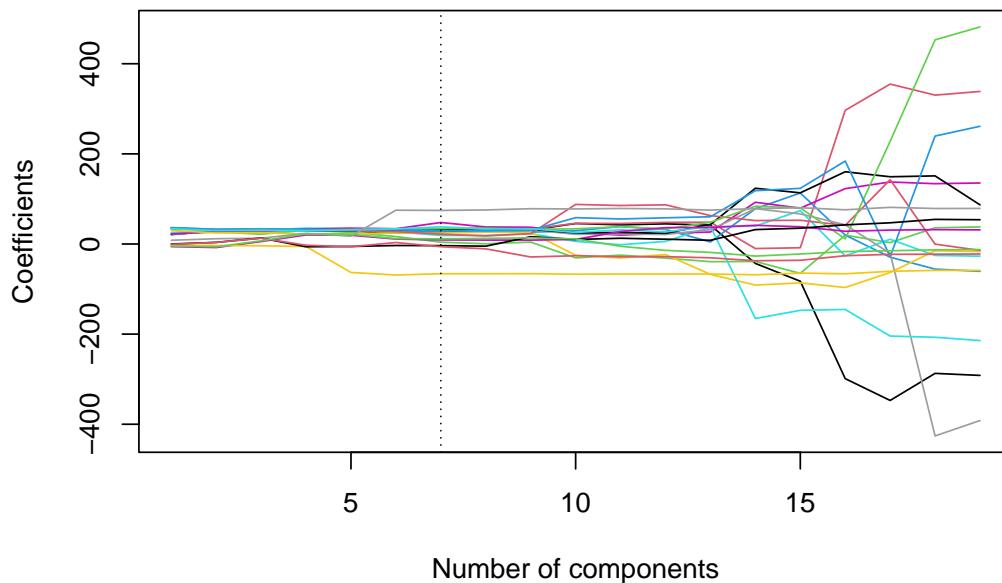
```

coef.mat = matrix(NA, 19, 19)
for(i in 1:19){
  coef.mat[,i] = pcr.fit$coefficients[,,i]
}

plot(coef.mat[1,], type = 'l', ylab = 'Coefficients',
      xlab = 'Number of components', ylim = c(min(coef.mat), max(coef.mat)))
for(i in 2:19){
  lines(coef.mat[i,], col = i)
}

abline(v = min.pcr, lty = 3)

```

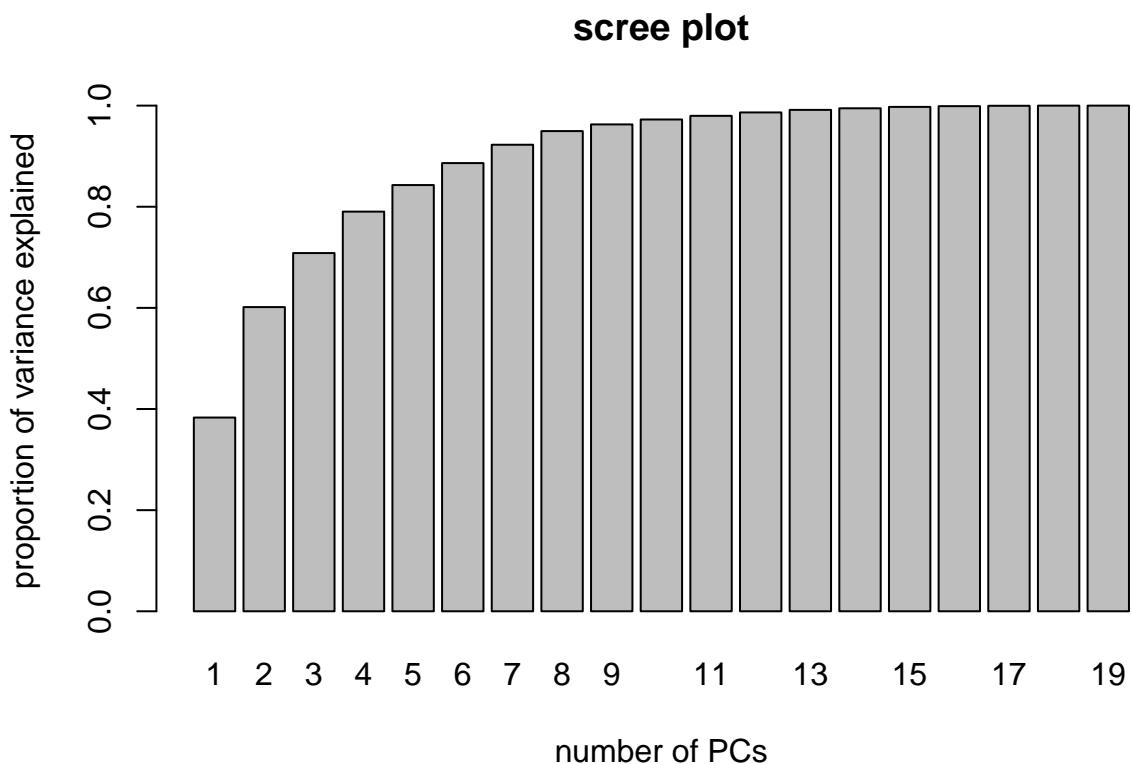


We can see from the plot the somewhat strange way in which PCR regularises the coefficients.

## Scree plots

Finally, consider here that we used the in-built CV within the function `pqr` to select the number of coefficients. We can, however, use the visual aid of a scree plot to help us decide on the appropriate number of principal components. This amounts to plotting as a bar chart the cumulative proportion of variance attributed to the first  $q$  principal components, such as was shown in the X row of the `% variance explained` matrix when we called `summary(pqr.fit)`. This information is a little difficult to extract from the `summary` object, however, so we construct the scree plot manually as shown below (try to understand what is happening):

```
PVE <- rep(NA, 19)
for(i in 1:19){ PVE[i] <- sum(pqr.fit$Xvar[1:i])/pqr.fit$Xtotvar }
barplot( PVE, names.arg = 1:19, main = "scree plot",
         xlab = "number of PCs",
         ylab = "proportion of variance explained" )
```



We may, for example, decide that we need the number of principal components required to explain 95% of the variance in the predictors. In this case, we could look at the scree plot and the output from `summary(pcr.fit)` to elicit that we need to select the first 9 principal components in order to explain 95% of the variance in  $\mathbf{X}$ .

# Chapter 8

## Polynomial Regression

### 8.1 The Assumption of Linearity

- So far, we have worked under the assumption that the relationships between predictors and the response are (first-order) linear.
- In reality, they are almost never exactly linear.
- However, as long as the relationships are *approximately linear*, linear models work fine.
- But what can we do when the relationships are clearly non-linear??

#### Example

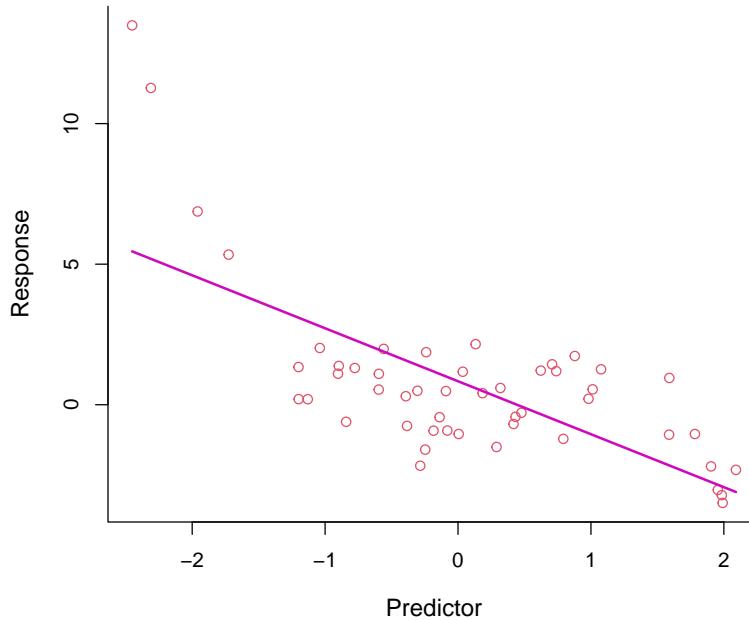


Figure 8.1: First-order polynomial regression line through non-linear data.

Figure 8.1 shows a fitted regression line from the model  $y = \beta_0 + \beta_1 x + \epsilon$ : clearly not a good fit!

Why not add  $x^2$  in the equation in order to bend the straight line? See Figure 8.2.

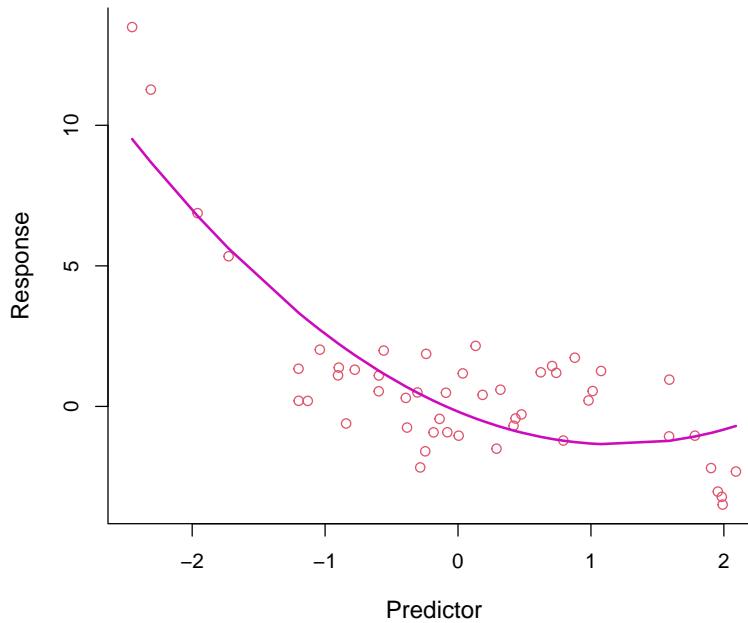


Figure 8.2: Second-order polynomial regression line through non-linear data.

The fitted regression line from the model  $y = \beta_0 + \beta_1x + \beta_2x^2 + \epsilon$  looks somehow better, especially on the left tail.

Why not make the line even more flexible by adding an  $x^3$ -term? see Figure 8.3.

The fitted regression line from model  $y = \beta_0 + \beta_1x + \beta_2x^2 + \beta_3x^3 + \epsilon$  looks just right.

## 8.2 Polynomial Regression Models

- We have just implemented *polynomial regression* - as easy as that!
- In general, polynomial models are of the form

$$y = f(x) = \beta_0 + \beta_1x + \beta_2x^2 + \beta_3x^3 + \dots + \beta_dx^d + \epsilon,$$

where  $d$  is called the *degree* of the polynomial.

- Notice how convenient this is: the non-linear relationship between  $y$  and  $x$  is captured by the polynomial terms *but* the models **remain linear** in the parameters/coefficients (the models are additive).
- This means we can use standard Least Squares for estimation!
- For example, previously we fitted a third-order polynomial, which can be also written as

$$y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_3 + \epsilon$$

with the predictors simply being  $X_1 = X$ ,  $X_2 = X^2$  and  $X_3 = X^3$ .

### What degree $d$ should I choose?

- What about  $d$  here - do we have a tuning-problem again?
- In this case not really. In practice, values of  $d$  greater than 3 or 4 are rarely used.

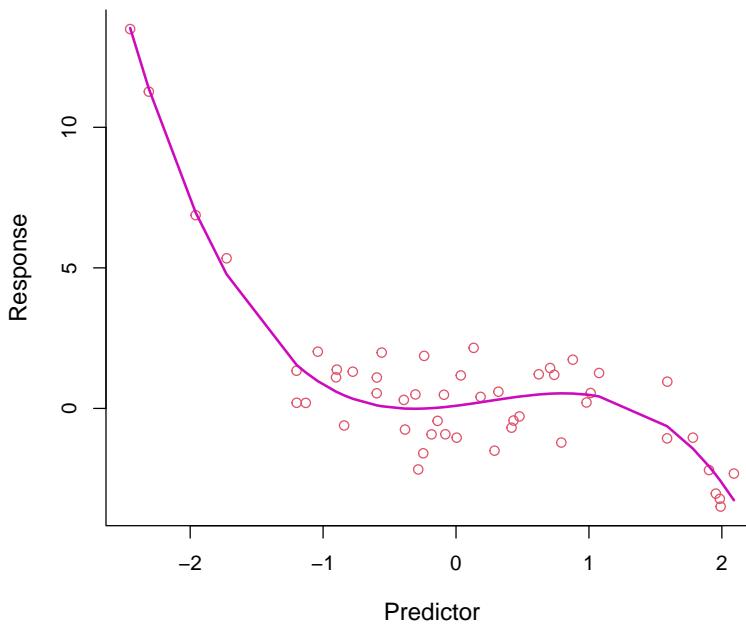


Figure 8.3: Third-order polynomial regression line through non-linear data.

- This is because if  $d$  is large the fitted curve becomes overly flexible/complex and can take very strange shapes.
- We want the curve to pass through the data in a *smooth* way.

### Example: Overfitting

We fit a polynomial of degree 3 to some training data. On the left of Figure 8.4, we compare the training data (red points) with their fitted values (that is, the model predictions at the training data inputs) joined up as a line. On the right, we compare some test data with their model predicted values (again, joined up as a line). We can see the learned curve fits relatively nicely on both the training data and the test data (it is very similar in both plots, noting that the fit is only plotted across the range of test points in the right-hand plot).

We then fit a polynomial of degree 14 to the same training data. On the left of Figure 8.5, we can see that the fitted curve tries to pass through every training data point. The result of this overfitting on prediction is clearly seen on the right.

### Example: Wage data

We consider the `Wage` dataset from library `ISLR`, which contains income and demographic information for males who reside in the central Atlantic region of the United States. In particular, each panel of Figure 8.6 is a plot of `wage` against `age`. We then see the results of plotting polynomial models  $\hat{f}(x)$  (solid lines) of `age` for `wage` of various degrees - 2, 3 and 4 respectively. We can see some slight differences between the models for this data.

The purpose of fitting any of these polynomial models is because we are interested in modelling `wage` as a function  $\hat{f}$  of the predictor `age` as best we can, this leading to an understanding of the relationship between `age` and `wage`. For example, in the left-hand panel we have a degree-2 polynomial fit  $\hat{f}$ . This would allow us to estimate wage at a particular value for

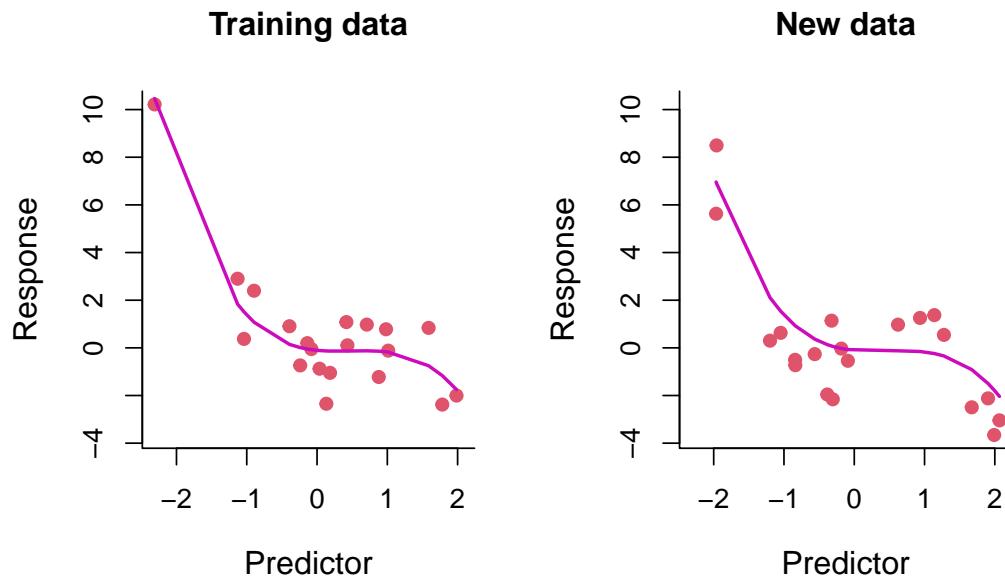


Figure 8.4: Degree-3 polynomial regression line through training and test data.

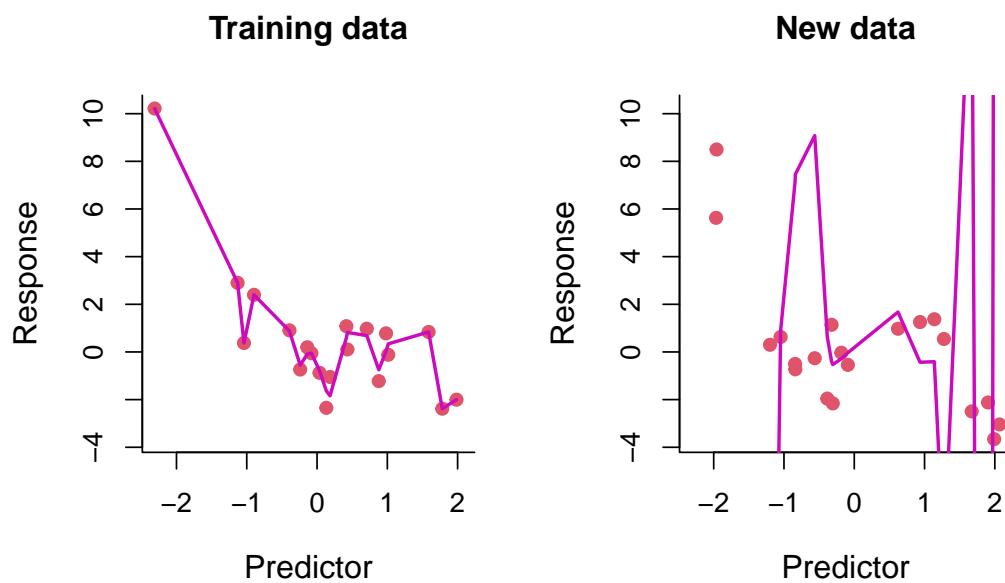


Figure 8.5: Degree-14 polynomial regression line through training and test data.

age,  $x_0$  say, as

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2$$

where  $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$  are the least squares fitted coefficients. Least squares also returns variance estimates for each of the fitted coefficients  $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ , as well as the covariances between pairs of coefficient estimates. We can use these to estimate the *point-wise* standard error of the (mean) prediction  $\hat{f}(x_0)$ .

The coloured points on either side of the solid-line fitted curve are therefore  $2 \times$  standard error curves, that is they correspond to  $\hat{f}(x) \pm 2 \times \text{SE}[\hat{f}(x)]$ , where  $\text{SE}[\hat{f}(x)]$  corresponds to the standard error on the mean prediction at  $x$ . We plot twice the standard error because, for normally distributed error terms, this quantity corresponds to an approximate 95% confidence interval (in this case for the mean prediction at  $x$ ).

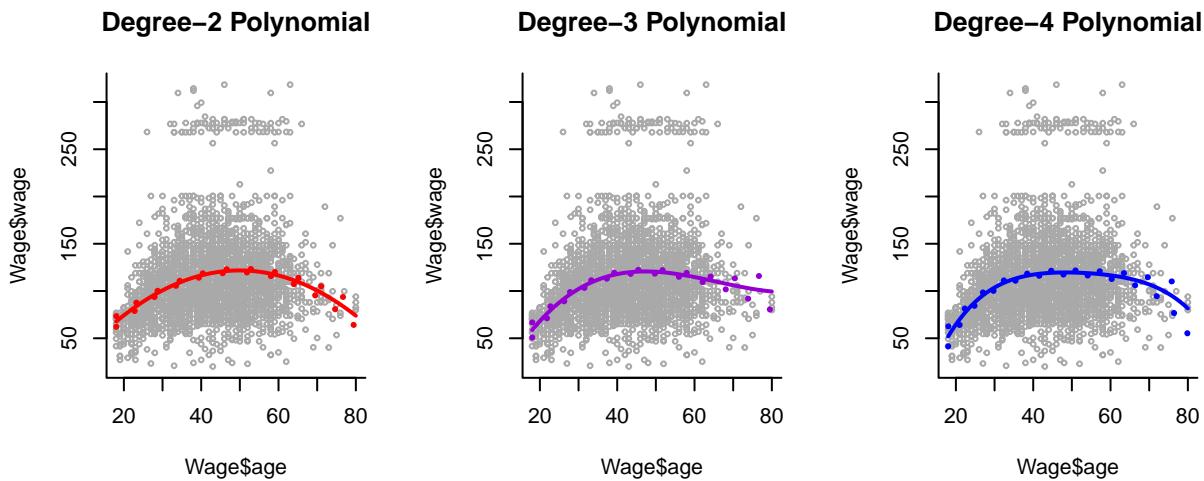


Figure 8.6: Plots of ‘wage’ against ‘age’ and the fits from three polynomial models.

## Disadvantages

- The fitted curve from polynomial regression is obtained by *global* training. That is, we use the entire range of values of the predictor to fit the curve.
- This can be problematic: if we get new samples from a specific subregion of the predictor this might change the shape of the curve in other subregions!
- Ideally, we would like the curve to adapt *locally* within subregions which are relevant to the analysis.
- A first simple approach for resolving this is via step functions (see Chapter 9).

## 8.3 Practical Demonstration

In this section we will start with the analysis of the `Boston` dataset included in the R library `MASS`. This dataset consists of 506 samples. The response variable is median value of owner-occupied homes in Boston (`medv`). The dataset has 13 associated predictor variables.

## Initial steps

We load library MASS (installing the package is not needed for MASS), check the help document for Boston, check for missing entries and use commands `names()` and `head()` to get a better picture of how this dataset looks like.

```
library(MASS)
help(Boston)
sum(is.na(Boston))

## [1] 0

names(Boston)

## [1] "crim"      "zn"        "indus"     "chas"      "nox"       "rm"        "age"
## [8] "dis"        "rad"       "tax"        "ptratio"   "black"     "lstat"     "medv"

head(Boston)

##      crim zn indus chas nox rm age dis rad tax ptratio black lstat
## 1 0.00632 18 2.31 0 0.538 6.575 65.2 4.0900 1 296 15.3 396.90 4.98
## 2 0.02731 0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8 396.90 9.14
## 3 0.02729 0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8 392.83 4.03
## 4 0.03237 0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7 394.63 2.94
## 5 0.06905 0 2.18 0 0.458 7.147 54.2 6.0622 3 222 18.7 396.90 5.33
## 6 0.02985 0 2.18 0 0.458 6.430 58.7 6.0622 3 222 18.7 394.12 5.21
##      medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

We will analyse `medv` with respect to the predictor `lstat` (percentage of lower status population).

```
head(cbind(Boston$medv, Boston$lstat))
```

```
##      [,1] [,2]
## [1,] 24.0 4.98
## [2,] 21.6 9.14
## [3,] 34.7 4.03
## [4,] 33.4 2.94
## [5,] 36.2 5.33
## [6,] 28.7 5.21
```

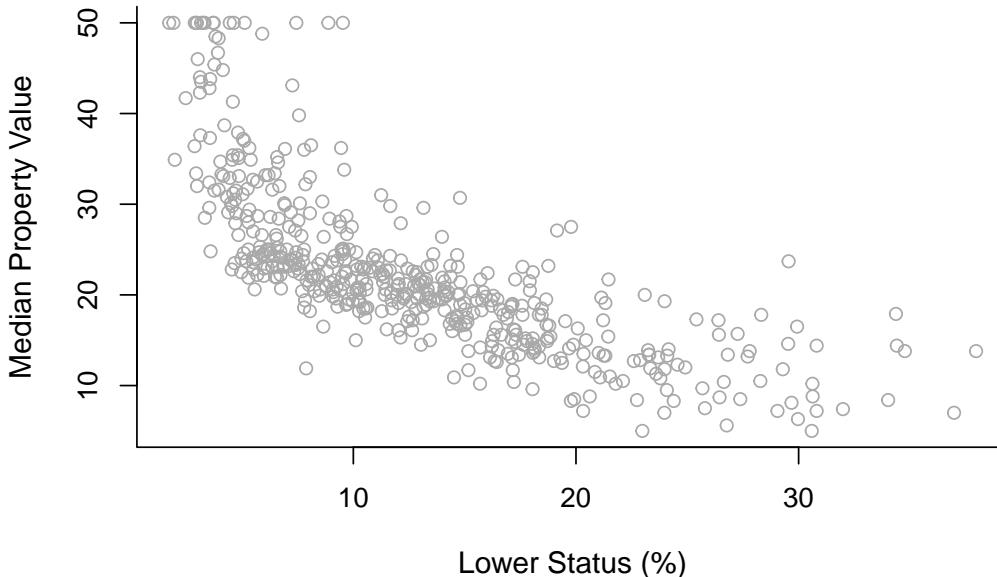
For convenience we will just name the response `y` and the predictor `x`. We will also pre-define the labels for the x and y-axes that we will use repeatedly in figures throughout this practical.

```
y = Boston$medv
x = Boston$lstat
```

```
y.lab = 'Median Property Value'
x.lab = 'Lower Status (%)'
```

First, lets plot the two variables.

```
plot( x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "", bty = 'l' )
```



The plot suggests a non-linear relationship between `medv` and `lstat`.

## Polynomial regression

Let us start by fitting to the data a degree-2 polynomial using the command `lm()` and summarising the results using `summary()`.

```
poly2 = lm( y ~ x + I(x^2) )
summary( poly2 )
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept) 42.862007   0.872084  49.15 <0.0000000000000002 ***
## x           -2.332821   0.123803 -18.84 <0.0000000000000002 ***
## I(x^2)       0.043547   0.003745  11.63 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 0.00000000000000022
```

**Important note:** Above we see that we need to enclose  $x^2$  within the envelope function  $I()$ . This is because  $x^2$  is a function of  $x$  and when we use a function (any function) of a predictor in  $lm()$  we need to do that, otherwise we get wrong results! Besides that we see the usual output from `summary()`.

The above syntax is not very convenient because we need to add manually further polynomial terms. For instance, for a 4-degree polynomial we would need to use  $lm(y \sim x + I(x^2) + I(x^3) + I(x^4))$ . Fortunately, we have the function `poly()` which makes things easier for us.

```
poly2 = lm(y ~ poly(x, 2, raw = TRUE))
summary(poly2)
```

```
##
## Call:
## lm(formula = y ~ poly(x, 2, raw = TRUE))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
##
## Coefficients:
##                               Estimate Std. Error t value     Pr(>|t|)
## (Intercept)             42.862007  0.872084  49.15 <0.0000000000000002 ***
## poly(x, 2, raw = TRUE)1 -2.332821  0.123803 -18.84 <0.0000000000000002 ***
## poly(x, 2, raw = TRUE)2  0.043547  0.003745  11.63 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 0.00000000000000022
```

The argument `raw = TRUE` above is used in order to get the same coefficients as previously. If we do not use this argument the coefficients will differ because they will be calculated on a different (orthogonal) basis. However, this may not be that important because with polynomial regression it is often the case that we are not interested in the regression coefficients as the model becomes more complex. In terms of fitting the curve `poly(x, 2, raw = TRUE)` and `poly(x, 2)` will give the same result!

Now, lets see how we can produce a plot similar to those shown in the Wage data example. A first important step is that we need to create an object, which we name `sort.x`, which has the *sorted values* of predictor  $x$  in a *ascending order*. Without `sort.x` we will not be able to produce the plots! Then, we need to use `predict()` with `sort.x` as input in order

to proceed to the next steps.

```
sort.x = sort(x)
sort.x[1:10]      # the first 10 sorted values of x

## [1] 1.73 1.92 1.98 2.47 2.87 2.88 2.94 2.96 2.97 2.98

pred2 = predict(poly2, newdata = list(x = sort.x), se = TRUE)
names(pred2)

## [1] "fit"           "se.fit"        "df"            "residual.scale"
```

The object `pred2` contains `fit`, which are the *fitted values*, and `se.fit`, which are the *standard errors* of the mean prediction, that we need in order to construct the approximate 95% confidence intervals (of the mean prediction). With this information we can construct the confidence intervals using `cbind()`. Lets see how the first 10 fitted values and confidence intervals look like.

```
pred2$fit[1:10]    # the first 10 fitted values of the curve

##       1       2       3       4       5       6       7       8
## 38.95656 38.54352 38.41374 37.36561 36.52550 36.50468 36.37992 36.33840
##       9      10
## 36.31765 36.29691

se.bands2 = cbind( pred2$fit - 2 * pred2$se.fit,
                  pred2$fit + 2 * pred2$se.fit )
se.bands2[1:10,] # the first 10 confidence intervals of the curve

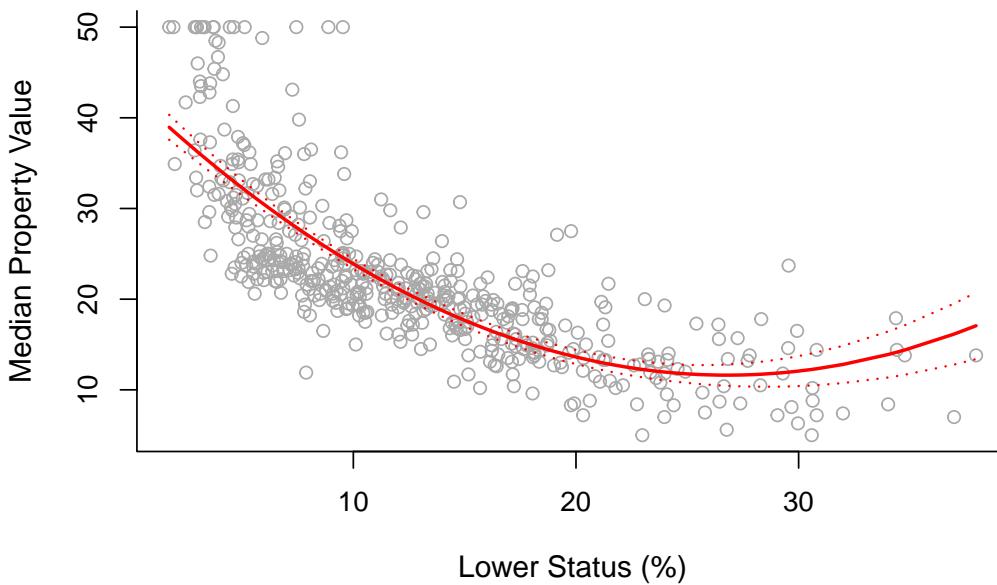
##      [,1]     [,2]
## 1 37.58243 40.33069
## 2 37.20668 39.88036
## 3 37.08853 39.73895
## 4 36.13278 38.59845
## 5 35.36453 37.68647
## 6 35.34546 37.66390
## 7 35.23118 37.52865
## 8 35.19314 37.48365
## 9 35.17413 37.46117
## 10 35.15513 37.43870
```

Now that we have all that is needed we can finally produce the plot!

**Final detail:** Below we use `lines()` for `pred2$fit` because this is a *vector*, but for `se.bands2`, which is a *matrix*, we have to use `matlines()`.

```
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Degree-2 polynomial", bty = 'l')
lines(sort.x, pred2$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands2, lwd = 1.4, col = "red", lty = 3)
```

### Degree-2 polynomial



Having seen this procedure in detail once, plotting the fitted curves for polynomials of different degrees is straightforward. The code below produces a plot of degree-2 up to degree-5 polynomial fits.

```

poly3 = lm(y ~ poly(x, 3))
poly4 = lm(y ~ poly(x, 4))
poly5 = lm(y ~ poly(x, 5))

pred3 = predict(poly3, newdata = list(x = sort.x), se = TRUE)
pred4 = predict(poly4, newdata = list(x = sort.x), se = TRUE)
pred5 = predict(poly5, newdata = list(x = sort.x), se = TRUE)

se.bands3 = cbind(pred3$fit + 2*pred3$se.fit, pred3$fit - 2*pred3$se.fit)
se.bands4 = cbind(pred4$fit + 2*pred4$se.fit, pred4$fit - 2*pred4$se.fit)
se.bands5 = cbind(pred5$fit + 2*pred5$se.fit, pred5$fit - 2*pred5$se.fit)

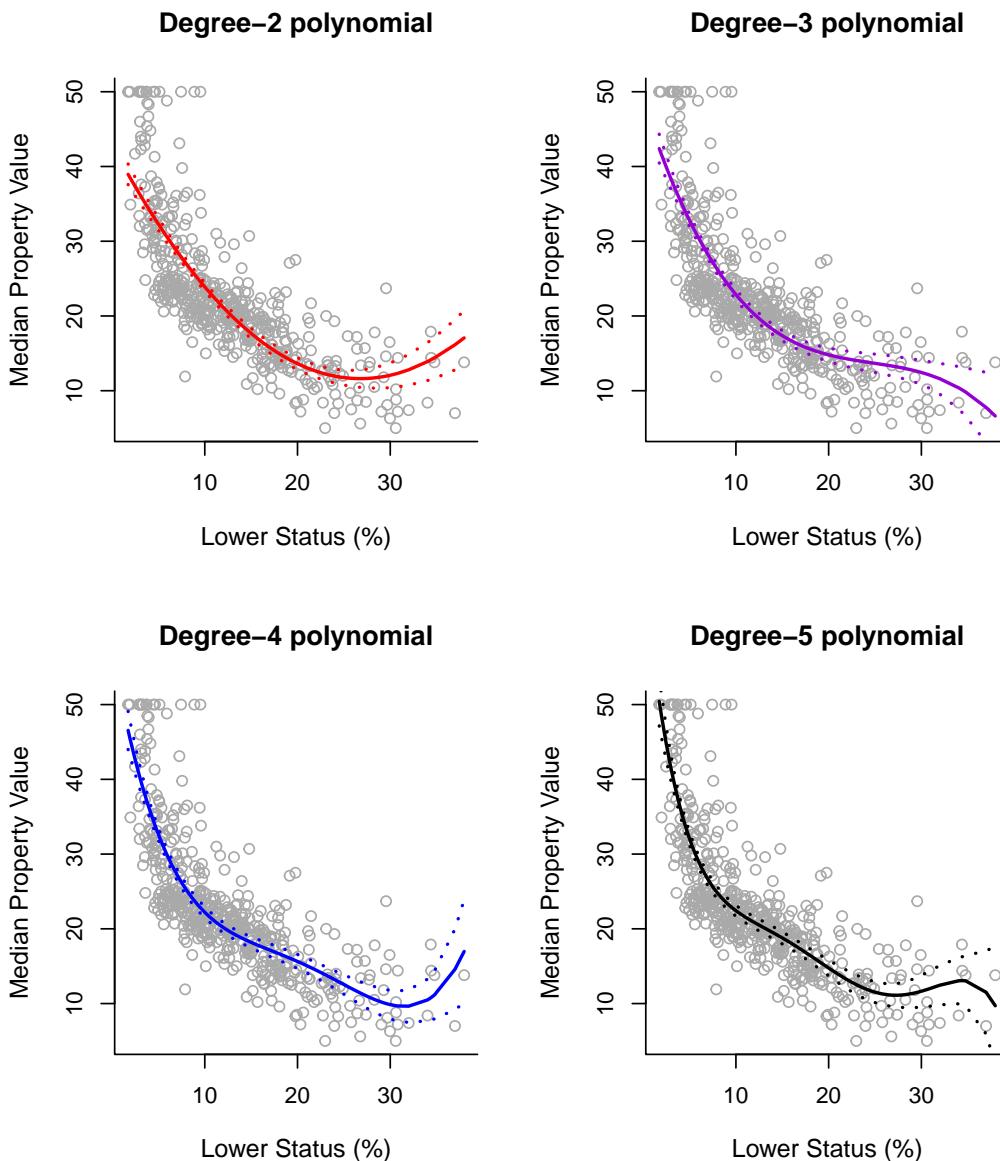
par(mfrow = c(2,2))
# Degree-2
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Degree-2 polynomial", bty = 'l')
lines(sort.x, pred2$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands2, lwd = 2, col = "red", lty = 3)

# Degree-3
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Degree-3 polynomial", bty = 'l')
lines(sort.x, pred3$fit, lwd = 2, col = "darkviolet")
matlines(sort.x, se.bands3, lwd = 2, col = "darkviolet", lty = 3)

```

```
# Degree-4
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Degree-4 polynomial", bty = 'l')
lines(sort.x, pred4$fit, lwd = 2, col = "blue")
matlines(sort.x, se.bands4, lwd = 2, col = "blue", lty = 3)

# Degree-5
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Degree-5 polynomial", bty = 'l')
lines(sort.x, pred5$fit, lwd = 2, col = "black")
matlines(sort.x, se.bands5, lwd = 2, col = "black", lty = 3)
```



For this data it is not clear which curve fits better. All four curves look reasonable given the data that we have available. Without further indications, one may choose the degree-2 polynomial since it is simpler and seems to do about as well as the others. However, if we

want to base our decision on a more formal procedure, rather than on a subjective decision, one option is to use the classical statistical methodology of analysis-of-variance (ANOVA). Specifically, we will perform sequential comparisons based on the F-test, comparing first the linear model vs. the quadratic model (degree-2 polynomial), then the quadratic model vs. the cubic model (degree-3 polynomial) and so on. We therefore have to fit the simple linear model, and we also choose to fit the degree-6 polynomial to investigate the effects of an additional predictor as well. We can perform this analysis in RStudio using the command `anova()` as displayed below.

```
poly1 = lm(y ~ x)
poly6 = lm(y ~ poly(x, 6))
anova(poly1, poly2, poly3, poly4, poly5, poly6)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ x
## Model 2: y ~ poly(x, 2, raw = TRUE)
## Model 3: y ~ poly(x, 3)
## Model 4: y ~ poly(x, 4)
## Model 5: y ~ poly(x, 5)
## Model 6: y ~ poly(x, 6)
##   Res.Df   RSS Df Sum of Sq      F      Pr(>F)
## 1    504 19472
## 2    503 15347  1    4125.1 151.8623 < 0.0000000000000022 ***
## 3    502 14616  1     731.8  26.9390      0.0000003061 ***
## 4    501 13968  1     647.8  23.8477      0.0000014062 ***
## 5    500 13597  1     370.7  13.6453      0.0002452 ***
## 6    499 13555  1      42.4   1.5596      0.2123125
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The hypothesis that is checked at each step is that the *decrease in RSS is not significant*. The hypothesis is rejected if the *p-value* (column `Pr(>F)`) is smaller than a given *significance level* (say 0.05). If the hypothesis is rejected then we move on to the next comparison. Based on this reasoning and the results above we would select the 5-degree polynomial.

Note that alternatively, we can always use (yes, you guessed it...) cross-validation to decide the degree of the polynomial model!

We could also try polynomial regression with multiple predictors. For example, include average number of rooms per dwelling `rm` as the second predictor. Note that in this case we will also including the interaction terms  $x_1x_2$ .

```
x1=Boston$lstat
x2=Boston$rm

polym1 <- lm(y ~ poly(x1, 2) + poly(x2 , 2) + x1:x2)
summary(polym1)
```

```

## 
## Call:
## lm(formula = y ~ poly(x1, 2) + poly(x2, 2) + x1:x2)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -28.9520  -2.5690  -0.4172   2.0432  27.6293 
## 
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)    
## (Intercept) 40.20765  4.91368  8.183 0.00000000000002315 ***
## poly(x1, 2)1 106.09456  61.94682  1.713 0.08739 .  
## poly(x1, 2)2  13.47448   7.30579  1.844 0.06572 .  
## poly(x2, 2)1 108.16782  12.75828  8.478 0.0000000000000259 ***
## poly(x2, 2)2  36.83909   6.27520  5.871 0.000000007919304425 ***
## x1:x2        -0.23121   0.06422 -3.600 0.00035 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 4.554 on 500 degrees of freedom
## Multiple R-squared:  0.7573, Adjusted R-squared:  0.7548 
## F-statistic: 312 on 5 and 500 DF,  p-value: < 0.0000000000000022 

# we could use polym() makes things easier for us
polym2=lm(y ~ polym(x1, x2, degree=2) )
summary(polym2)

```

```

## 
## Call:
## lm(formula = y ~ polym(x1, x2, degree = 2))
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -28.9520  -2.5690  -0.4172   2.0432  27.6293 
## 
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)    
## (Intercept)      21.8222    0.2828  77.178 < 0.000000000000002 
## polym(x1, x2, degree = 2)1.0 -127.0837   6.6032 -19.246 < 0.000000000000002 
## polym(x1, x2, degree = 2)2.0   13.4745   7.3058  1.844 0.06572 
## polym(x1, x2, degree = 2)0.1   61.9766   6.0811 10.192 < 0.000000000000002 
## polym(x1, x2, degree = 2)1.1  -585.8312  162.7253 -3.600 0.00035 
## polym(x1, x2, degree = 2)0.2   36.8391   6.2752  5.871 0.00000000792 
## 
## (Intercept)      ***
## polym(x1, x2, degree = 2)1.0 ***
## polym(x1, x2, degree = 2)2.0 .

```

```
## polym(x1, x2, degree = 2)0.1 ***
## polym(x1, x2, degree = 2)1.1 ***
## polym(x1, x2, degree = 2)0.2 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.554 on 500 degrees of freedom
## Multiple R-squared:  0.7573, Adjusted R-squared:  0.7548
## F-statistic:    312 on 5 and 500 DF,  p-value: < 0.0000000000000022
```

# Chapter 9

## Step Functions

### 9.1 Step Functions

Step functions use *cut-points*  $c_1, c_2, \dots, c_K$  in the range of  $X$  in order to construct  $K + 1$  *dummy variables* in the following way:

$$\begin{aligned} C_0(X) &= I(X \leq c_1) \\ C_1(X) &= I(c_1 < X \leq c_2) \\ &\vdots \\ C_{K-1}(X) &= I(c_{K-1} < X \leq c_K) \\ C_K(X) &= I(c_K < X) \end{aligned}$$

Above  $I(\cdot)$  is the *indicator* function. For example  $I(c_1 < X \leq c_2) = 1$  if  $c_1 < X \leq c_2$  and equals 0 otherwise.

*Note: trivially we have that:  $C_0(X) + C_1(X) + \dots + C_K(X) = 1$  since  $X$  must be in exactly one of the  $K + 1$  intervals.*

After defining appropriate intervals and constructing the corresponding dummy variables we fit LS to the following model:

$$y = f(x) = \beta_0 + \beta_1 C_1(x) + \beta_2 C_2(x) + \dots + \beta_K C_K(x) + \epsilon.$$

Inclusion of  $C_0(x)$  is not needed because this effect is represented by  $\beta_0$ .

This means that if  $X \leq c_1$  our prediction is  $\beta_0$ , while if  $c_j < X \leq c_{j+1}$  we predict  $\beta_0 + \beta_j$  for  $j = 1, \dots, K$ . In other words,  $\beta_0$  can be interpreted as the mean value of  $Y$  for  $X \leq c_1$ , and  $\beta_j$  represents the average increase in the response for  $X$  in  $c_j < X \leq c_{j+1}$  relative to  $X \leq c_1$ .

#### Example: Wage Data

We apply step-function regression to model the `Wage` data introduced in Section 8.2, with different numbers of step functions (Figure 9.1). We can see that this modelling approach is local: new samples within a sampling interval affect only the predicted effect within that interval. We see that this approach creates a bumpy fit to the data. This is desirable sometimes in applications in biomedicine or social sciences where the goal is to report overall summaries within specific age groups. In other applications smoothness is preferable.

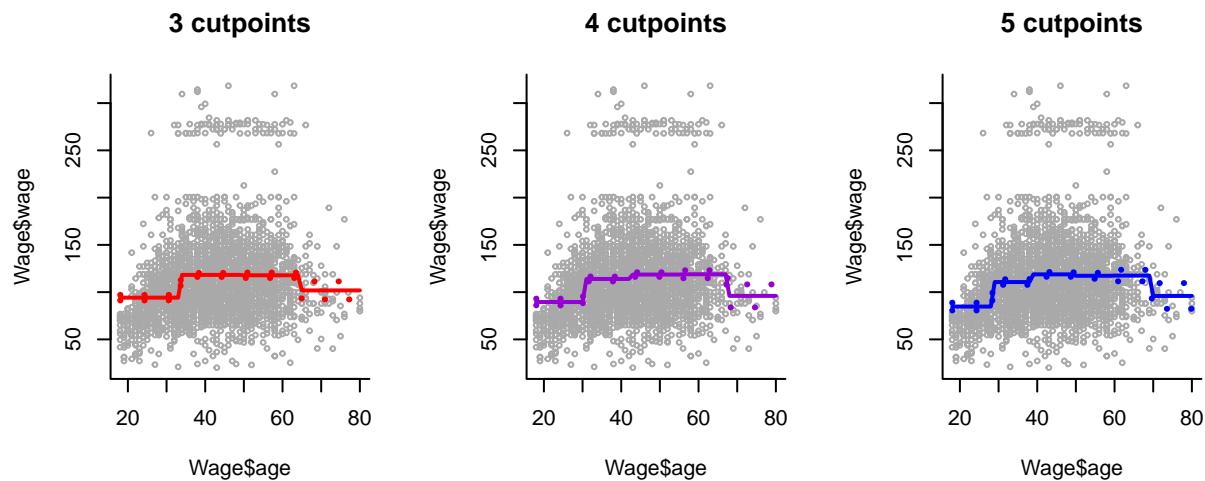


Figure 9.1: Step function regression with different numbers of cutpoints applied to the Wage data.

## 9.2 Practical Demonstration

We once again make use of the Boston dataset in library MASS.

```
library("MASS")
y = Boston$medv
x = Boston$lstat
y.lab = 'Median Property Value'
x.lab = 'Lower Status (%)'
```

For step function regression we can make use of the command `cut()`, which automatically assigns samples to intervals given a specific number of intervals. We can check how this works by executing the following line of code.

```
table(cut(x, 2))

##
## (1.69,19.9]    (19.9,38]
##          430           76
```

What we see is that `cut(x, 2)` automatically created a factor with two levels, corresponding to the intervals  $(1.69, 19.9]$  and  $(19.9, 38]$ , and assigned each entry in  $x$  to one of these factors depending on which interval it was in. The command `table()` tells us that 430 samples of  $x$  fall within the first interval and that 76 samples fall within the second interval. Note that `cut(x, 2)` generated 2 intervals, but this means there is only 1 cutpoint (at 19.9). The number of cutpoints is naturally one less than the number of intervals, but it is important to be aware that `cut` requires specification of the **number of required intervals**.

So, we can use `cut()` within `lm()` to easily fit regression models with step functions. Below we consider 4 models with 1, 2, 3 and 4 cutpoints (2, 3, 4 and 5 intervals) respectively.

```
step2 = lm(y ~ cut(x, 2))
step3 = lm(y ~ cut(x, 3))
step4 = lm(y ~ cut(x, 4))
```

```
step5 = lm(y ~ cut(x, 5))
```

The analysis then is essentially the same as previously. The following code produces the 2 by 2 plot of the fitted lines of the four models, along with approximate 95% confidence intervals for the mean predictions.

```
pred2 = predict(step2, newdata = list(x = sort(x)), se = TRUE)
pred3 = predict(step3, newdata = list(x = sort(x)), se = TRUE)
pred4 = predict(step4, newdata = list(x = sort(x)), se = TRUE)
pred5 = predict(step5, newdata = list(x = sort(x)), se = TRUE)

se.bands2 = cbind(pred2$fit + 2*pred2$se.fit, pred2$fit-2*pred2$se.fit)
se.bands3 = cbind(pred3$fit + 2*pred3$se.fit, pred3$fit-2*pred3$se.fit)
se.bands4 = cbind(pred4$fit + 2*pred4$se.fit, pred4$fit-2*pred4$se.fit)
se.bands5 = cbind(pred5$fit + 2*pred5$se.fit, pred5$fit-2*pred5$se.fit)

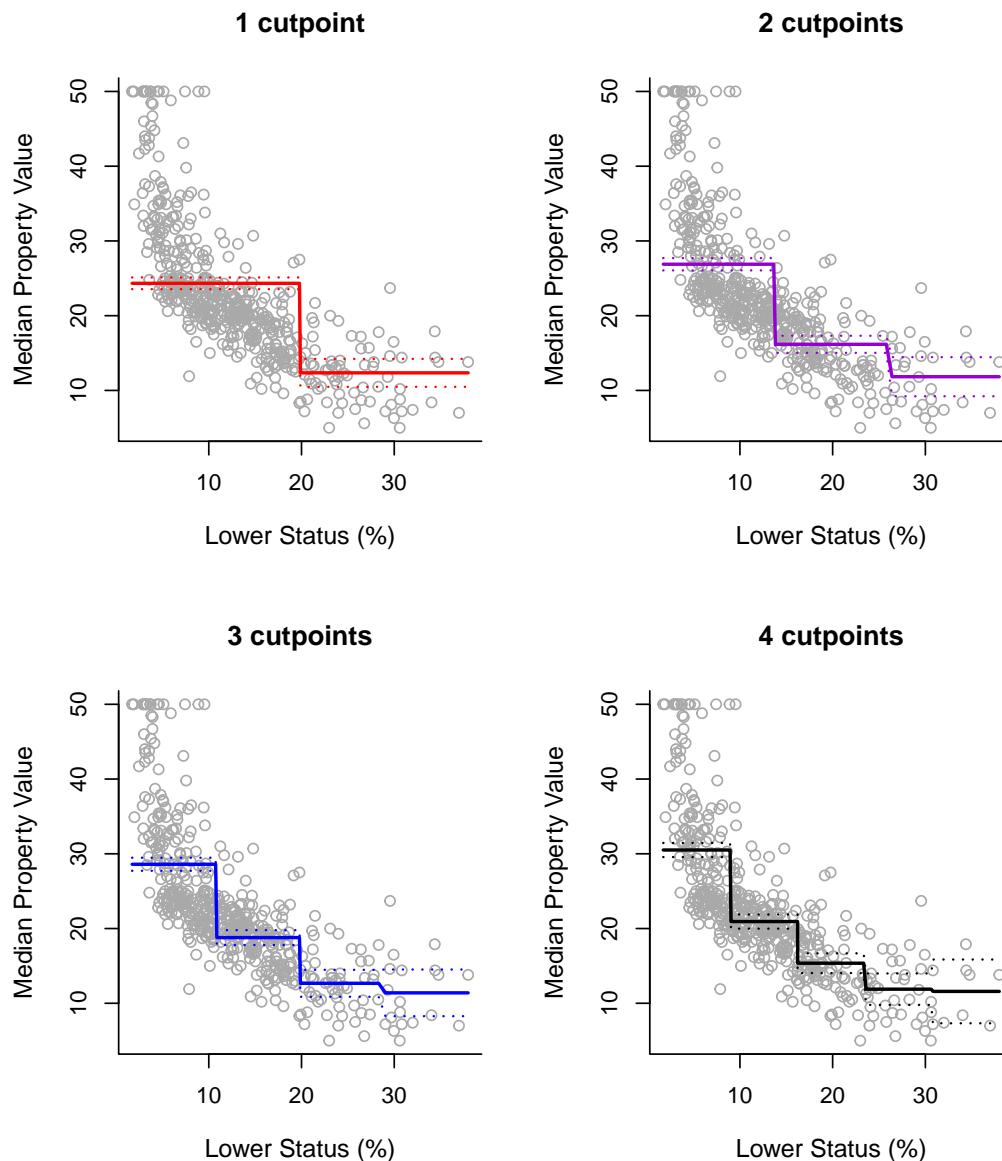
par(mfrow = c(2,2))

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "1 cutpoint", bty = 'l')
lines(sort(x), pred2$fit, lwd = 2, col = "red")
matlines(sort(x), se.bands2, lwd = 1.4, col = "red", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "2 cutpoints", bty = 'l')
lines(sort(x), pred3$fit, lwd = 2, col = "darkviolet")
matlines(sort(x), se.bands3, lwd = 1.4, col = "darkviolet", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "3 cutpoints", bty = 'l')
lines(sort(x), pred4$fit, lwd = 2, col = "blue")
matlines(sort(x), se.bands4, lwd = 1.4, col = "blue", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "4 cutpoints", bty = 'l')
lines(sort(x), pred5$fit, lwd = 2, col = "black")
matlines(sort(x), se.bands5, lwd = 1.4, col = "black", lty = 3)
```



Note that we do not necessarily need to rely on the automatic selections of cutpoints used by `cut()`. We can define the intervals if we want to. For instance, if we want cutpoints at 10, 20 and 30 we can do the following

```
breaks4 = c(min(x), 10, 20, 30, max(x))
table(cut(x, breaks = breaks4))
```

```
##
## (1.73,10]   (10,20]    (20,30]    (30,38]
##      218       213        62         12
```

Note that when defining the sequence of breaks, we included `min(x)` and `max(x)` at the start and end in order to ensure the intervals covered the entire range of `x`.

Then our model would be:

```
step.new4 = lm(y ~ cut(x, breaks = breaks4))
summary(step.new4)
```

```

## 
## Call:
## lm(formula = y ~ cut(x, breaks = breaks4))
## 
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.4803  -4.6239  -0.4239   2.8968  20.6197
## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  29.3803    0.4415  66.540 <0.0000000000000002
## cut(x, breaks = breaks4)(10,20] -10.4563    0.6281 -16.648 <0.0000000000000002
## cut(x, breaks = breaks4)(20,30] -16.6770    0.9383 -17.773 <0.0000000000000002
## cut(x, breaks = breaks4)(30,38] -18.6886    1.9331  -9.668 <0.0000000000000002
## 
## (Intercept) *** 
## cut(x, breaks = breaks4)(10,20] ***
## cut(x, breaks = breaks4)(20,30] ***
## cut(x, breaks = breaks4)(30,38] ***
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 6.519 on 501 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.4925, Adjusted R-squared:  0.4895
## F-statistic: 162.1 on 3 and 501 DF,  p-value: < 0.0000000000000022

```

Although somewhat trivial, we can make predictions at new data points using the constructed linear model as usual.

```

newx <- c(10.56, 5.89)
preds = predict(step.new4, newdata = list(x = newx), se = TRUE)
preds

```

```

## $fit
##      1      2
## 18.92394 29.38028
## 
## $se.fit
##      1      2
## 0.4466955 0.4415432
## 
## $df
## [1] 501
## 
## $residual.scale
## [1] 6.519307

```



# Chapter 10

## Splines

### 10.1 Regression Splines

#### Why Splines?

- We have seen that polynomial regression leads to flexible and smooth curves, but is trained globally which is problematic.
- Step functions are trained locally but produce “bumpy” fits, which are desirable only in specific applications.
- Regression splines bridge these differences by providing adaptive local smoothness.

#### Piecewise Polynomials

- Regression based on splines is a general approach which *encompasses* different models.
- The basis of regression splines is *piecewise polynomial regression*.
- Piecewise polynomials are not fitted over the entire range of  $X$  but *over different regions* of  $X$ .
- Polynomial regression and step functions are special simple cases of piecewise polynomial regression.

*Standard polynomial regression:*

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \dots + \beta_d x_i^d + \epsilon_i.$$

*Piecewise polynomial regression:*

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \dots + \beta_{d1} x_i^d + \epsilon_i, & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \dots + \beta_{d2} x_i^d + \epsilon_i, & \text{if } x_i \geq c, \end{cases}$$

where  $i = 1, \dots, n$  in both cases.

- The above piecewise polynomial requires two LS fits: one for the samples where  $x < c$ , and one for the samples where  $x \geq c$ .
- Within this framework the *cutpoint*  $c$  is called a *knot*.
- When there is no knot we have standard polynomial regression.
- When we include only the intercept terms ( $\beta_{01}, \beta_{02}$ ) we have step-function regression.

## Knots

- Before we had 1 knot leading to 2 different models.
- Of course, we can introduce more than one knot. In general, if we have  $K$  knots we are fitting  $K + 1$  polynomial models.
- Using more knots leads to more *flexible* polynomials.
- Of course, *too many* knots  $\rightarrow$  *overfitting*!
- More on the selection of number of knots later...

## Polynomial Degree

*Linear piecewise:*

$$y = \begin{cases} \beta_{01} + \beta_{11}x, & \text{if } x_i < c, \\ \beta_{02} + \beta_{12}x, & \text{if } x_i \geq c. \end{cases}$$

*Cubic piecewise:*

$$y = \begin{cases} \beta_{01} + \beta_{11}x + \beta_{21}x^2 + \beta_{31}x^3, & \text{if } x_i < c, \\ \beta_{02} + \beta_{12}x + \beta_{22}x^2 + \beta_{32}x^3, & \text{if } x_i \geq c. \end{cases}$$

- Here, for simplicity we consider the case  $K = 1$ .
- In the first case  $d = 1$ , while in the second  $d = 3$ .
- The quadratic case with  $d = 2$  is usually not considered: it is typically outperformed by the cubic case under complex data structures and does not perform remarkably better than the linear case on simpler data structures.
- In general, *cubic piecewise* is the most *popular* choice.

## Example: Wage Data

We consider the `Wage` data once again, however, for visual clarity only a subset of the data is considered. In Figure 10.1 James et al. [2013], we have fit a piecewise cubic polynomial with a single knot placed at `age` = 50. We can see that there are two different curves on the left and right of the knot (which is what we wanted), however, the *discontinuity* at `age` = 50 is obviously not what we wanted!

How do we fix this? The answer is to use constraints again!

The two polynomials in the above plot are of the following form.

$$\text{wage} = \begin{cases} f_1(\text{age}) = \beta_{01} + \beta_{11}\text{age} + \beta_{21}\text{age}^2 + \beta_{31}\text{age}^3, & \text{if age} < 50, \\ f_2(\text{age}) = \beta_{02} + \beta_{12}\text{age} + \beta_{22}\text{age}^2 + \beta_{32}\text{age}^3, & \text{if age} \geq 50, \end{cases}$$

We can make these two cubic polynomial lines meet at `age` = 50 by imposing the constraint

$$\beta_{01} + \beta_{11}50 + \beta_{21}50^2 + \beta_{31}50^3 = \beta_{02} + \beta_{12}50 + \beta_{22}50^2 + \beta_{32}50^3$$

Where we can see this as a constraint on a single parameter, since, by rearranging, we get

$$\beta_{01} = \beta_{02} + (\beta_{12} - \beta_{11})50 + (\beta_{22} - \beta_{21})50^2 + (\beta_{32} - \beta_{31})50^3$$

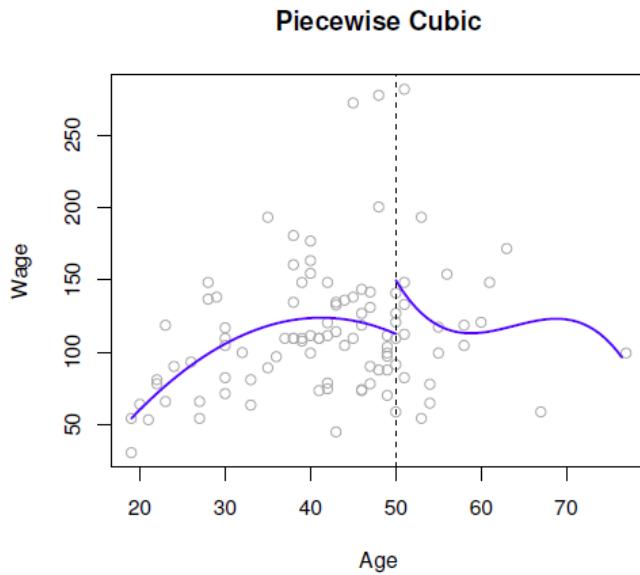


Figure 10.1: Unconstrained piecewise cubic polynomials are fit to a subset of the Wage data.

In other words, if we have estimates for  $\hat{\beta}_{11}$ ,  $\hat{\beta}_{21}$ ,  $\hat{\beta}_{31}$ ,  $\hat{\beta}_{02}$ ,  $\hat{\beta}_{12}$ ,  $\hat{\beta}_{22}$  and  $\hat{\beta}_{32}$ , then  $\hat{\beta}_{01}$  is defined by the constraint.

This constraint leads to the continuous piecewise cubic polynomial seen in Figure 10.2.

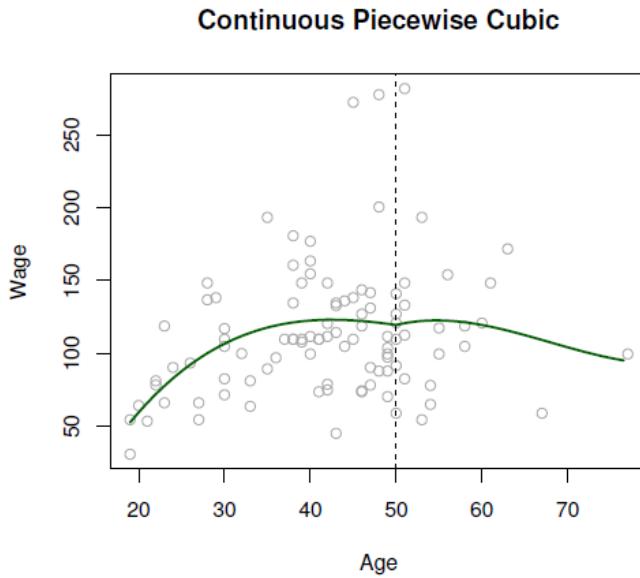


Figure 10.2: Constrained piecewise cubic polynomials are fit to a subset of the Wage data.

This looks better, although the small dip at  $age = 50$  doesn't make it perfect.

We therefore extend further the strategy based on constraints.

- So far we have set  $f_1(age = 50) = f_2(age = 50)$ , which ensures continuity.
- We can do the same for the *first derivative*:  $f'_1(age = 50) = f'_2(age = 50)$ . This will

constrain an additional parameter (coefficient), but it will make the function *smoother* at age = 50.

- Doing the same for the *second derivative*:  $f_1''(\text{age} = 50) = f_2''(\text{age} = 50)$ , will constrain one more parameter, and the function will become *very smooth* at age = 50.

Can we add a further constraint on the third derivative? No, our function is cubic so the 3rd derivative is a constant (it is not a function of age).

A cubic piecewise polynomial with these three constraints is called a *cubic spline*.

We fit a cubic spline in Figure 10.3,

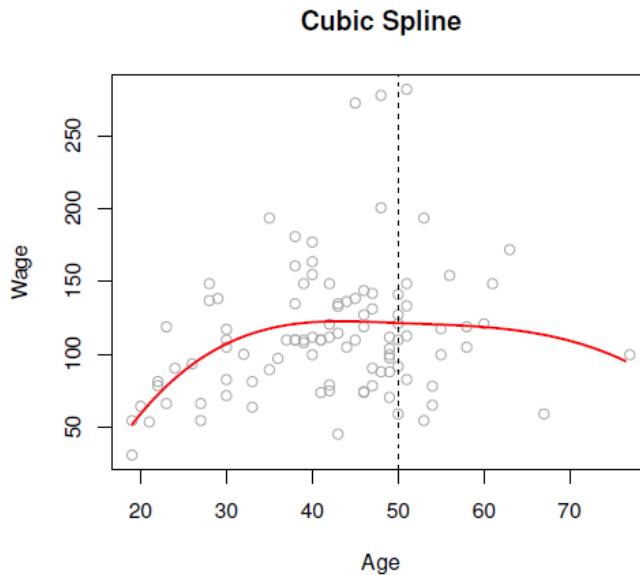


Figure 10.3: A cubic spline is fit to a subset of the Wage data.

Cubic splines are so popular because they look perfectly smooth to the human eyes! At least that is the claim...

## Constraints and Degrees of Freedom

- In the previous example, we started with a cubic piecewise polynomial with 8 unconstrained parameters, so we started with 8 *degrees of freedom (df)*.
- We initially imposed one constraint, which restricted one parameter, so we lost a degree of freedom  $\rightarrow 8 - 1 = 7 \text{ df}$ .
- With the further two constraints:  $8 - 3 = 5 \text{ df}$ .

**In general**, a cubic spline with  $K$  knots has  $4 + K$  degrees of freedom. This is useful to know because as we will see in RStudio we can specify either the number of knots  $K$  or just the degrees of freedom.

## General Definition

*A degree-d regression spline is a piecewise degree-d polynomial with continuity in derivatives up to degree  $d - 1$  at each knot.*

The degrees of freedom of a degree- $d$  spline with  $K$  knots are  $(d + 1) + K$ .

- Linear spline: just continuous ( $2+K$  df)
- Quadratic spline: continuous and continuous 1st derivative ( $3+K$  df)
- Cubic spline: continuous and continuous 1st and 2nd derivatives ( $4+K$  df)
- And so on...

In general, regression splines give us the maximum amount of continuity we can have given the degree  $d$ .

## Spline Basis Representation

A degree- $d$  spline with knots at  $\xi_k$  for  $k = 1, \dots, K$  can be represented by *truncated power basis functions*, denoted by  $b_i$  for  $i = 1, \dots, K + d$ , so that:

$$y = \beta_0 + \beta_1 b_1(x) + \dots + \beta_{K+d} b_{K+d}(x) + \epsilon.$$

where:

$$\begin{aligned} b_1(x) &= x^1 \\ &\vdots \\ b_d(x) &= x^d \\ b_{(k+d)}(x) &= (x - \xi_k)_+^d, \quad k = 1, \dots, K, \end{aligned}$$

where

$$(x - \xi_k)_+^d = \begin{cases} (x - \xi_k)^d & \text{if } x > \xi_k \\ 0 & \text{otherwise.} \end{cases}$$

### Example: linear spline, one knot

In the illustration shown in Figure 10.4, we just have  $y = \begin{cases} \beta_1 x, & \text{if } x < \text{knot} \\ \beta_1 x + \beta_2 (x - \text{knot}), & \text{if } x \geq \text{knot.} \end{cases}$

See the recommended literature for further detail on the spline basis representation.

### Example: Wage Data

In Figure 10.5, we fit a linear, quadratic and cubic spline with three knots (at the dotted lines) to a subset of the `Wage` data.

```
## Warning in if (se.fit) list(fit = predictor, se.fit = se, df = df,
## residual.scale = sqrt(res.var)) else predictor: the condition has length > 1 and
## only the first element will be used

## Warning in if (se.fit) list(fit = predictor, se.fit = se, df = df,
## residual.scale = sqrt(res.var)) else predictor: the condition has length > 1 and
## only the first element will be used
```

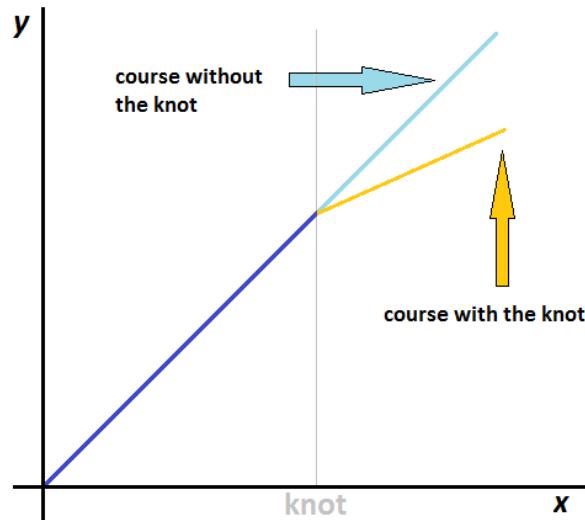


Figure 10.4: An illustration of a linear spline with a single knot.

```
## Warning in if (se.fit) list(fit = predictor, se.fit = se, df = df,
## residual.scale = sqrt(res.var)) else predictor: the condition has length > 1 and
## only the first element will be used
```

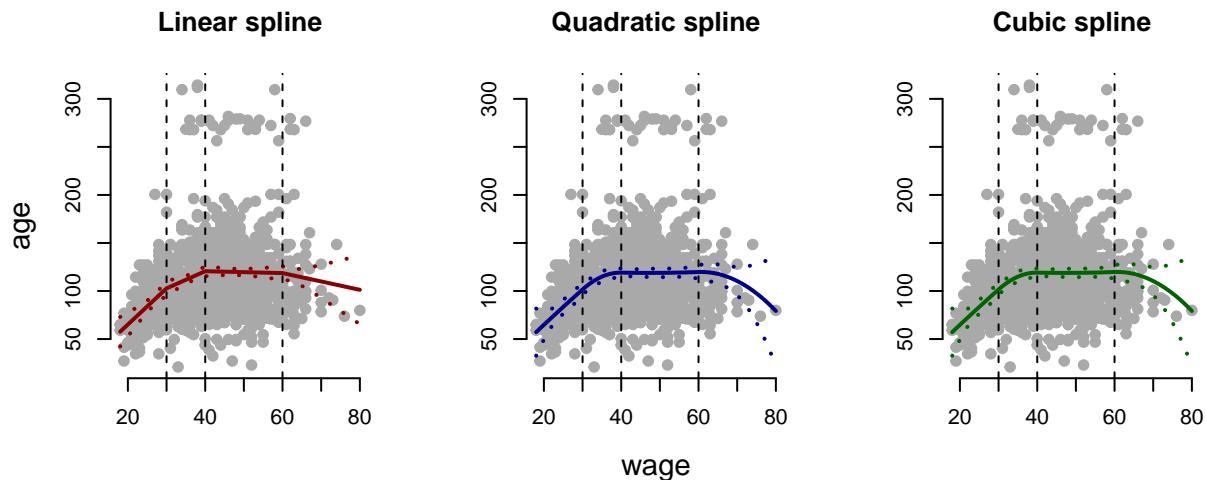


Figure 10.5: A linear, quadratic and cubic spline with three knots fit to a subset of the Wage data.

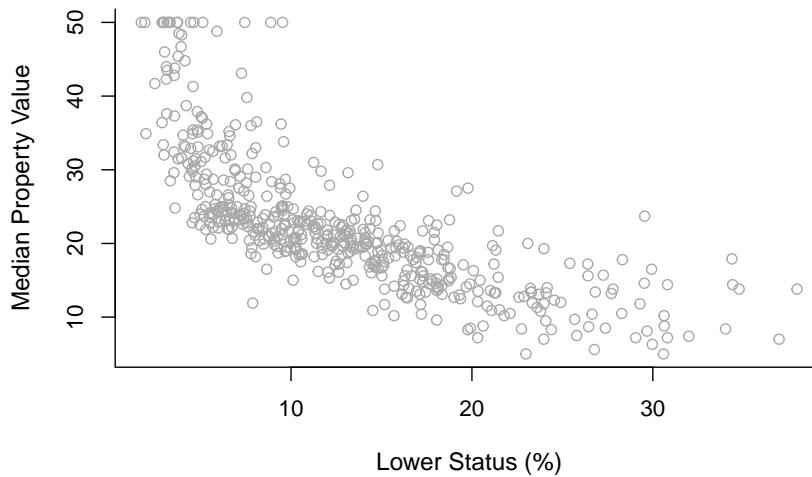
## 10.2 Practical Demonstration

We will continue analysing the Boston dataset included in library MASS.

### Initial steps

We load library MASS again. As previously, we will analyse `medv` with respect to the predictor `lstat`. Once again, for convenience, we will name the response `y` and the predictor `x`, and pre-define plot axis labels.

```
library(MASS)
y = Boston$medv
x = Boston$lstat
y.lab = 'Median Property Value'
x.lab = 'Lower Status (%)'
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "", bty = 'l')
```



## Regression Splines

For this analysis we will require package **splines**.

```
library(splines)
```

Initially let's fit regression splines by specifying *knots*. From the previous plot it is not clear where exactly we should place knots, so we will make use of the command `summary` in order to find the 25th, 50th and 75th percentiles of `x`, which will be the positions where we will place the knots.

```
summary(x)
```

```
##      Min. 1st Qu. Median   Mean 3rd Qu.    Max.
##      1.73    6.95   11.36   12.65   16.95   37.97
```

```
cuts = summary(x)[c(2, 3, 5)]
cuts
```

```
## 1st Qu. Median 3rd Qu.
## 6.950 11.360 16.955
```

Also, before we fit the splines it is again important to sort the variable `x`.

```
sort.x = sort(x)
```

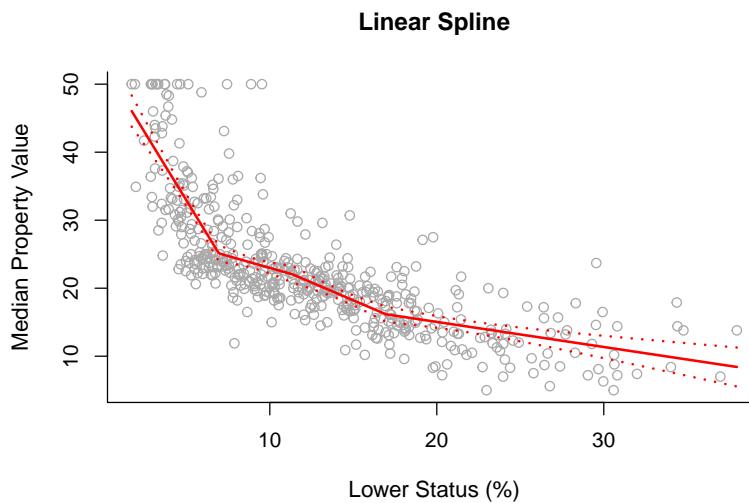
For a start lets fit a linear spline using our selected placement of knots. For this we can use command `lm()` and inside it we use the command `bs()` in which we specify `degree = 1` for a linear spline and `knots = cuts` for the placement of the knots at the three percentiles. We

also calculate the corresponding fitted values and confidence intervals exactly in the same way we did in previous practical demonstrations.

```
spline1 = lm(y ~ bs(x, degree = 1, knots = cuts))
pred1 = predict(spline1, newdata = list(x = sort.x), se = TRUE)
se.bands1 = cbind(pred1$fit + 2 * pred1$se.fit,
                  pred1$fit - 2 * pred1$se.fit)
```

Having done that we can now produce the corresponding plot.

```
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Linear Spline", bty = 'l')
lines(sort.x, pred1$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands1, lwd = 2, col = "red", lty = 3)
```



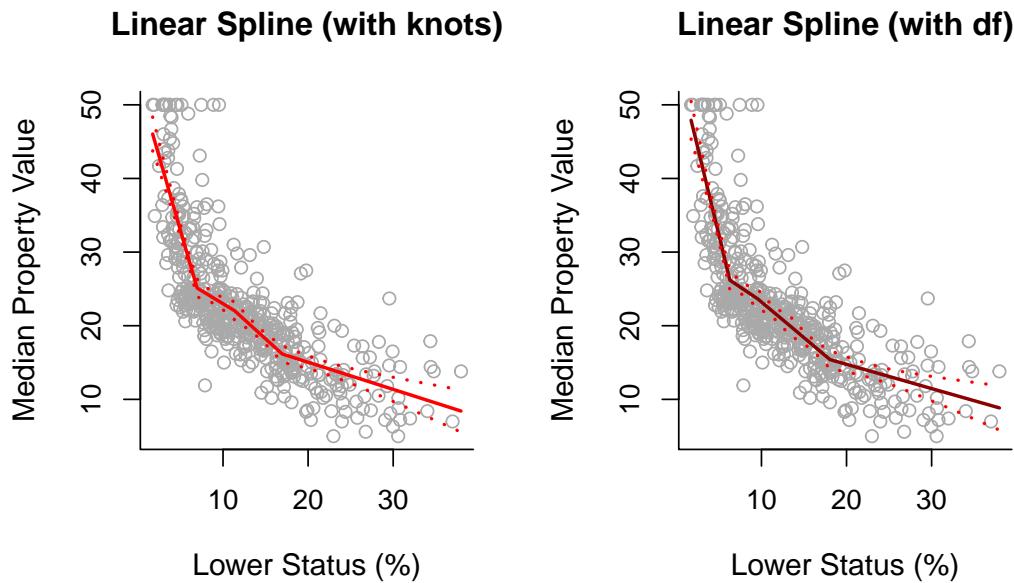
Using `?bs` we see that instead of using the argument `knots` we can use the argument `df`, which are the *degrees of freedom*. We know that splines have  $(d+1)+K$  degrees of freedom, where  $d$  is the degree of the polynomial. So in this case we have  $1+1+3=5$  degrees of freedom. Selecting `df = 5` in `bs()` will automatically use 3 knots placed at the 25th, 50th and 75th percentiles. Below we check whether the plot based on `df=5` is indeed the same as the previous plot and as we can see it is.

```
spline1df = lm(y ~ bs(x, degree = 1, df = 5))
pred1df = predict(spline1df, newdata = list(x = sort.x), se = TRUE)
se.bands1df = cbind( pred1df$fit + 2 * pred1df$se.fit,
                     pred1df$fit - 2 * pred1df$se.fit )

par(mfrow = c(1, 2))
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Linear Spline (with knots)", bty = 'l')
lines(sort.x, pred1$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands1, lwd = 2, col = "red", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Linear Spline (with df)", bty = 'l')
```

```
lines(sort.x, pred1df$fit, lwd = 2, col = "darkred")
matlines(sort.x, se.bands1df, lwd = 2, col = "red", lty = 3)
```



Having seen how this works we can also fit a degree-2 (quadratic) and degree-3 (cubic) spline to the data, all we have to do is change `degree = 1` to `degree = 2` and `degree = 3` respectively. Also we increase the respective degrees of freedom from `df = 5` to `df = 6` and `df = 7` in order to keep the same number (and position) of knots in the quadratic and cubic spline models.

```
spline2 = lm(y ~ bs(x, degree = 2, df = 6))
pred2 = predict(spline2, newdata = list(x = sort.x), se = TRUE)
se.bands2 = cbind(pred2$fit + 2 * pred2$se.fit, pred2$fit - 2 * pred2$se.fit)

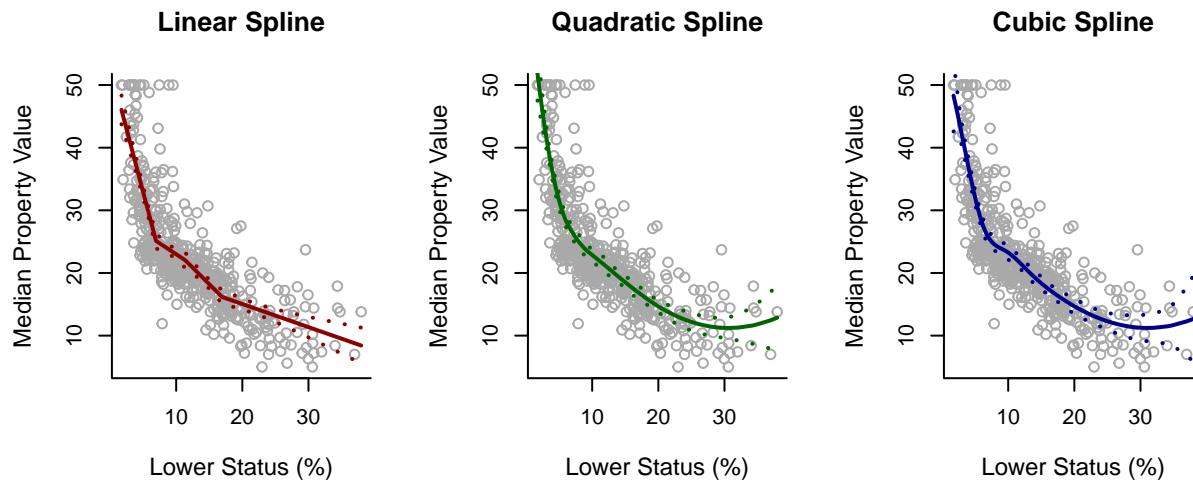
spline3 = lm(y ~ bs(x, degree = 3, df = 7))
pred3 = predict(spline3, newdata = list(x = sort.x), se = TRUE)
se.bands3 = cbind(pred3$fit + 2 * pred3$se.fit, pred3$fit - 2 * pred3$se.fit)

par(mfrow = c(1,3))
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Linear Spline", bty = 'l')
lines(sort.x, pred1$fit, lwd = 2, col = "darkred")
matlines(sort.x, se.bands1, lwd = 2, col = "darkred", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Quadratic Spline", bty = 'l')
lines(sort.x, pred2$fit, lwd = 2, col = "darkgreen")
matlines(sort.x, se.bands2, lwd = 2, col = "darkgreen", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Cubic Spline", bty = 'l')
lines(sort.x, pred3$fit, lwd = 2, col = "darkblue")
```

```
matlines(sort.x, se.bands3, lwd = 2, col = "darkblue", lty = 3)
```



## 10.3 Natural Splines

Splines, especially cubic splines, seem to be the best tool we have so far, right?

Well, there is always room for improvement:

- Regression splines usually have *high variance* at the outer range of the predictor (the *tails*).
- Sometimes the confidence intervals at the tails are wiggly (especially for small sample sizes).

*Natural splines* are extensions of regression splines which remedy these problems to some extent.

## Additional Constraints

Natural splines impose two additional constraints at each boundary region:

- The spline function is constrained to be close to linear when  $X <$  smallest knot ( $c_1$ ) by  $f''_0(c_1) = 0$ .
- The spline function is constrained to be close to linear when  $X >$  largest knot ( $c_n$ ) by  $f''_n(c_n) = 0$ .

*These additional constraints of natural splines generally result in more stable estimates at the boundaries.*

## Example: Wage Data

We can see from Figure 10.6 that use of a natural spline has constrained the model at the lower and upper ends, and reduced the variance.

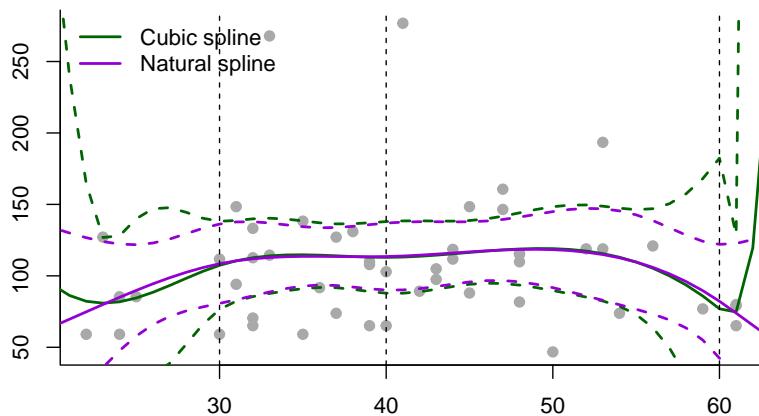


Figure 10.6: Fitting cubic and natural splines to a subset of the Wage data.

## How Many Knots and Where?

- Provided there is evidence from the data we can do it empirically:
    - Place knots where it is clearly obvious that we have a distributional shift in direction.
    - Place more knots on regions where we see more variability.
    - Place fewer knots in places which look more stable.
  - Alternatively, we can place knots in a uniform fashion (e.g. at the 25th, 50th and 75th percentiles, as we saw how to do in Section 10.2).

## 10.4 Practical Demonstration

For natural splines, we can use the command `ns()` in RStudio. As with the command `bs()` previously, we again have the option to either specify the knots manually (via the argument `knots`) or to simply pre-define the degrees of freedom (via the argument `df`). Below we use the latter option to fit four natural splines with 1, 2, 3 and 4 degrees of freedom. As we see using 1 degree of freedom actually results in just a linear model.

```
spline.ns4 = lm(y ~ ns(x, df = 4))
pred.ns4 = predict(spline.ns4, newdata = list(x = sort.x), se = TRUE)
se.bands.ns4 = cbind(pred.ns4$fit + 2 * pred.ns4$se.fit,
                      pred.ns4$fit - 2 * pred.ns4$se.fit)

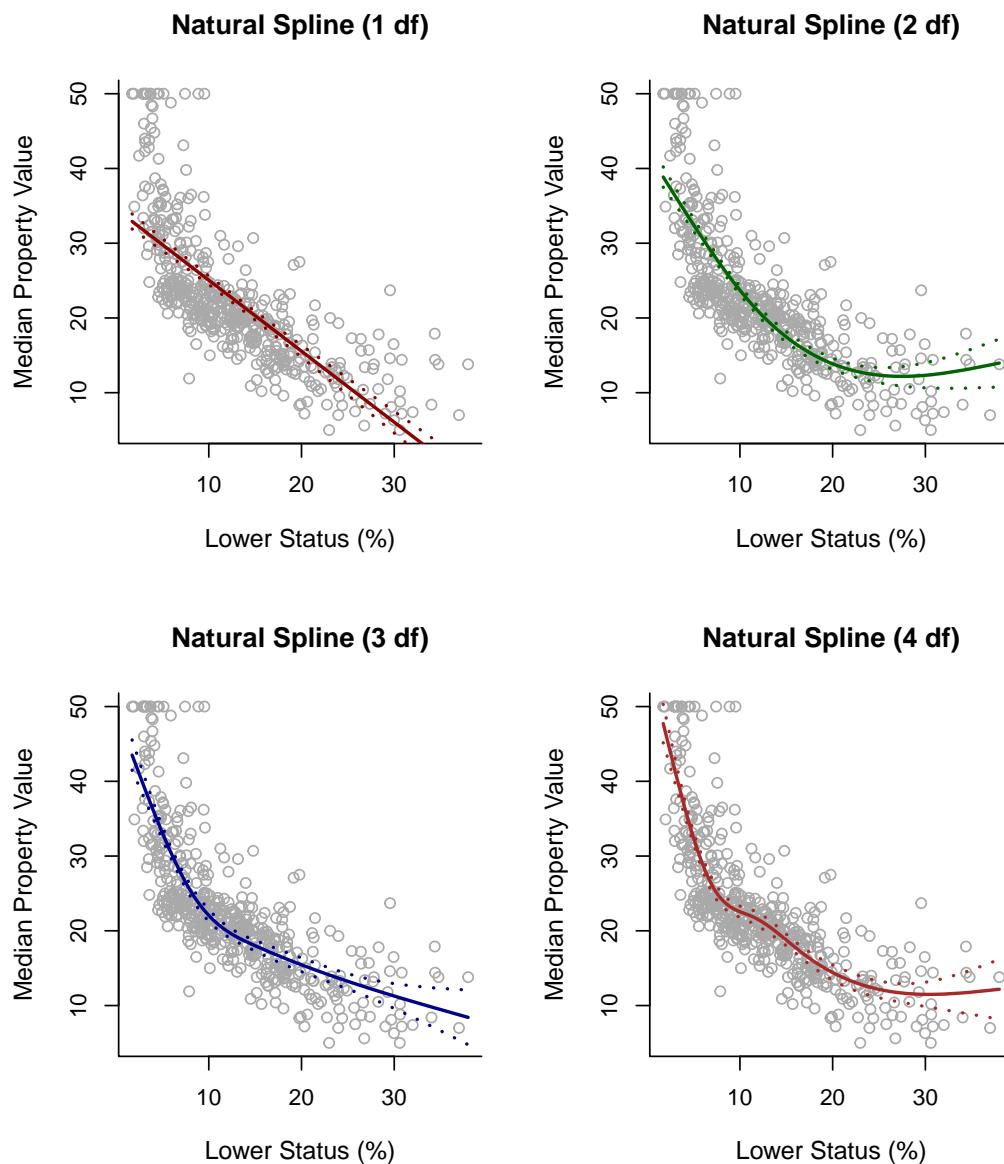
par(mfrow = c(2, 2))

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Natural Spline (1 df)", bty = 'l')
lines(sort.x, pred.ns1$fit, lwd = 2, col = "darkred")
matlines(sort.x, se.bands.ns1, lwd = 2, col = "darkred", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Natural Spline (2 df)", bty = 'l')
lines(sort.x, pred.ns2$fit, lwd = 2, col = "darkgreen")
matlines(sort.x, se.bands.ns2, lwd = 2, col = "darkgreen", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Natural Spline (3 df)", bty = 'l')
lines(sort.x, pred.ns3$fit, lwd = 2, col = "darkblue")
matlines(sort.x, se.bands.ns3, lwd = 2, col = "darkblue", lty = 3)

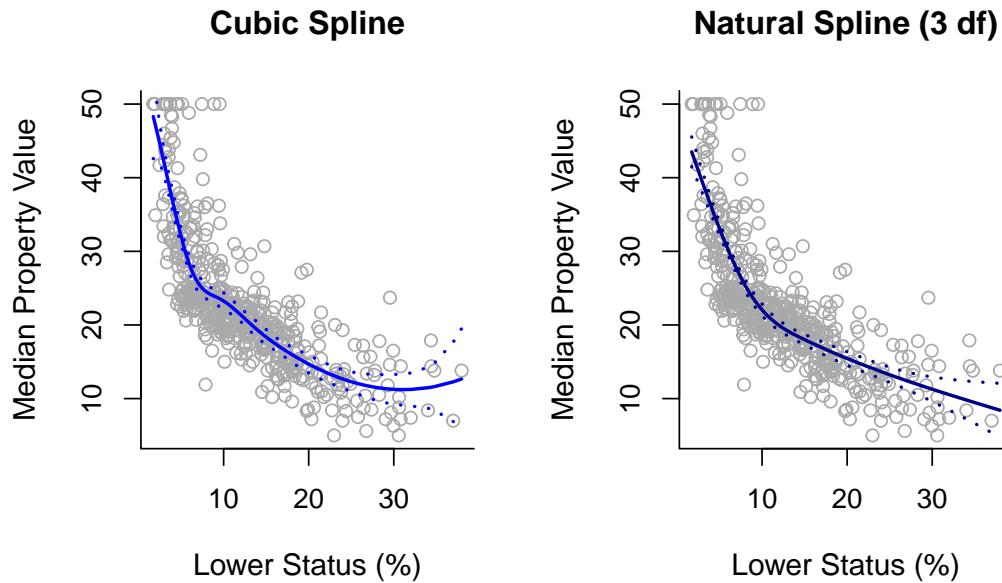
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Natural Spline (4 df)", bty = 'l')
lines(sort.x, pred.ns4$fit, lwd = 2, col = "brown")
matlines(sort.x, se.bands.ns4, lwd = 2, col = "brown", lty = 3)
```



Below we plot the cubic spline next to the natural cubic spline for comparison. As we can see, the natural cubic spline is generally smoother and closer to linear on the right boundary of the predictor space, where it has, additionally, narrower confidence intervals in comparison to the cubic spline.

```
par(mfrow = c(1,2))
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Cubic Spline", bty = 'l')
lines(sort.x, pred3$fit, lwd = 2, col = "blue")
matlines(sort.x, se.bands3, lwd = 2, col = "blue", lty = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Natural Spline (3 df)", bty = 'l')
lines(sort.x, pred.ns3$fit, lwd = 2, col = "darkblue")
matlines(sort.x, se.bands.ns3, lwd = 2, col = "darkblue", lty = 3)
```



## 10.5 Smoothing Splines

*Smoothing splines* are quite different from the non-linear modelling methods we have seen so far. Unlike regression splines and natural splines, there are no knots! Smoothing splines turns the *discrete* problem of selecting a number of knots into a *continuous penalisation* problem. The maths here is rather complicated, so we mainly introduce this topic by intuition rather than go into a load of technical details.

### Minimisation

We seek a function  $g$  among all possible functions (linear or non-linear) which minimises

$$\sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int (g''(t))^2 dt,$$

where  $\lambda \geq 0$  is a *tuning parameter*.

The function  $g$  that minimises the above quantity is called a *smoothing spline*.

### Model Fit + Penalty Term

$$\underbrace{\sum_{i=1}^n (y_i - g(x_i))^2}_{\text{Model fit}} + \underbrace{\lambda \int (g''(t))^2 dt}_{\text{Penalty term}}$$

- When  $\lambda = 0$  we are just left with the model fit term.
- When  $\lambda \rightarrow \infty$ ... well more on that later.
- Lets examine the two terms separately.

## The Model Fit Term

$$\sum_{i=1}^n (y_i - g(x_i))^2$$

This is a RSS but not our “usual” RSS:

- In all previous approaches, function  $g$  was linear in the parameters (coefficients of the predictors) and we minimised with respect to these parameters.
- Here, we have one predictor but the minimisation is with respect to  $g$ !
- So, if we let  $g$  be *unconstrained* and just minimise the above RSS, then we can end up with any weird-looking non linear function that passes through exactly every data point, leading to an RSS equal to 0 (*overfitting!*).

Thus, the penalty on  $g$ ...

## The Penalty Term

$$\lambda \int (g''(t))^2 dt$$

- $g''(t)$  is the 2nd derivative of  $g$  at a point  $t$  within the space of predictor  $X$ .
- The 2nd derivative of a function catches wiggles or non-linearities: if  $g''(t^*)^2$  at a point  $t^*$  is large, this means that  $g$  at this point  $t^*$  suddenly jumps either up or down.
- The integration above over every  $t$  in the space of  $X$  is essentially a sum. The larger it is the more non-linear the function  $g$  is.

## Putting it all Together

Minimizing

$$\sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int (g''(t))^2 dt,$$

with respect to  $g$  turns out to be interesting.

- When  $\lambda = 0$  we get an *extremely wiggly non-linear* function  $g$  (completely useless).
- As  $\lambda$  increases, the function  $g$  will become *smoother*.
- In the *theoretical* case of  $\lambda \rightarrow \infty$ ,  $g''$  will be zero everywhere. When does that happen? When  $g(x) = \beta_0 + \beta_1 x$ . So, in this case we return back to the *linear model*!

That is why the penalty term is also called a *roughness penalty* within the framework of smoothing splines.

## Solution

Surprisingly, the solution for any finite and non-zero  $\lambda$  is that the function  $g$  is a *natural cubic spline* **but** with *knots placed on each individual sample point  $x_1, \dots, x_n$* .

But doesn't that lead to overfitting? No, because now we have once again the penalty parameter  $\lambda$  which can *shrink* coefficients to zero and thus avoid overfitting.

## Tuning $\lambda$

- As usual  $\lambda$  can be tuned via cross-validation.
- Also,  $\lambda$  is associated with the *effective degrees of freedom* of a smoothing spline. These are similar to the degrees of freedom in standard spline models and can be used as an alternative to cross-validation as a way to fix  $\lambda$ .

## 10.6 Practical Demonstration

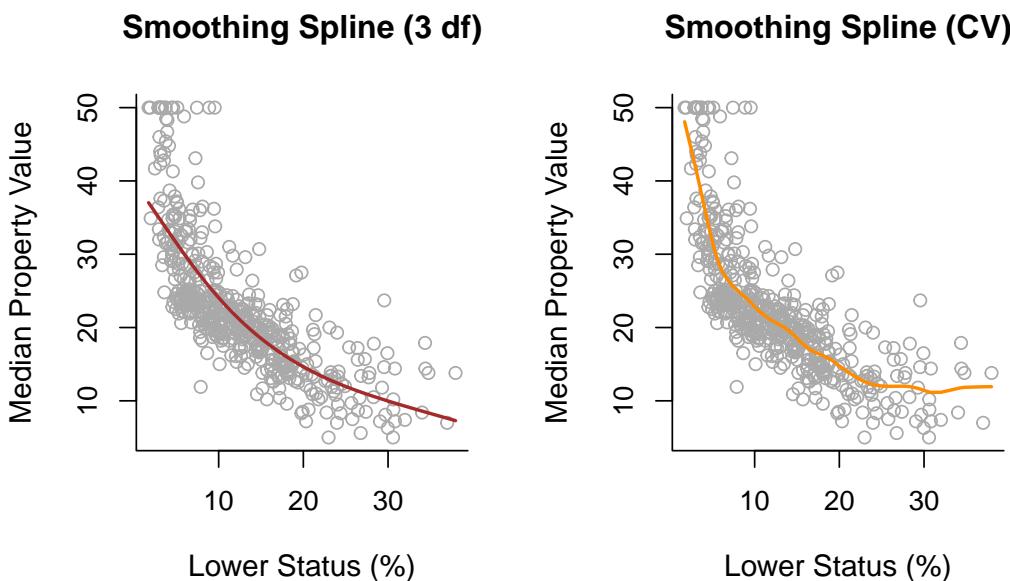
### Boston Data

For fitting smoothing splines we use the command `smooth.spline()` instead of `lm()`. Under smoothing splines there are no knots to specify; the only parameter is  $\lambda$ . This can be specified via cross-validation by specifying `cv = TRUE` inside `smooth.spline()`. Alternatively, we can specify the *effective degrees of freedom* which correspond to some value of  $\lambda$ . Below we use cross-validation as well as a smoothing spline with 3 effective degrees of freedom (via the argument `df = 3`). In this case we see that tuning  $\lambda$  through cross-validation results in a curve which is slightly wiggly on the right boundary of the predictor space.

```
smooth1 = smooth.spline(x, y, df = 3)
smooth2 = smooth.spline(x, y, cv = TRUE)

par(mfrow = c(1,2))
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Smoothing Spline (3 df)", bty = 'l')
lines(smooth1, lwd = 2, col = "brown")

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "Smoothing Spline (CV)", bty = 'l')
lines(smooth2, lwd = 2, col = "darkorange")
```



## Wage Data

Lets see how cubic splines, natural cubic splines and smoothing splines compare on the `Wage` data.

We will take sample size into consideration and see how the fitted curves look like for:

- A small sample of 50.
- A medium sample of 200.
- A large sample of 1000.

For the cubic spline and natural cubic spline we have to define the number and position of the knots. For simplicity we will adopt the previous strategy: 3 knots placed at the ages of 30, 40 and 60.

For the smoothing spline, everything is automatic -  $\lambda$  will be determined via CV by setting `cv = TRUE`.

Note that the code below runs this comparison for a sample size of  $n = 200$ . To plot results for  $n = 50$  and  $n = 1000$  (or any other value of  $n!$ ), run the code yourself in R and change the line `n <- 200` so that `n` is your chosen size.

```
library("ISLR")
library("splines") # Required for regression and natural splines.
agelims=range(Wage$age)
age.grid=seq(from=agelims[1],to=agelims[2])

set.seed(1)

# Number of data points - change this to investigate
# small, medium and large samples.
n <- 200

# Take the a sample of n points from the data.
ind = sample(1:3000, n)
Wage1 = Wage[ind,] # Label subset of data as Wage1.

# Cubic Spline
fitbs = lm(wage~bs(age, degree = 3, knots = c(30,40,60)), data = Wage1)
predbs = predict(fitbs, newdata = list(age = age.grid), se = T)

# Natural Spline
fitsns = lm(wage~ns(age, knots = c(30,40,60)), data = Wage1)
predns = predict(fitsns, newdata = list(age = age.grid), se = T)

# Smoothing Spline
fitss = smooth.spline(Wage1$age, Wage1$wage, cv = TRUE)

# Generate the Plots.
par(mfrow=c(1,3))
```

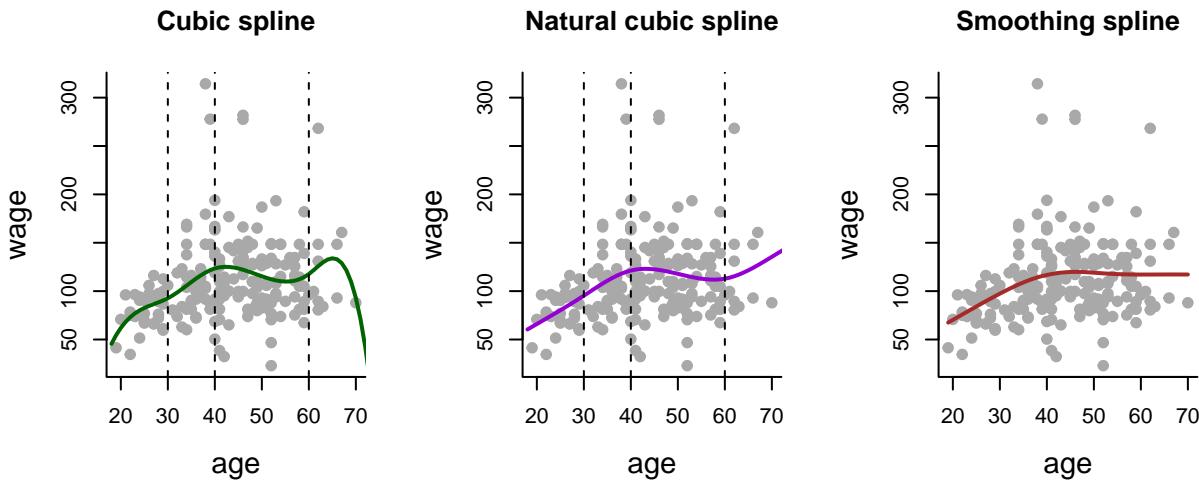
```

# Cubic Spline
plot(Wage1$age, Wage1$wage, col = "darkgray", pch = 19,
      main = 'Cubic spline', bty = 'l',
      xlab = 'age', ylab = 'wage', cex.lab = 1.4)
lines(age.grid, predbs$fit, lwd = 2, col = 'darkgreen')
abline(v = c(30,40,60), lty = 'dashed')

# Natural Spline
plot(Wage1$age, Wage1$wage, col = "darkgray", pch = 19,
      main = 'Natural cubic spline', bty = 'l',
      xlab = 'age', ylab = 'wage', cex.lab = 1.4)
lines(age.grid, predns$fit, lwd = 2, col = 'darkviolet')
abline(v = c(30,40,60), lty = 'dashed')

# Smoothing Spline
plot(Wage1$age, Wage1$wage, col = "darkgray", pch = 19,
      main = 'Smoothing spline', bty = 'l',
      xlab = 'age', ylab = 'wage', cex.lab = 1.4)
lines(fitss, lwd = 2, col = 'brown')

```



- If you run the above code for various sample sizes, you will see that, overall, the smoothing spline method is more robust: the shape of the curve remains more or less stable across different sample sizes. Importantly, this means that the fitted curve from the smoothing spline under  $n = 50$  would fit well to the data with  $n = 1000$ .
- That is not the case for the other two methods, especially for the cubic spline. These methods are more sensitive to the particular sample being used to train them.

## Drawback

You will notice that we didn't plot confidence intervals, because there is no built-in function in R package `smooth.splines` (that we used) that produces confidence intervals for smoothing splines. This is not a flaw of the package; this is a *generic issue* with smoothing splines as it is not as straightforward as it is with other spline methods to calculate confidence intervals.

One can get confidence-like intervals by using:

- Bayesian credible intervals
- Bootstrap

These methods require additional work and are beyond the scope of this course.



# Chapter 11

## Generalised Additive Models

### 11.1 Review

- We started with polynomial regression, which is a flexible smoother but is global and therefore overall sensitive to changes in the data.
- Then we considered regression based on step functions, which has the advantage of being local, but is not smooth.
- We have introduced piecewise polynomials. These achieve local smoothness, but can result in strange-looking discontinuous curves.
- We started to add constraints, which ensure continuity and smoothness, leading to more modern methods like cubic splines and natural splines.
- Finally, we discussed smoothing splines, which are continuous non-linear smoothers that bypass the problem of knot selection altogether.

All the non-linear models we have seen so far

- Global Polynomials
- Step Functions
- Regression Splines
- Natural Splines
- Smoothing Splines

take as input one predictor and utilise suitable transformations of the predictor (namely powers) to produce flexible curves that fit data that exhibit non-linearities.

This final chapter covers the case of multiple predictors!

### 11.2 GAMs

A *Generalised Additive Model (GAM)* is an extension of the multiple linear model, which recall is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon.$$

In order to allow for non-linear effects a GAM replaces each linear component  $\beta_j x_j$  with a *non-linear* function  $f_j(x_j)$ .

So, in this case we have the following general formula

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \dots + f_p(x_p) + \epsilon.$$

This is called an *additive* model because we estimate each  $f_j(x_j)$  for  $j = 1, \dots, p$  and then add together all of these individual contributions.

## Backfitting algorithm

A simple iterative procedure, *backfitting algorithm*, can be used to fit the model.

- We set  $\hat{\beta}_0 = \bar{y}$ , and it never changes.
- For each predictor  $j$ , fit the function  $\hat{f}_j$  to  $\{y_i - \hat{\beta}_0 - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_{i=1}^n$  using the current estimates of the other functions  $\hat{f}_k$ .
- Continue the process until the estimates  $\hat{f}_j$  stabilize.

## Flexibility

Note that because we can have a different function  $f_j$  for each  $X_j$ , GAMs are extremely flexible. So, for example a GAM may include:

- Any kind of non-linear polynomial method from the ones we have seen for continuous predictors.
- Step functions which are more appropriate for categorical predictors.
- Linear models if that seems more appropriate for some predictors.

## Example: Wage Data

GAMs are very useful as they estimate the contribution of the effects of each predictor.

## Pros and Cons

### Pros:

- Very flexible in choosing non-linear models and generalisable to different types of responses.
- Because of the additivity we can still interpret the contribution of each predictor while considering the other predictors fixed.
- GAMs can outperform linear models in terms of prediction.

### Cons:

- Additivity is convenient but it is also one of the main limitations of GAMs.
- GAMs might miss non-linear interactions among predictors. Of course, we can add manually interaction terms but ideally we would prefer a procedure which does that automatically.

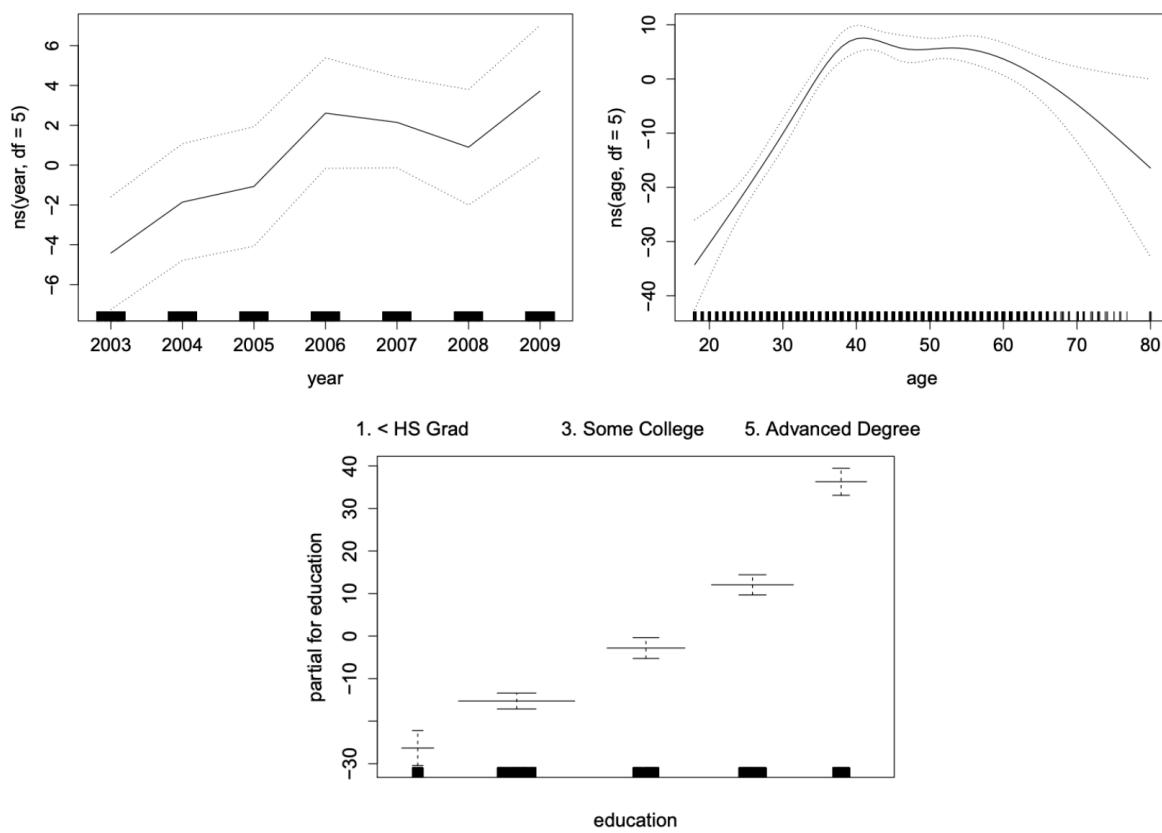


Figure 11.1: GAM using cubic splines for year and age, and step functions for education (which is categorical).

## 11.3 Practical Demonstration

In this final part we will fit a generalised additive model (GAM) utilising more than one predictor from the Boston dataset.

```
library(MASS)
y = Boston$medv
x = Boston$lstat
y.lab = 'Median Property Value'
x.lab = 'Lower Status (%)'
```

We first use the command `names()` in order to check once again the available predictor variables.

```
names(Boston)
```

```
## [1] "crim"      "zn"        "indus"     "chas"      "nox"      "rm"       "age"
## [8] "dis"       "rad"       "tax"       "ptratio"   "black"    "lstat"    "medv"
```

Let's say that we want to use predictors `lstat`, `indus` and `chas` for the analysis (use `?Boston` again to check what these refer to).

For GAMs we will make use of the library `gam` in RStudio, so the first thing that we have to do is to install this package by executing `install.packages("gam")` once. Then we load the library.

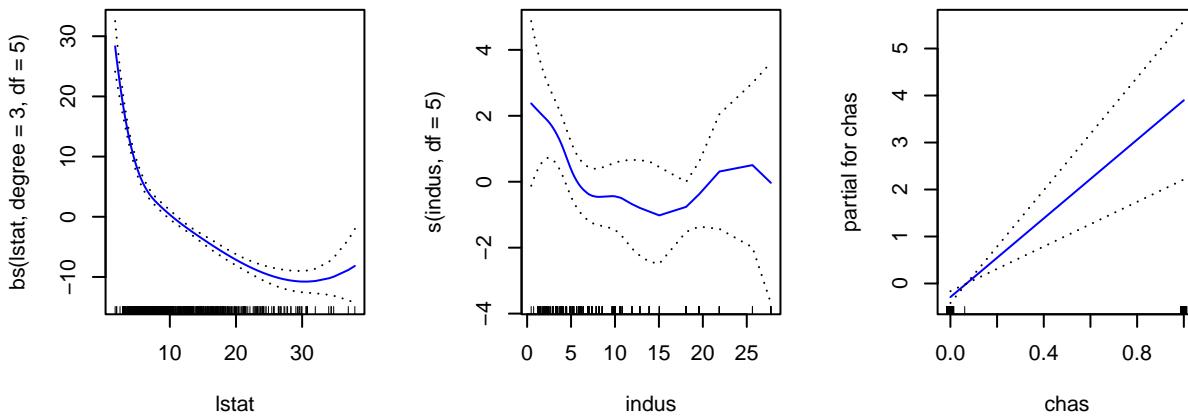
```
library(gam)
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.20
```

The main function is `gam()`. Inside this function we can use any combination of non-linear and linear modelling of the various predictors. For, example below we use a cubic spline with 3 degrees of freedom for `lstat`, a smoothing spline with 3 degrees of freedom for `indus` and a simple linear model for variable `chas`. We then plot the contributions of each predictor using the command `plot()`. As we can see, GAMs are very useful as they estimate the contribution of the effects of each predictor.

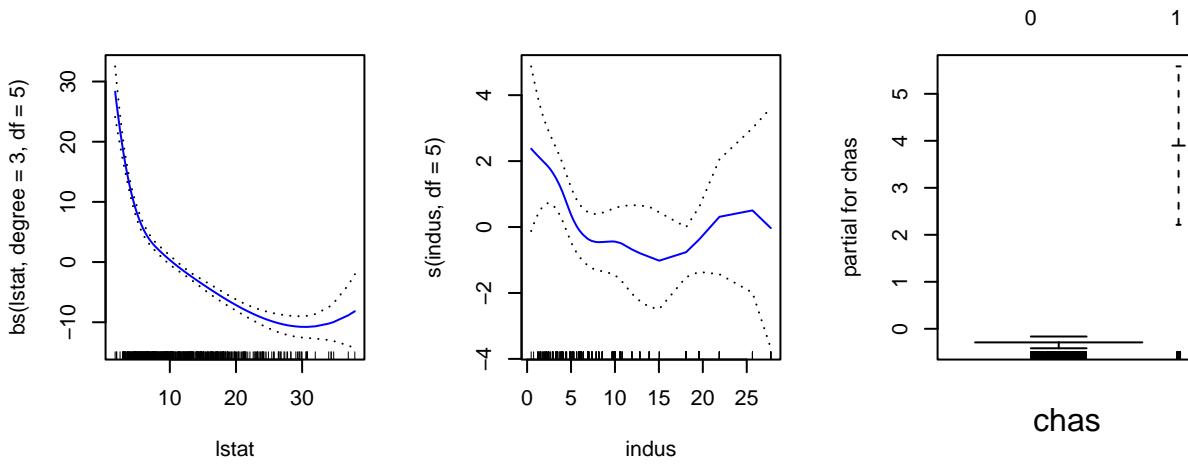
```
gam = gam( medv ~ bs(lstat, degree = 3, df = 5) + s(indus, df = 5) + chas,
           data = Boston )
par( mfrow = c(1,3) )
plot( gam, se = TRUE, col = "blue" )
```



Note that simply using `chas` inside `gam()` is just fitting a linear model for this variable. However, one thing that we observe is that variable is binary as it only takes the values of 0 and 1. This we can see from the x-axis of the `chas` plot on the right above. So, it would be preferable to use a step function for this variable. In order to do this we have to change the variable `chas` to a factor. We first create a second object called `Boston1` (in order not to change the initial dataset `Boston`) and then we use the command `factor()` to change variable `chas`. Then we fit again the same model. As we can see below now `gam()` fitted a step function for variable `chas` which is more appropriate.

```
Boston1 = Boston
Boston1$chas = factor(Boston1$chas)

gam1 = gam( medv ~ bs(lstat, degree = 3, df = 5) + s(indus, df = 5) + chas,
            data = Boston1 )
par(mfrow = c(1,3))
plot(gam1, se = TRUE, col = "blue")
```



We can make predictions from `gam` objects, just like `lm` objects, using the `predict()` method for the class `gam`. Here we make predictions on some new data. Note that when assigning the value 0 to `chas`, we enclose it in "" since we informed R to treat `chas` as a categorical factor with two levels - "0" and "1".

```
preds <- predict( gam1,
                  newdata = data.frame( chas = "0", indus = 3, lstat = 5 ) )
```

```
preds
```

```
##      1  
## 32.10065
```

# Chapter 12

## Logistic Regression

### 12.1 Motivation

In previous weeks, we have been focusing on regression problems, where the response variables are quantitative. Here we will see how we can work with qualitative response variables which leads to classification problems.

Qualitative variables take values in an unordered set. For example:

- patient status  $\in \{\text{dead, alive}\}$
- weather  $\in \{\text{rainy, cloudy, sunny}\}$

Take the case where we are trying to predict a patient's outcome. We might have access to a number of variables (quantitative or qualitative) that indicate the patient's health status, e.g. height, weight, age, gender, heart rate, consciousness.

- Quantitative variables: height, weight, age, heart rate
- Qualitative variables: gender, consciousness

The task here is to build a function that takes features  $X$  (e.g. height, age) as input and predicts the value for the response  $Y$  (i.e. dead or alive). While predicting the response itself is valuable, we often want to estimate the probability that  $X$  maps to one of the potential categories.

For example, if we are looking at insurance fraud, estimating the probability that a claim is fraudulent is often more valuable to a bank than the actual classification of the event in fraudulent or not. The same applies if we are looking at health outcomes, say for example, whether someone is likely to have cancer or not in the future. While life threatening, estimating probabilities and consequently the risk associated to a patient (ideally over time) is more valuable than asserting their future state.

Example: The Default data set

Here we will look at the Default dataset that contains information on credit card balance and income for 10,000 people, and also whether they have defaulted and whether they are a student or not.

```
library("ISLR2")

##
## Attaching package: 'ISLR2'

## The following object is masked _by_ '.GlobalEnv':
##
##      Hitters

## The following object is masked from 'package:MASS':
##
##      Boston

## The following objects are masked from 'package:ISLR':
##
##      Auto, Credit

data(Default)
head(Default)
```

```
##   default student    balance    income
## 1      No        No  729.5265 44361.625
## 2      No       Yes  817.1804 12106.135
## 3      No        No 1073.5492 31767.139
## 4      No        No  529.2506 35704.494
## 5      No        No  785.6559 38463.496
## 6     Yes       Yes  919.5885  7491.559
```

It's good practice to check for missing data, here we check for missing values in the default variable:

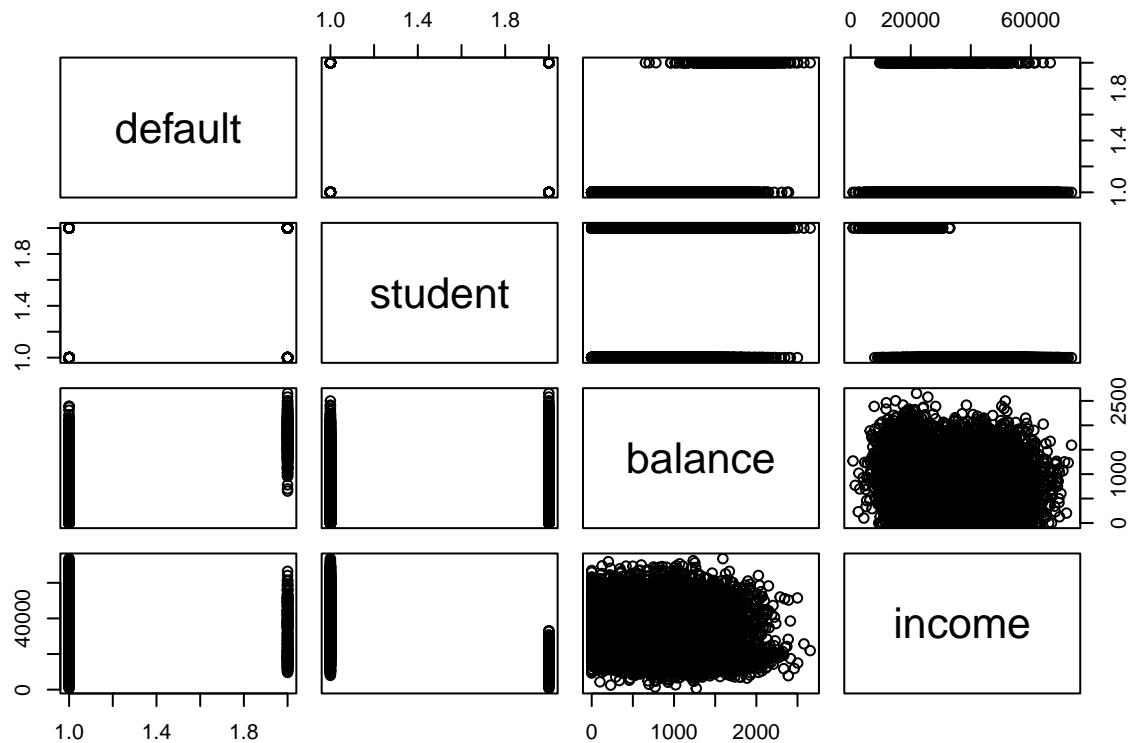
```
print(paste("Missing data:", sum(is.na(Default$default)), sep=" ", collapse=""))

## [1] "Missing data: 0"
table(Default$default, dnn="Default")

## Default
##   No  Yes
## 9667  333
```

We now produce a `pairs` plot to look at the relationship between all variables in our dataset:

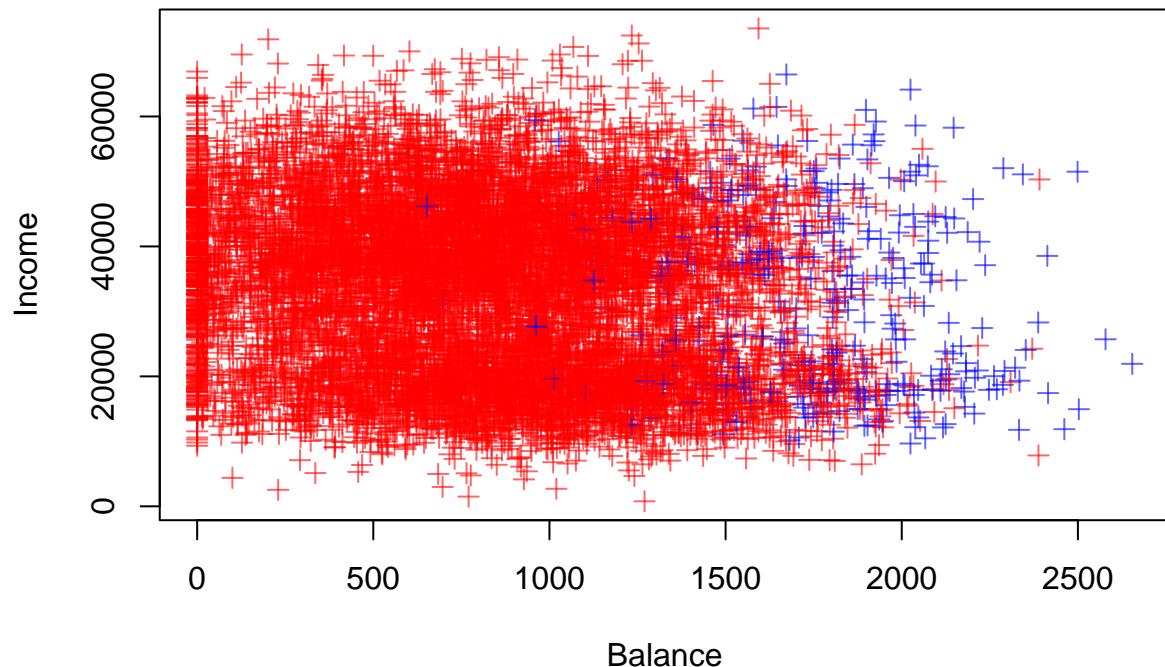
```
pairs(Default)
```



The pairs plot isn't particularly interesting but we note that `balance` and `income` are the only two quantitative predictors. Let's produce a scatter of `balance` vs `income` and colour the points by the `default` variable.

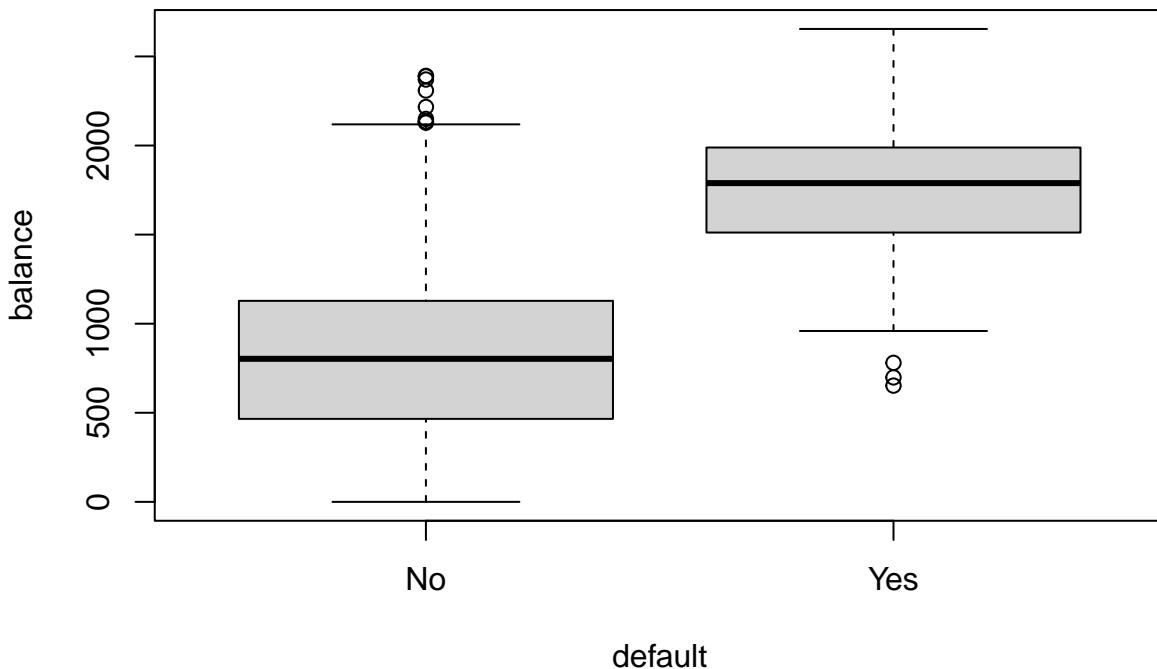
```
library(scales) #loads the alpha function to add transparency to colours
```

```
plot(Default$balance, Default$income, col=alpha(c("red","blue") [Default$default], 0.6), p
```

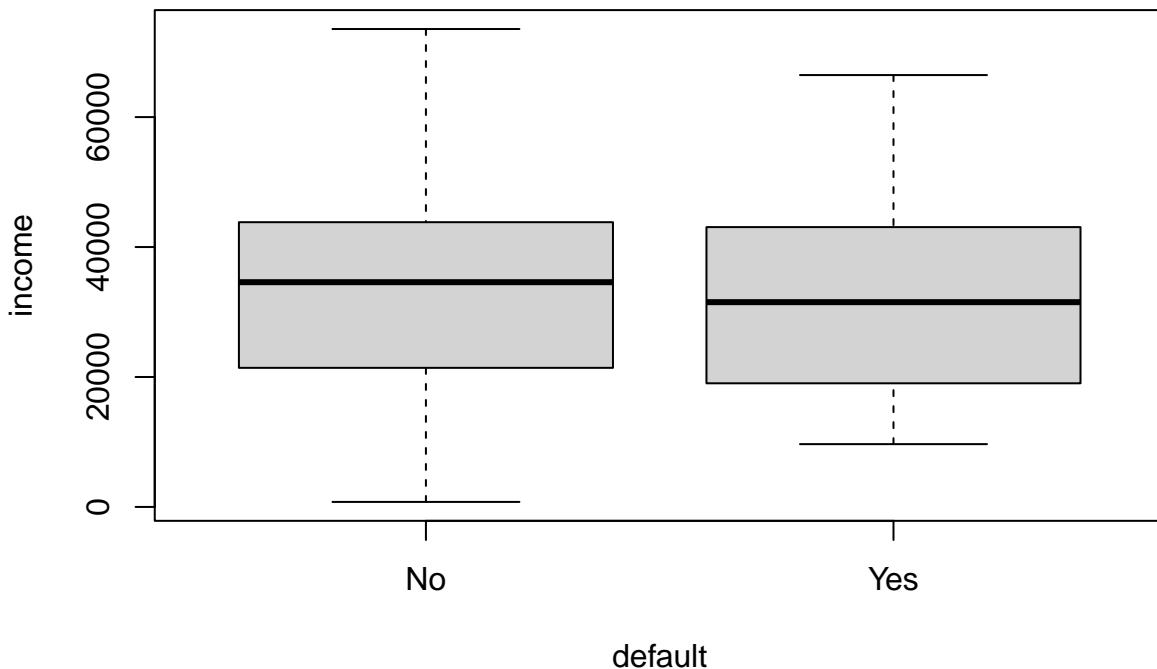


There seems to be a pattern emerging where users with a high `balance` are more likely to default. Let's inspect this using a boxplot:

```
boxplot(balance~default, data=Default)
```



```
boxplot(income~default, data=Default)
```



It seems that `balance` is likely to be a good predictor for whether someone is likely to `default` or not, but not `income`.

Let's take `default` as our response variable Y and we will rewrite it as:

$$Y = \begin{cases} 0, & \text{if } \text{No}, \\ 1, & \text{if } \text{Yes}. \end{cases}$$

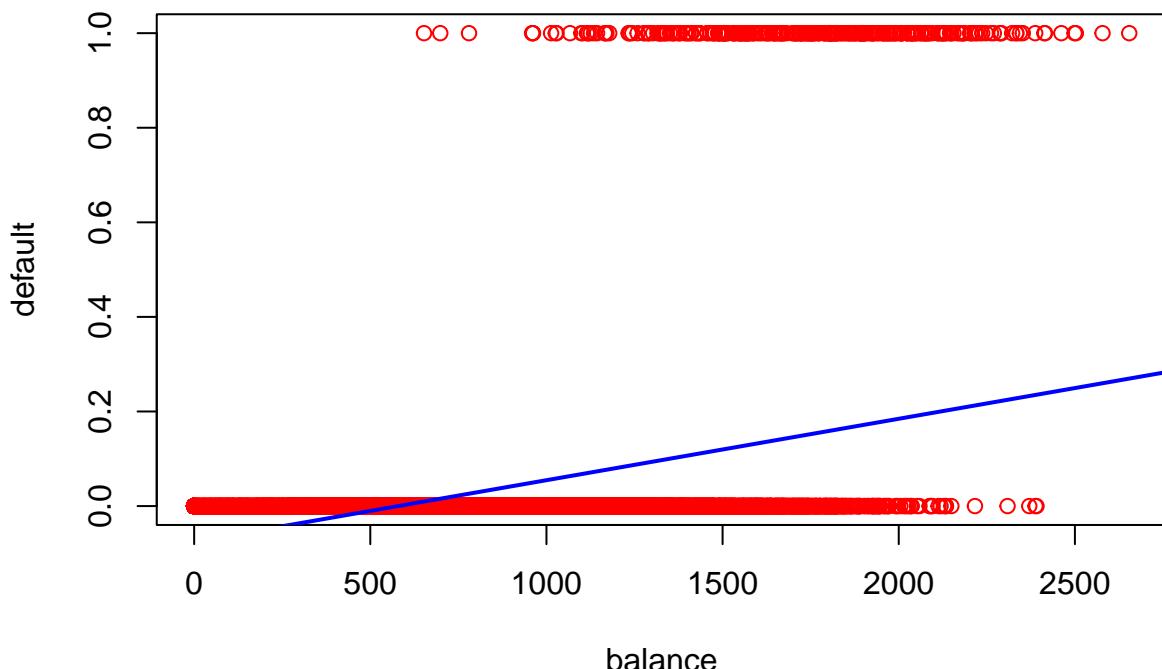
Note that by treating the response  $Y$  as a quantitative variable, we can apply the linear regression model ( $E(Y|X = x) = \beta_0 + \beta_1 x$ ):

```
lm.fit = lm(as.numeric(default == "Yes") ~ balance, data=Default)
summary(lm.fit)
```

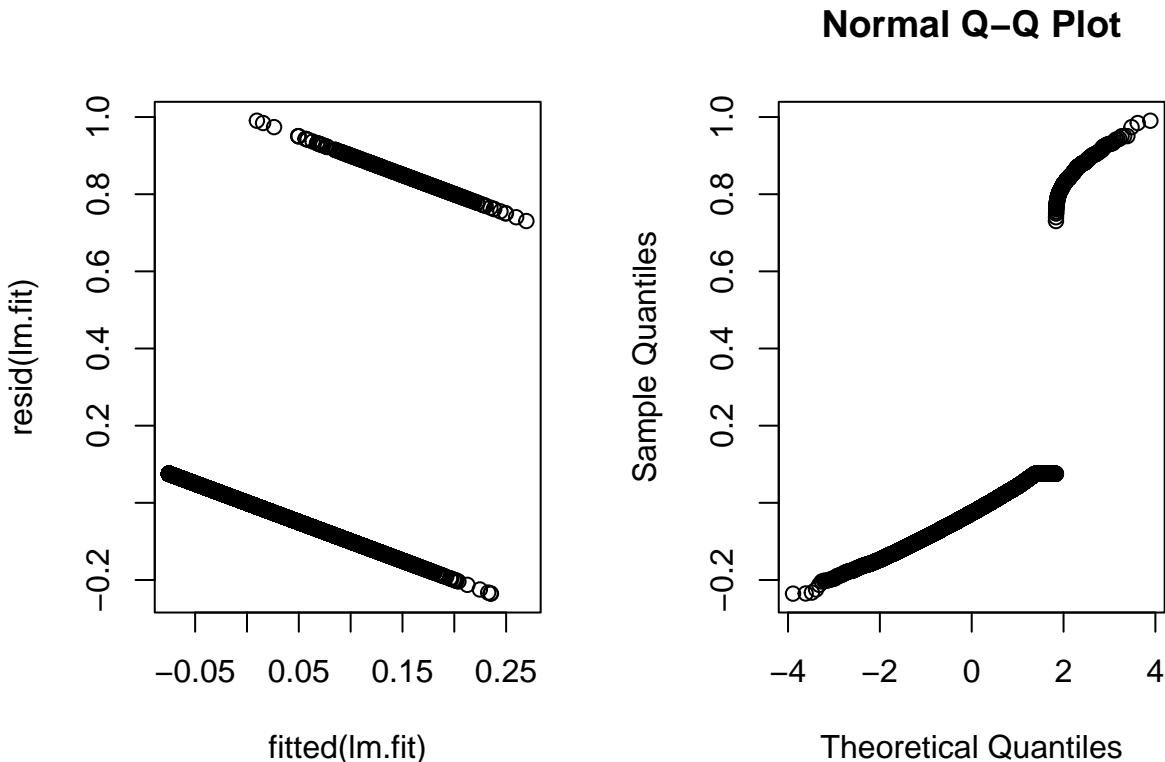
```
##
## Call:
## lm(formula = as.numeric(default == "Yes") ~ balance, data = Default)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.23533 -0.06939 -0.02628  0.02004  0.99046
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept) -0.075191959  0.003354360 -22.42 <0.000000000000002 ***
## balance      0.000129872  0.000003475   37.37 <0.000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1681 on 9998 degrees of freedom
## Multiple R-squared:  0.1226, Adjusted R-squared:  0.1225
## F-statistic: 1397 on 1 and 9998 DF,  p-value: < 0.0000000000000022
```

Let's explore it further by producing a scatter plot of `balance` vs `default` and add the regression line to the plot.

```
plot(Default$balance, as.numeric(Default$default=="Yes"), col="red", xlab="balance", ylab="default")
abline(lm.fit, col="blue", lwd = 2)
```



```
par(mfrow=c(1,2))
plot(y=resid(lm.fit), x=fitted(lm.fit))
qqnorm(resid(lm.fit))
```



```
par(mfrow=c(1,1))
```

This doesn't seem like a particularly good model...

## 12.2 Simple Logistic Regression

Note that in the Default data example,  $E(Y|X = x) = P(Y = 1|X = x)$ , which means the linear regression model is trying to predict the probability of defaulting. Unfortunately, for balances close to zero we predict a negative probability of defaulting; if we were to predict for very large balances, we would get values bigger than 1. These predictions are not sensible, since of course the true probability of defaulting, regardless of credit card balance, must fall between 0 and 1.

To avoid the inadequacies of the linear model fit on a binary response, we must model the probability of our response using a function that gives outputs between 0 and 1 for all values of  $X$ . In logistic regression, we use the logistic function to transform/map the outputs to  $(0, 1)$ .

Say  $p(X) = P(Y = 1|X)$ , we write the *logistic function*

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

This transformation would guarantee that  $p(X) \in (0, 1)$  for any  $\beta_0$ ,  $\beta_1$ , and  $X$ . Note that

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

and we can modify this to take any number of response variables. This transformation is called the *logit transformation*.

The standard approach for producing estimates  $\beta_0$  and  $\beta_1$  of the regression coefficients in the logistic regression model is to use maximum likelihood estimation. One of the advantages of this method is that maximum likelihood estimators have a number of nice theoretical properties which can be exploited to derive confidence intervals for the regression coefficients, perform hypothesis tests, and so on. In R, all this goes on under the hood when we use the `glm` function to fit the logistic regression model. When using the `glm` function, we need to set the argument `family="binomial"` to indicate that we want to fit a logistic regression model, and not some other kind of generalised linear model.

Now let's try to fit a linear logistic regression model to the Default dataset using the `glm` function in R with `family` set to `binomial`.

```
glm.fit = glm(as.numeric(Default$default=="Yes") ~ balance, data = Default, family = "bi")
```

```
##  
## Call:  
## glm(formula = as.numeric(Default$default == "Yes") ~ balance,  
##       family = "binomial", data = Default)  
##  
## Deviance Residuals:  
##      Min        1Q    Median        3Q       Max  
## -2.2697  -0.1465  -0.0589  -0.0221   3.7589  
##  
## Coefficients:  
##                 Estimate Std. Error z value          Pr(>|z|)  
## (Intercept) -10.6513306  0.3611574 -29.49 <0.0000000000000002 ***  
## balance      0.0054989  0.0002204  24.95 <0.0000000000000002 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## (Dispersion parameter for binomial family taken to be 1)  
##  
## Null deviance: 2920.6 on 9999 degrees of freedom  
## Residual deviance: 1596.5 on 9998 degrees of freedom  
## AIC: 1600.5  
##  
## Number of Fisher Scoring iterations: 8
```

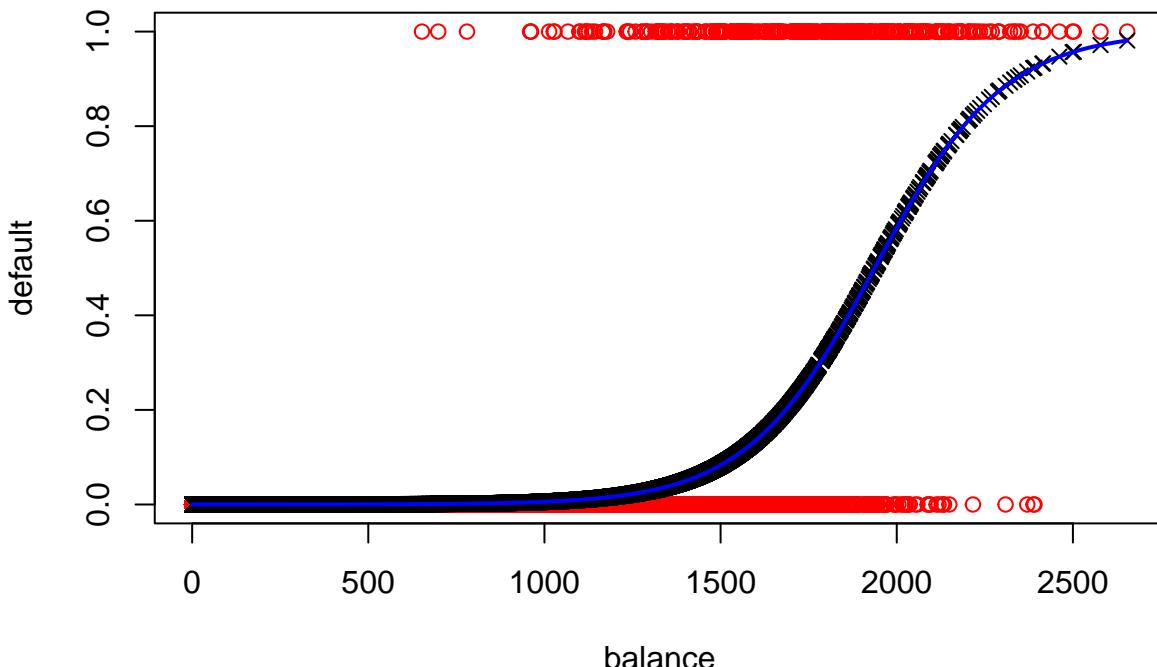
First note that now the fitted values in our regression are between 0 and 1.

```
summary(glm.fit$fitted.values)
```

```
##      Min.   1st Qu.    Median     Mean   3rd Qu.   Max.
## 0.0000237 0.0003346 0.0021888 0.0333000 0.0142324 0.9810081
```

And a plot of our model output looks far more sensible. In the plot below, we have the raw data in red, the fitted logistic curve in blue, and the fitted values in black.

```
plot(Default$balance, as.numeric(Default$default=="Yes"), col="red", xlab="balance", ylab="default")
points(glm.fit$data$balance, glm.fit$fitted.values, col = "black", pch = 4)
curve(predict(glm.fit, data.frame(balance = x), type="resp"), col="blue", lwd=2, add=TRUE)
```



From the logistic regression summary, we have estimates for  $\beta_0$  and  $\beta_1$  and we can estimate the probability of default for an individual given their balance. Say someone has a balance of £500, then

$$\hat{p}(X) = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 X)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 X)}$$

and  $\hat{p}$  equals

```
exp(sum(glm.fit$coefficients*c(1,500)))/(1+exp(sum(glm.fit$coefficients*c(1,500))))
```

```
## [1] 0.0003699132
```

Repeat the above experiment with different values for the `balance` variable. What do you see?

## 12.3 Logistic regression with several variables

We can extend the logit transformation to accommodate any number of response variables. Simply write:

$$\ln \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

and

$$p(X) = \frac{\exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p)}{1 + \exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p)}$$

And this can even be extended to the Generalised Additive Models to allow for non-linear relationships to be considered in the model by:

$$\ln \frac{p(X)}{1 - p(X)} = \beta_0 + f_1(X_1) + f_2(X_2) + \dots + f_p(X_p)$$

## 12.4 Interpreting coefficients

The interpretation of the coefficients in the logistic regression isn't as simple as in the linear regression scenario since we are predicting the probability of an outcome rather than the outcome itself.

If  $\beta_1 = 0$ , then there is no relationship between  $Y$  and  $X$ . If  $\beta_1 > 0$ ,  $X$  gets larger and so does the probability that  $Y = 1$ . If  $\beta_1 < 0$ , the opposite happens, that is, as  $X$  gets larger, the probability that  $Y = 1$  gets smaller.

## 12.5 Significance

We also want to investigate whether the coefficient values are significant. That is, we want to know whether the coefficients are statistically different from zero.

In linear regression, we use a  $t$ -test to check significance for coefficients. In logistic regression, we use a  $z$ -test (normal test) instead. Let's look at the summary of the regression:

```
summary(glm.fit) #default vs balance

##
## Call:
## glm(formula = as.numeric(Default$default == "Yes") ~ balance,
##      family = "binomial", data = Default)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.2697  -0.1465  -0.0589  -0.0221   3.7589
##
## Coefficients:
##             Estimate Std. Error z value          Pr(>|z|)
## (Intercept)  0.00000   0.00000  0.00000  0.0000000000000000
## balance     0.00000   0.00000  0.00000  0.0000000000000000
```

```
## (Intercept) -10.6513306  0.3611574  -29.49 <0.0000000000000002 ***
## balance       0.0054989  0.0002204   24.95 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1596.5  on 9998  degrees of freedom
## AIC: 1600.5
##
## Number of Fisher Scoring iterations: 8
```

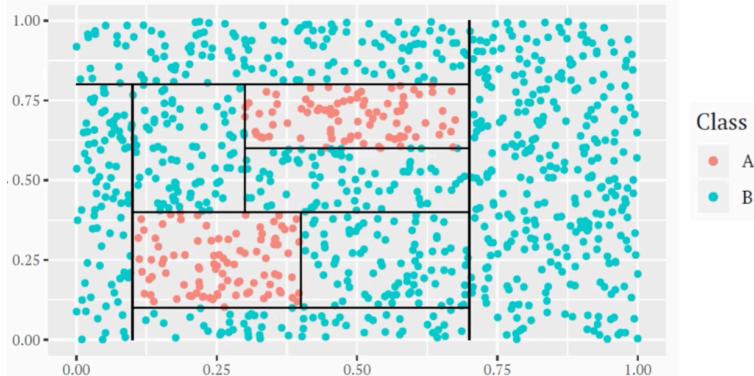
In this case, we see that `balance` is likely to be a good predictor for `default` and the *z*-test indicates that we should include this variable in our model. And as we had seen before, the higher the balance, the more likely someone is to default.

# Chapter 13

## Tree-based Models

Tree-based models are a class of *nonparametric* algorithms that work by partitioning the feature space into a number of smaller (non-overlapping) regions with similar response values using a set of splitting rules.

For example, the following figures show a tree-based classification model built on two predictors.



### Definitions

- *Nodes*: The subgroups that are formed recursively using binary partitions formed by asking simple yes-or-no questions about each feature (e.g.,  $x_1 < 0.7?$ ).
- We refer to the first subgroup at the top of the tree as the *root node* (this node contains all of the training data).
- The final subgroups at the bottom of the tree are called the *terminal nodes* or *leaves*.
- The rest of the subgroups are referred to as *internal nodes*.
- The connections between nodes are called *branches*.
- *Depth of a tree* is the length of the longest path from a root to a leaf.
- *Size of a tree* is the number of terminal nodes in the tree.

In the tree model example above, there are 8 internal nodes and 9 terminal nodes; the depth of the tree is 8 and the size of the tree is 9.

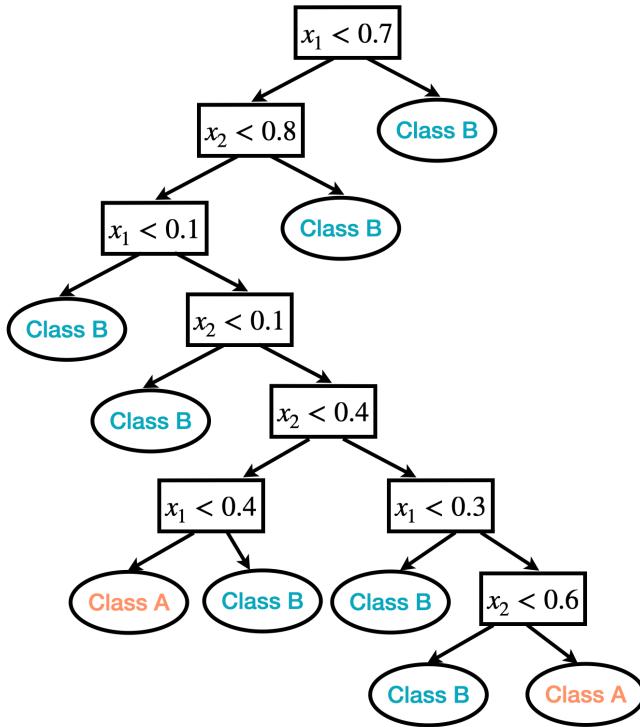


Figure 13.1: An example of classification tree based one two predictors.

As we'll see, tree-based models offer many benefits; however, classic tree-based models, classification and regression trees, typically lack in predictive performance compared to more complex algorithms like neural networks. Fortunately, we can blend powerful ensemble algorithms into the tree-based model, for example, random forests and boosting trees, to improve the predictive performance. In this chapter, we will explore various tree-based models.

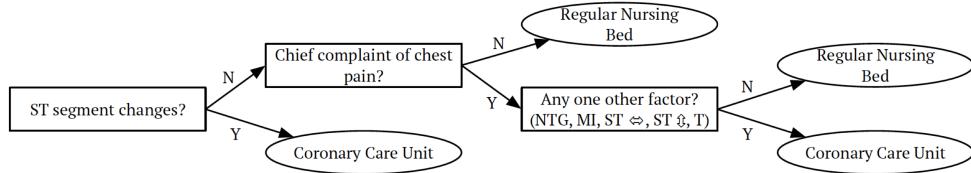
## 13.1 Classification and Regression Trees

There are many methodologies for constructing tree-based models, but the most well-known and classic is the classification and regression tree (CART) algorithm. A classification and regression tree partitions the training data into homogeneous subgroups (i.e., groups with similar response values) and then fits a simple constant in each subgroup. Mathematically, A classification and regression tree ends in a leaf node which corresponds to some hyper-rectangle in feature (predictor) space,  $R_j$ ,  $j = \{1, \dots, J\}$ , i.e. the feature space defined by  $X_1, X_2, \dots, X_p$  is divided into  $J$  distinct and non-overlapping regions.

- Regression tree, assigns a value to each  $R_j$ , that is the model predicts the output based on the average response values for all observations that fall in that subgroup.
- Classification tree, assigns a class label to each  $R_j$ , that is the model predicts the output based on the class that has majority representation or provides predicted probabilities using the proportion of each class within the subgroup.

## Interpretation

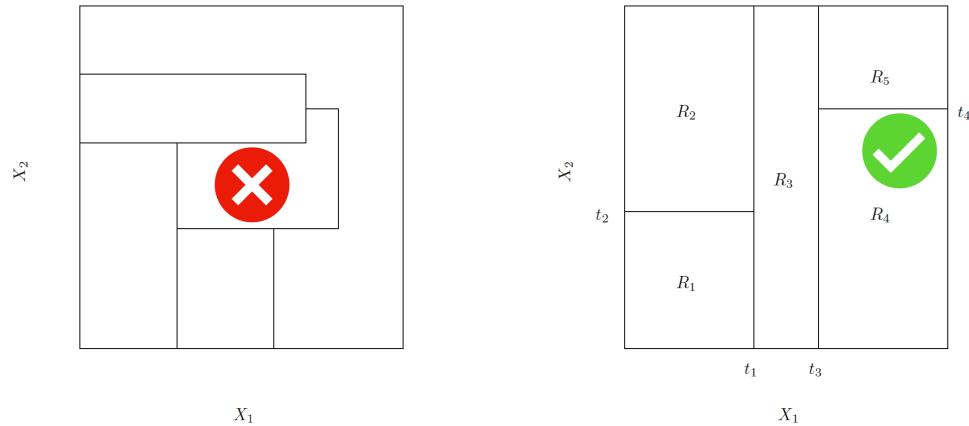
Classification trees mimic some decision making processes. For example, the following decision tree is used by A&E doctors to decide what bed type to assign:



Note that trees can be constructed even in the presence of qualitative predictor variables, for example, eye colors (blue, green, brown). As tree methods can produce simple rules that are easy to interpret and visualize with tree diagrams, tree methods often refer to decision trees, which perhaps makes them popular for the feeling of interpretability and of being like a data-learned expert system.

## Building trees

In theory, the regions  $R_j$  could have any shape. But we choose hyper (high-dimensional)-rectangles (boxes) for simplicity. The left panel is an example of an invalid partitioning, and the right is an example of a valid one.



## Performance Metric

For regression trees, the goal is to find  $R_j$ ,  $j = \{1, \dots, J\}$  that minimizing the RSS

$$\sum_{j=1}^J \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where  $\hat{y}_{R_j}$  is the mean response for the training observations within the  $j$ th rectangle.

For classification trees, we may adopt classification error rate  $E = 1 - \max_k(\hat{p}_{jk})$  as performance criteria, where  $\hat{p}_{jk}$  represents the proportion of training observations in the  $j$ th region that are from the  $k$ th class. However, it turns out that the classification error rate is not sufficiently sensitive for tree-growing, especially when the data is noisy. And in practice, two other metrics are preferable:

- Gini index:  $G = \sum_{k=1}^K \hat{p}_{jk}(1 - \hat{p}_{jk})$
- Entropy:  $D = -\sum_{k=1}^K \hat{p}_{jk} \log \hat{p}_{jk}$

Note that Both functions will take on a value near zero if the  $\hat{p}_{jk}$ 's are all near zero or near one.

### Greedy fitting algorithm

It is computationally infeasible to consider every possible partition of the feature space into  $J$  boxes. Think about how many ways one could divide this 2-D space into 5 boxes. Not to mention high dimensional feature space. Therefore, in practice, we adopt the so-called *Greedy(top-down) fitting algorithm*:

1. Initialise hyper-rectangle,  $R = \mathbb{R}^d$ .
2. Find the best (based on some objective function) split on variable  $x_i$  at location  $s$ , gives  $R_1(i, s) = \{x \in R : x_i < s\}$  and  $R_2(i, s) = \{x \in R : x_i \geq s\}$ .
3. Repeat step 2. twice, once with  $R = R_1(i, s)$  and once with  $R = R_2(i, s)$

There will be some stopping rule, for example, requiring a minimum number (for example, 5) of training observations in each hyper-rectangle (box).

Note that such a greedy fitting algorithm does NOT guarantee an optimal tree since it is constructed stepwisely.

### Pruning trees

The fitting procedure described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. To balance this trade-off between variance and bias and avoid overfitting, we often prune trees by adopting the so-called weakest link pruning algorithm in practice.

The *weakest link pruning* (or cost complexity pruning) introduces a nonnegative tuning parameter  $\alpha$ , for regression trees minimising

$$\sum_{j=1}^{|T|} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|,$$

where  $|T|$  is the number of terminal nodes (size) of the tree  $T$  (for classification trees simply replace the RSS with classification objective function, e.g. Gini index). Minimisation is achieved by attempting to remove terminal nodes from the base of the tree upward. Similar to Lasso regression,  $\alpha$  is a penalty on the size (complexity) of the tree. When  $\alpha = 0$ , we will keep the whole tree and when  $\alpha = \infty$ , we will prune everything back to null tree (the forecast is simply the average of the response). And as always, we may Select  $\alpha$  using cross-validation.

### Pros and Cons

The advantages of CART are:

- Simple and easy to interpret

- Akin to common decision making processes
- Good visualisations
- Easy to handle qualitative predictors (avoid dummy variables)

The disadvantages of CART are:

- Trees tend to have high variance, small changes in the sample lead to dramatic cascading changes in the fit!
- poor prediction accuracy.

## Practical Demonstration

This practical demonstration will be based on synthetic (surrogate) datasets. As the name suggests, quite obviously, a synthetic dataset is a repository of data that is generated programmatically (artificially), it is not collected by any real-life survey or experiment. In practice, it is almost impossible to know the underlying system behind the real data. For synthetic data, however, we know exactly what is the underlying system behind the data. It provides a flexible and rich “test” environment to help us to explore a methodology, demonstrate its effectiveness and uncover its pros and cons by conducting experiments upon. For example, if we want to test a linear regression fitting algorithm, we may create a synthetic dataset using a linear regression model and pretend not to know the parameters of the model.

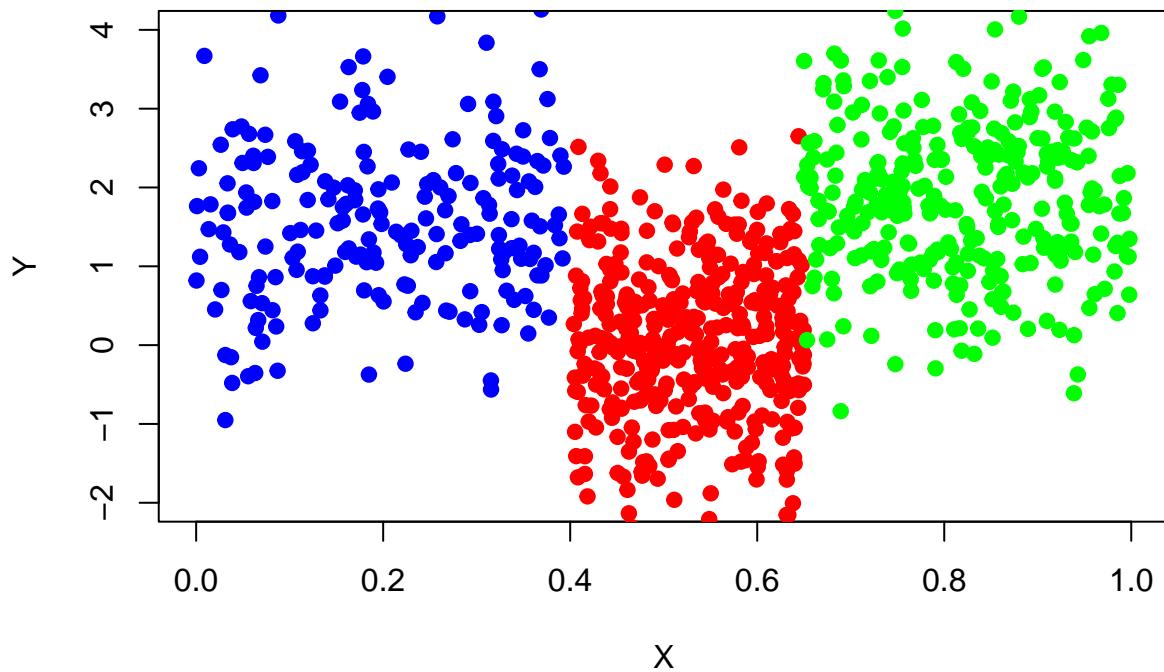
We have introduced a greedy (top-down) fitting algorithm for building classification and regression trees. Let’s create a synthetic dataset to explore the algorithm.

```
set.seed(212)
#set.seed() function is used to set the random generator state.
#So that the random data generated later can be reproduced using the same "seed".
data_surrogate <- data.frame(x = c(runif(200, 0, 0.4), runif(400, 0.4, 0.65), runif(300,
                                         y = c(rnorm(200, 1.5), rnorm(400, 0), rnorm(300, 2)))
```

In this synthetic dataset we created,  $x$  and  $y$  are paired, for the first 200 observations,  $x$  is drawn uniformly between 0 and 0.4 and  $y$  is drawn from a normal distribution with mean 1.5 and standard deviation 1; for the next 400 observations,  $x$  is drawn uniformly between 0.4 and 0.65 and  $y$  is drawn from a normal distribution with mean 0 and standard deviation 1; and for the last 300 observations,  $x$  is drawn uniformly between 0.65 and 1 and  $y$  is drawn from a normal distribution with mean 2 and standard deviation 1.

Here is a visualization of the data set.

```
plot(x=data_surrogate$x[1:200], y=data_surrogate$y[1:200], col='blue',
      xlab="X", ylab="Y", pch=19, xlim=c(0,1), ylim=c(-2,4))
points(x=data_surrogate$x[201:600], y=data_surrogate$y[201:600], col='red', pch=19)
points(x=data_surrogate$x[601:900], y=data_surrogate$y[601:900], col='green', pch=19)
```



Given this data set and knowing how it was generated, the best tree-based model we would expect when modelling variable  $y$  using the variable  $x$  is a regression tree with three terminal nodes, when  $x$  is between 0 and 0.4,  $y$  equals 1.5; when  $x$  is between 0.4 and 0.65,  $y$  equals 0;  $x$  is between 0.65 and 1,  $y$  equals 2.

Now let's build a regression tree model for this dataset using R.

```
library("tree")
```

You may need to install the “tree” package first, using `install.packages("tree")`.

```
tree_fit=tree(y~x,data_surrogate)
```

The “tree” function is to fit a classification or regression tree using the greedy fitting algorithm.

```
summary(tree_fit)
```

```
##  
## Regression tree:  
## tree(formula = y ~ x, data = data_surrogate)  
## Number of terminal nodes: 3  
## Residual mean deviance: 1.001 = 897.9 / 897  
## Distribution of residuals:  
##      Min.    1st Qu.     Median      Mean    3rd Qu.      Max.  
## -3.137000 -0.662000 -0.001074  0.000000  0.638100  2.984000
```

*#The “summary” function prints a summary of the fitted tree object.*

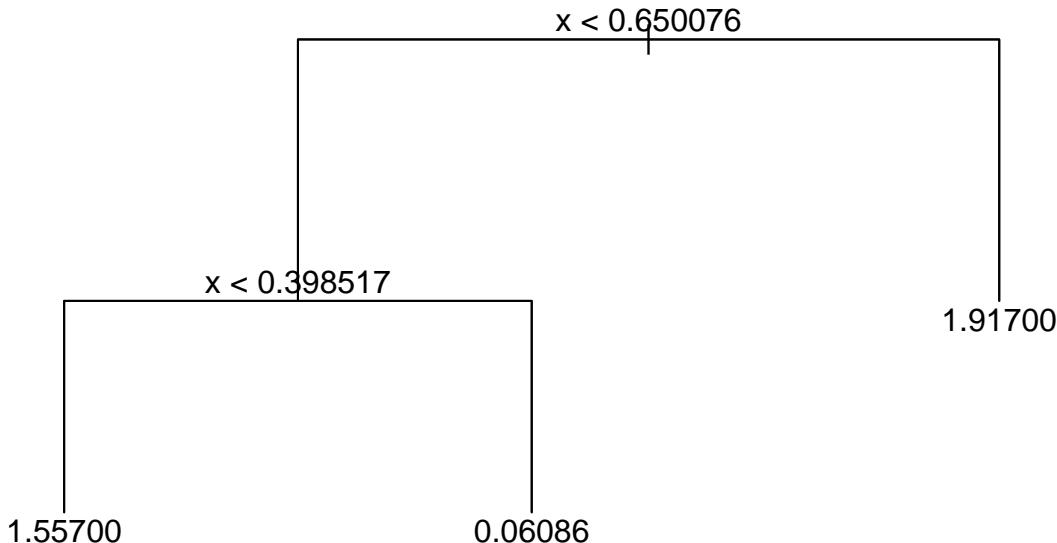
```
tree_fit
```

```
## node), split, n, deviance, yval  
## * denotes terminal node
```

```
##  
## 1) root 900 1565.0 1.01200  
##   2) x < 0.650076 600  883.8 0.55940  
##     4) x < 0.398517 200  194.1 1.55700 *  
##     5) x > 0.398517 400  391.4 0.06086 *  
##   3) x > 0.650076 300  312.4 1.91700 *  
  
#print each node as well as the corresponding statistics
```

The deviance is simply the residual sum of squares (RSS) for the tree/subtree.

```
plot(tree_fit)  
text(tree_fit, pretty=0)
```



*#If pretty = 0 then the level names of a factor split attributes are used unchanged.*

This is a nice way to plot the tree model. Is it consistent with our previous expectations?

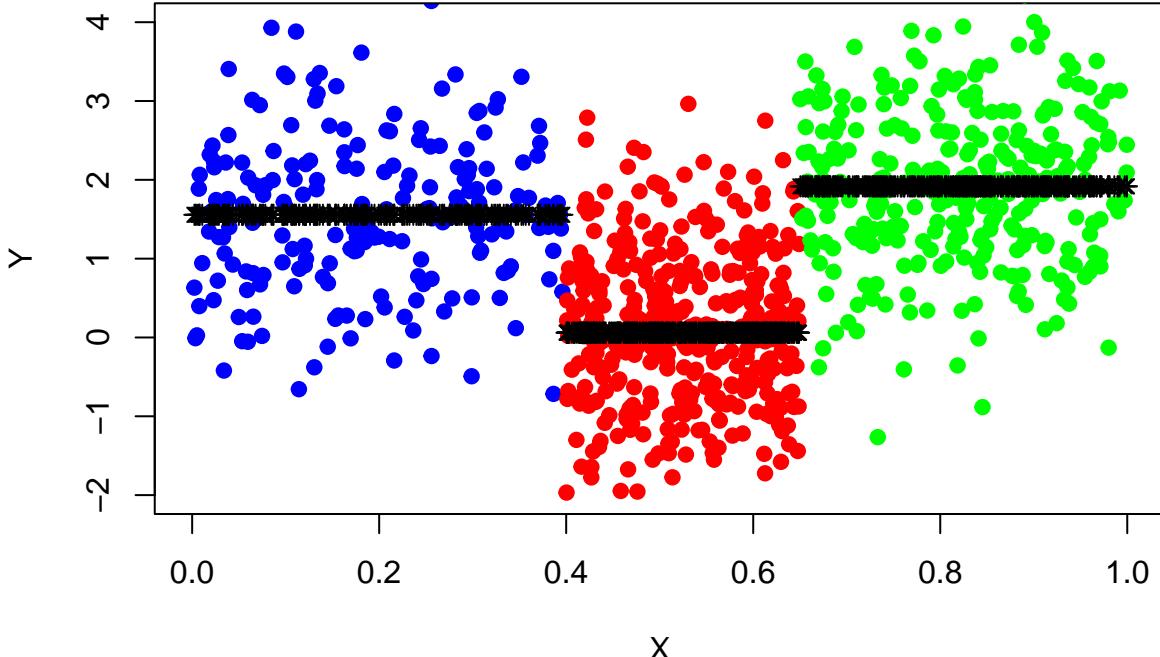
Clearly, one of the advantages of the regression and classification tree model is its interpretation.

Now let's examine the model performance by creating an independent data set (the same way that data\_surrogate was created) as a test set, naming it data\_surrogate\_test and using `tree_pred=predict(tree_fit,data_surrogate_test)` to generate the model prediction of y variable (in the test set) using the tree model fitted early.

```
set.seed(347)  
data_surrogate_test <- data.frame(x = c(runif(200, 0, 0.4), runif(400, 0.4, 0.65), runif(400, 0.65, 1)),  
                                    y = c(rnorm(200, 1.5), rnorm(400, 0), rnorm(400, 1.5)),  
                                    tree_pred=predict(tree_fit,data_surrogate_test))
```

Draw a scatter plot (same as the one drawn for data set “data\_surrogate”) for the test set and add the prediction of y, “tree\_pred”, to the plot. And calculate the residual sum of squares.

```
plot(x=data_surrogate_test$x[1:200], y=data_surrogate_test$y[1:200],
      col='blue', xlab="X", ylab="Y", pch=19, xlim=c(0,1), ylim=c(-2,4))
points(x=data_surrogate_test$x[201:600], y=data_surrogate_test$y[201:600], col='red', pch=19)
points(x=data_surrogate_test$x[601:900], y=data_surrogate_test$y[601:900], col='green', pch=19)
points(x=data_surrogate_test$x, y=tree_pred, col='black', pch=8)
```



```
pred_mse=mean((data_surrogate_test$y-tree_pred)^2)
pred_mse
```

```
## [1] 1.005645
```

So far, everything seems all right! The regression tree works almost perfectly. Note that we have used a rather large data set as a train set to fit the tree, what if the historical data is rather small?

Create an independent training data set (the same way that `data_surrogate` was created) but with the number of observations 10 times smaller. And draw a scatter plot of the data.

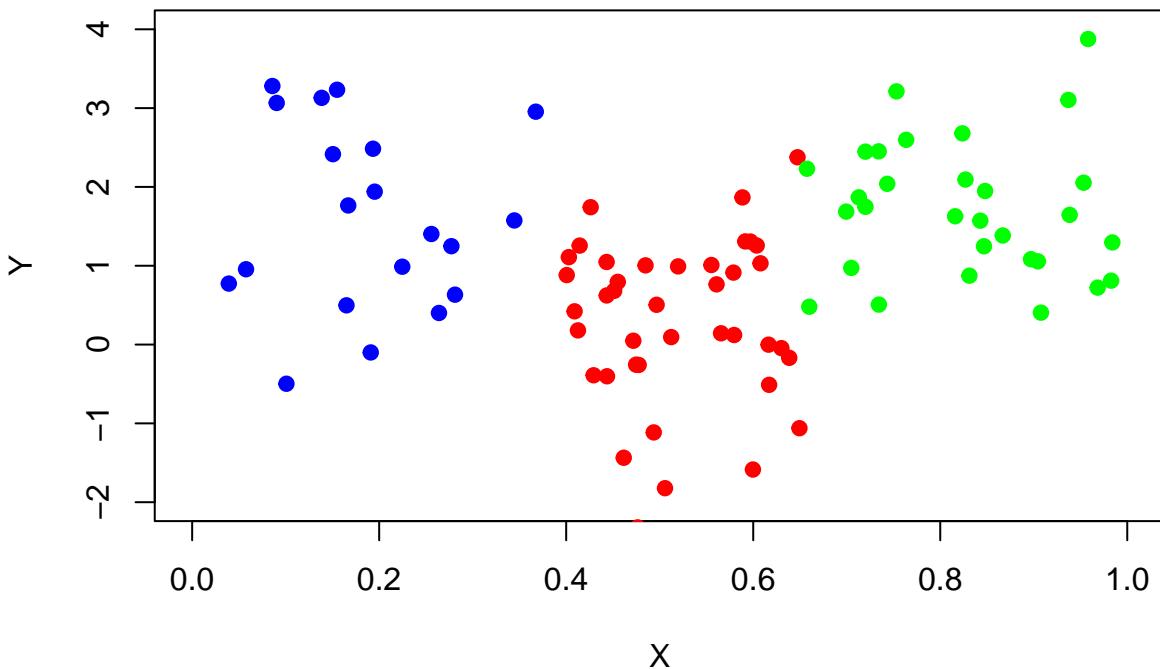
Then build a regression tree, name “`tree_fit_s`”, using the new small training set and use the new tree model to predict the `y` variable in the same test set `data_surrogate_test` and record the residual sum of squares and compare it with the one you calculated before using the tree model fitted with the large training set.

---

```
=====
set.seed(739)

data_surrogate_s <- data.frame(x = c(runif(20, 0, 0.4), runif(40, 0.4, 0.65),
                                 runif(30, 0.65, 1)), y = c(rnorm(20, 1.5), rnorm(40, 0.2, 0.5)))
plot(x=data_surrogate_s$x[1:20], y=data_surrogate_s$y[1:20], col='blue',
      xlab="X", ylab="Y", pch=19, xlim=c(0,1), ylim=c(-2,4))
```

```
points(x=data_surrogate_s$x[21:60], y=data_surrogate_s$y[21:60], col='red', pch=19)
points(x=data_surrogate_s$x[61:90], y=data_surrogate_s$y[61:90], col='green', pch=19)
```



```
tree_fit_s=tree(y~x,data_surrogate_s)
summary(tree_fit_s)
```

```
##
## Regression tree:
## tree(formula = y ~ x, data = data_surrogate_s)
## Number of terminal nodes:  7
## Residual mean deviance:  0.89 = 73.87 / 83
## Distribution of residuals:
##      Min. 1st Qu. Median 3rd Qu. Max.
## -2.5420 -0.7364  0.0396  0.0000  0.5973  2.2330
tree_fit_s
```

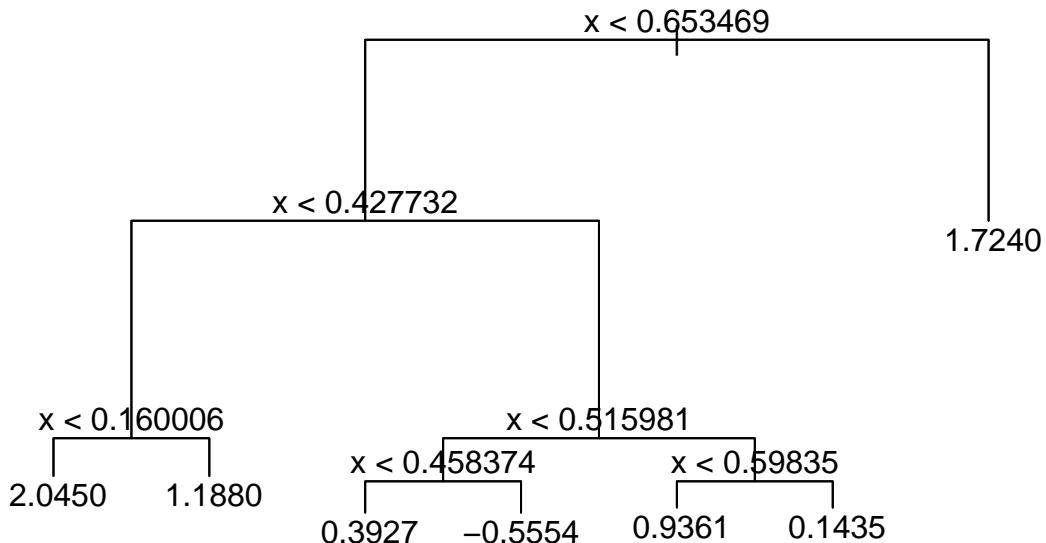
```
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 90 131.600  1.0660
##   2) x < 0.653469 60  90.670  0.7376
##     4) x < 0.427732 26  29.650  1.4510
##       8) x < 0.160006 8   14.560  2.0450 *
##       9) x >  0.160006 18  11.020  1.1880 *
##     5) x >  0.427732 34  37.640  0.1917
##     10) x < 0.515981 16  15.670 -0.1998
##       20) x < 0.458374 6   1.973  0.3927 *
##       21) x >  0.458374 10  10.330 -0.5554 *
##     11) x >  0.515981 18  17.340  0.5398
```

```

##      22) x < 0.59835 9    2.474  0.9361 *
##      23) x > 0.59835 9   12.040  0.1435 *
##      3) x > 0.653469 30  21.480  1.7240 *

plot(tree_fit_s)
text(tree_fit_s, pretty=0)

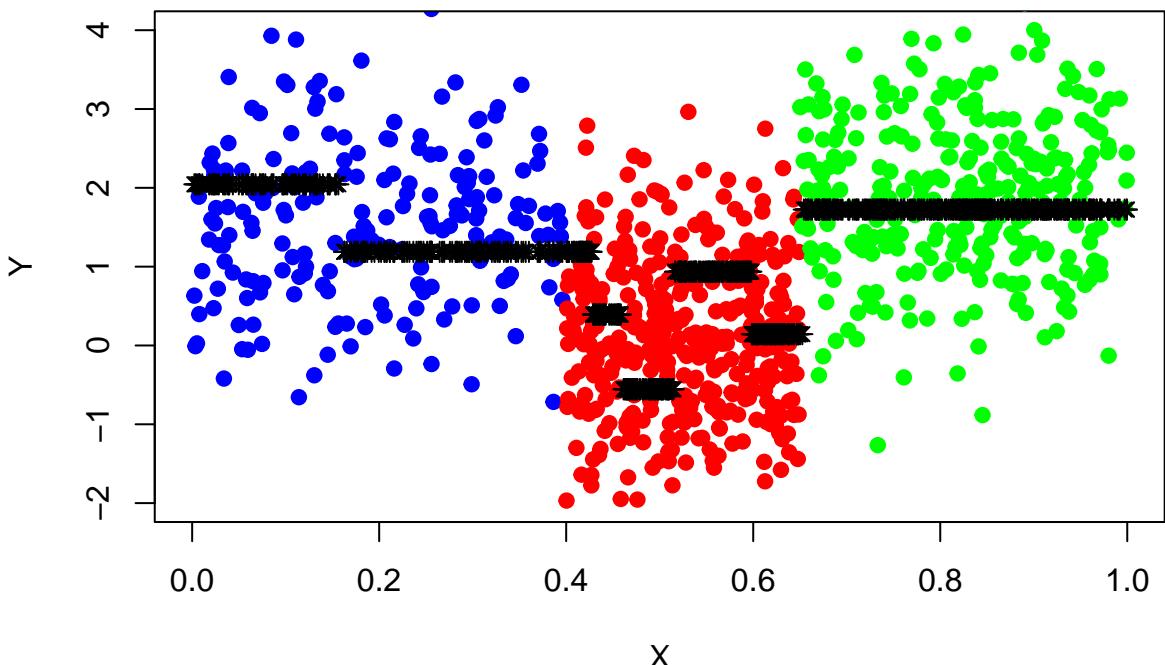
```



```

tree_pred_s=predict(tree_fit_s,data_surrogate_test)
plot(x=data_surrogate_test$x[1:200], y=data_surrogate_test$y[1:200],
     col='blue', xlab="X", ylab="Y", pch=19, xlim=c(0,1), ylim=c(-2,4))
points(x=data_surrogate_test$x[201:600], y=data_surrogate_test$y[201:600], col='red', pch=19)
points(x=data_surrogate_test$x[601:900], y=data_surrogate_test$y[601:900], col='green', pch=19)
points(x=data_surrogate_test$x,y=tree_pred_s,col='black',pch=8)

```



```

pred_mse=mean((data_surrogate_test$y-tree_pred_s)^2)
pred_mse

## [1] 1.296465

```

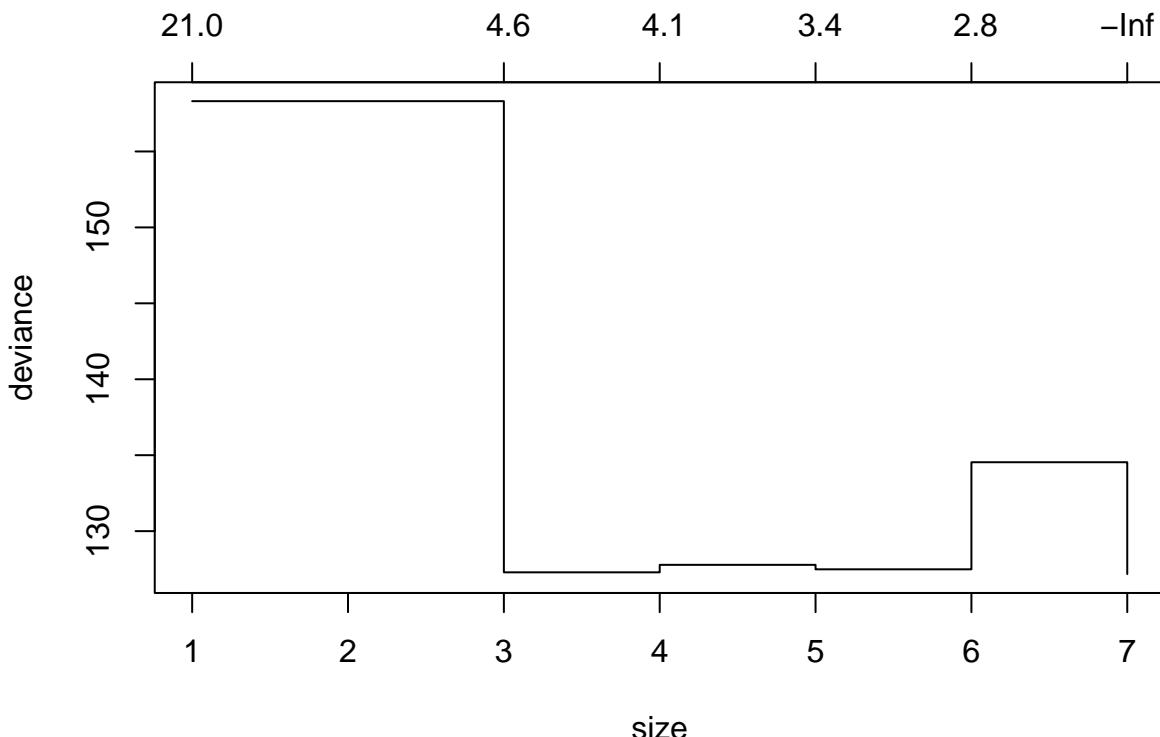
The newly fitted tree, “tree\_fit\_s”, seems to overfit the data, which led to poor forecast performance in the test set. Now we use the `cv.tree()` function to see whether pruning the tree using the weakest link algorithm will improve performance.

```

tree_cv_prune=cv.tree(tree_fit_s,FUN=prune.tree)
tree_cv_prune

## $size
## [1] 7 6 5 4 3 1
##
## $dev
## [1] 127.1730 134.5368 127.4856 127.7807 127.2861 158.3113
##
## $k
## [1]      -Inf  2.826888  3.370479  4.065974  4.633948 21.419490
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"          "tree.sequence"
plot(tree_cv_prune)

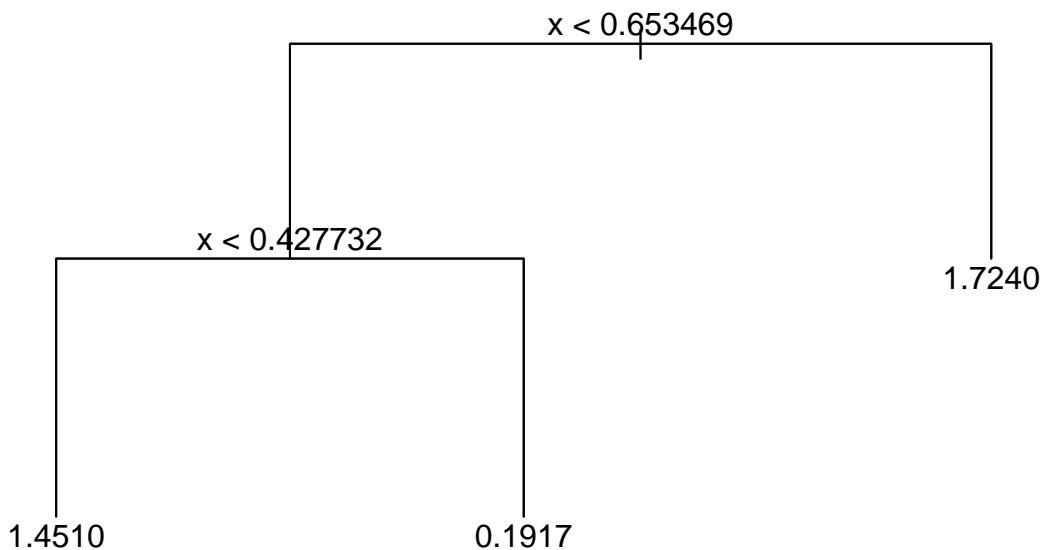
```



Note the output k in “tree\_cv\_prune” is the cost-complexity parameter in the weakest link pruning introduced in the lecture ( $\alpha$ ).

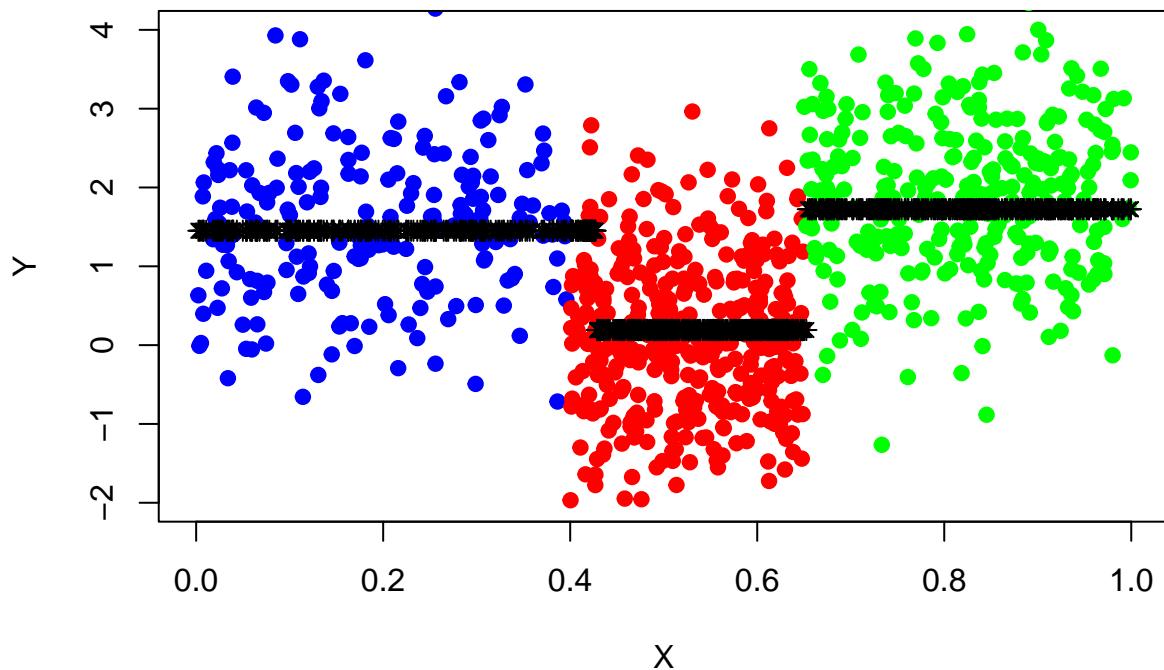
```
tree_fit_prune=prune.tree(tree_fit_s,best=3)
# you may also prune the tree by specifying the cost-complexity parameter k
# for example tree_fit_prune=prune.tree(tree_fit_s,k=5)
tree_fit_prune
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 90 131.60 1.0660
##   2) x < 0.653469 60  90.67 0.7376
##     4) x < 0.427732 26  29.65 1.4510 *
##     5) x > 0.427732 34  37.64 0.1917 *
##   3) x > 0.653469 30  21.48 1.7240 *
plot(tree_fit_prune)
text(tree_fit_prune,pretty=0)
```



Use the pruned tree model to predict the y variable in the test set and record the mean squared error and compare the mean squared error with that resulted from an unpruned tree.

```
tree_pred_prune=predict(tree_fit_prune,data_surrogate_test)
plot(x=data_surrogate_test$x[1:200], y=data_surrogate_test$y[1:200],
     col='blue', xlab="X", ylab="Y", pch=19, xlim=c(0,1), ylim=c(-2,4))
points(x=data_surrogate_test$x[201:600], y=data_surrogate_test$y[201:600], col='red', pch=19)
points(x=data_surrogate_test$x[601:900], y=data_surrogate_test$y[601:900], col='green', pch=19)
points(x=data_surrogate_test$x, y=tree_pred_prune, col='black', pch=8)
```



```
pred_mse_prune=mean((data_surrogate_test$y-tree_pred_prune)^2)
pred_mse_prune
```

```
## [1] 1.119899
```

```
pred_mse-pred_mse_prune
```

```
## [1] 0.1765653
```

How would we know that the improved forecast performance is due to pruning rather than the “unlucky” small training set? To demonstrate the robustness of the conclusion, one can conduct the same experiments many times using an independent training set.

First, create a function named “prune\_improvement(k)” that creates an independent small training set and compares pruned and unpruned trees with cost-complexity parameter as input parameter and returns the difference in prediction mean squared error. Then run the function 1024 times and calculate the average improvement in prediction mean squared error.

```
=====
prune_improvement <- function(k) {
  data_surrogate_s <- data.frame(x = c(runif(20, 0, 0.4), runif(40, 0.4, 0.65), runif(30,
  tree_fit_s=tree(y~x,data_surrogate_s)
  tree_pred_s=predict(tree_fit_s,data_surrogate_test)
  pred_mse_s=mean((data_surrogate_test$y-tree_pred_s)^2)
  tree_fit_prune=prune.tree(tree_fit_s,k=5)
  tree_pred_prune=predict(tree_fit_prune,data_surrogate_test)
  pred_mse_prune=mean((data_surrogate_test$y-tree_pred_prune)^2)
  dif_t=pred_mse_s-pred_mse_prune
  return(dif_t)
}
mean(sapply(1:1024, FUN=function(i){prune_improvement(5)}))
```

```
## [1] 0.09926117
```

## 13.2 Bagging

We have seen CART has attractive features, but biggest problem is high variance. Bootstrap aggregating (an ensemble algorithms), also called *bagging*, is designed to improve the stability and accuracy of regression and classification algorithms by model averaging. Although bagging helps to reduce variance and avoid overfitting, model interpretability will be sacrificed.

### Bootstrap

A *bootstrap* sample of size  $m$  is  $(y_i^*, x_i^*)_{i=1}^m$ , where each  $(y_i^*, x_i^*)$  is a uniformly random draw from the training data  $(y_1, x_1), \dots, (y_n, x_n)$ . A bootstrap sample is a random sample of the data taken *with replacement*, which means samples in the original data set may appear more than once in the bootstrap sample. Note that when sampling from the training data  $(y_1, x_1), \dots, (y_n, x_n)$  in pairs, relationship between  $y$  and  $x$  is reserved.

There is a nice mathematical property about bootstrap samples. Let  $(y_i^*, x_i^*)_{i=1}^m$  are an approximation to drawing IID samples from  $\mathbb{P}(X, Y)$ . If we set  $m = n$ , then notionally we sampled whole new training set. However, we will only have on average  $\approx 63.2\%$  of the original training data in the bootstrap resample.

$$\begin{aligned}
 & \mathbb{P}(\text{sample } i \text{ is in the bootstrap resample}) \\
 &= 1 - \mathbb{P}(\text{sample } i \text{ is not in the bootstrap resample}) \\
 &= 1 - \mathbb{P}[(x_1^*, y_1^*) \neq (x_i, y_i) \cap \dots \cap (x_n^*, y_n^*) \neq (x_i, y_i)] \\
 &= 1 - \prod_{j=1}^n \mathbb{P}[(x_j^*, y_j^*) \neq (x_i, y_i)] \tag{13.1} \\
 &= 1 - \left(1 - \frac{1}{n}\right)^n \\
 &\approx 1 - e^{-1} \\
 &\approx 0.632
 \end{aligned}$$

Note that we don't really imagine we have a new IID training sample! But bootstrap methods allow us to produce model-free estimates of sample distributions. If we have a lot of data then the estimate will be quite good. Intuitively we can roughly approximate the sampling distribution of trees for training sets of size  $n$ . We then can achieve variance reduction by averaging many sampled trees.

### Bagging for CART

We can apply bagging to CART in the following. For  $b = 1, \dots, B$ , draw  $n$  bootstrap samples  $(y_i^*, x_i^*)_{i=1}^n$  and each time fit a tree and get  $\hat{f}^{*b}(x)$ , the prediction at a point  $x$ .

- For regression trees, average all the predictions to obtain:

$$\hat{f}^{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

- For classification trees, we choose the class label for which the most bootstrapped trees “voted”. Or to construct a probabilistic prediction for  $j^{th}$  class by directly averaging tree output probabilities

$$\hat{f}_j^{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}_j^{*b}(x)$$

Bagging for CART address the overfitting issue by

- Grow large trees with minimal (or no) pruning. Relies on bagging procedure directly to avoid overfit.
- Prune each tree as was described in the last lecture.

Another nice property of bagging for CART is that it automatically provides out of sample evaluation. In fact, we don't need to do cross-validation for bagging, because we know on average 36.8% of original data will NOT be in bootstrap sample so can be used as a test set. Such kind of *Out Of Bag*(OOB) error estimation is approximately equal to a 3-fold cross validation.

## Pros and Cons

The advantages of bagging are:

- If you have a lot of data, bagging is a good choice because the empirical distribution will be close to the true population distribution
- Bagging is not restricted to trees, it can be used with any type of method.
- It is most useful when it is desirable to reduce the variance of a predictor. Note under the (inaccurate) independence assumption, variance will be reduced by a factor of  $\sim 1/B$  for  $B$  bootstrap resamples (Central limit theorem).
- Out of sample evaluation without using cross-validation, since any given observation will not be used in around 36.8% of the models.

The disadvantages of bagging are:

- We lose interpretability as the final estimate is not a tree.
- Computational cost is multiplied by a factor of at least  $B$ .
- Bagging trees are correlated, the more correlated random variables are, the less the variance reduction of their average.

### 13.3 Random Forests

Bagging improved on a single CART model by reducing the variance through resampling. But, in fact the bagged models are still quite correlated. And the more correlated random variables are, the less the variance reduction of their average. Can we make them more independent in order to produce a better variance reduction? Random forecasts achieve this by not only resampling observations, but by restricting the model to random subspaces,  $\mathcal{X}' \subset \mathcal{X}$ .

#### Methodology

The Random Forests algorithm can be implemented in the following:

- 1) Take a bootstrap resample  $(y_i^*, x_i^*)_{i=1}^n$  of the training data as for bagging.
- 2) Building a tree, each time a split in a tree is considered, random select  $m$  predictors out of the full set of  $p$  predictors as split candidates, and find the “optimal” split within those  $m$  predictors. (typically choose  $m \approx \sqrt{p}$ )
- 3) Repeat 1) and 2), average the prediction of all trees.

#### Interpretation

Classification and regression trees are easy to interpret and follow common human decision making mechanisms. Linear models provides statistical significance tests (e.g. t-test, z-test of parameter significance). Random Forecasts may seem great, but we’ve sacrificed interpretability with bagging and random subspace. There are two popular approaches that can help us to quantify the importance of each variable.

- 1) For each feature variable  $x_j$ ,  $j = 1, \dots, p$ , loop over each tree in the forest,
  - find all nodes that make a split on  $x_j$
  - compute the improvement in loss criterion the split causes (e.g. accuracy/Gini/etc)
  - sum improvements across nodes in the tree

Finally, sum improvement across all trees.

- 2) We already discussed that bagging enables out of bag error estimate. We can then compute out of bag variable importance for each feature  $x_j$ ,  $j = 1, \dots, p$  in the following:
  - For each tree, take the out of bag samples data matrix,  $x^{oob}$ , and compute the predictive accuracy for that tree;
  - Take  $x^{oob}$  and randomly permute all the entries  $j^{th}$  column (break the ties between  $x_j$  and the rest of variables);
  - Pass the modified  $x^{oob*}$  through the tree and compute the change in predictive accuracy for that tree.

Finally average the decrease in accuracy over all trees.

#### Pros and Cons

The advantages of random forests are:

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Works well with high dimensional data
- Tuning is rarely needed (easy to get good quality forecast)

The disadvantages of random forests are:

- Often quite suboptimal for regression.
- Same extrapolation issue as trees.
- Harder to implement and memory hungry model.

## 13.4 Boosting

Boosting is an extremely popular machine learning algorithm that has proven successful across many domains and is one of the leading methods for winning Kaggle competitions. Whereas random forests build an ensemble of deep independent trees, Boosting builds an ensemble of shallow trees in sequence with each tree learning and improving on the previous one. Although shallow trees by themselves are rather weak predictive models, they can be “boosted” to produce a powerful “committee” that, when appropriately tuned, is often hard to beat with other algorithms.

**Methodology** Boosting is similar to bagging in the sense that we combine results from multiple classifiers, but it is fundamentally different in approach:

- 1) Set  $\hat{f}(x) = 0$  and  $\epsilon_i = y_i$  for  $i = 1, \dots, n$ .
- 2) For  $b = 1, \dots, B$ , iterate:
  - i) Fit a model (eg tree)  $\hat{f}^b(x)$  to the response  $\epsilon_1, \dots, \epsilon_n$
  - ii) Update the predictive model to

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

- iii) Update the residuals
 
$$\epsilon_i \leftarrow \epsilon_i - \lambda \hat{f}^b(x) \quad \forall i$$
- 3) Output as final boosted model

$$\hat{f}(x) = \sum_{b=1}^B \lambda_b \hat{f}^b(x)$$

Intuitively Boosting is a

- Slow learning approach: Aim to learn usually simple model (which leads to low variance, but high bias) in each round. Then bring bias down by repeating over many rounds.
- Sequential learning: Each round of boosting aims to correct the error of the previous rounds.
- Generic methodology: Any model at all can be boosted! Completely general method.

## Tuning parameters for boosting

There are three main hyper-parameters that need to be tuned for building boosting tree model.

- 1) The number of trees  $B$ . Unlike bagging and random forests, boosting can overfit if  $B$  is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select  $B$ .
- 2) The shrinkage parameter  $\lambda$ , a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small  $\lambda$  can require using a very large value of  $B$  in order to achieve good performance.
- 3) The number of splits  $d$  in each tree, which controls the complexity of the boosted ensemble. Often  $d = 1$  works well, in which case each tree consists of a single split and resulting in an additive model. More generally  $d$  is the interaction depth, and controls the interaction order of the boosted model, since  $d$  splits can involve at most  $d$  variables.

**Boosting discussion** Boosting is an incredibly powerful technique and regularly is instrumental in winning machine learning competitions. Any model at all can be boosted. However, tuning is needed (not trivial) and some difficulties in interpretation and extrapolation.

## 13.5 Practical Demonstration

We now explore all the tree-based model by analyzing the Boston housing data. The goal is to build a model to predict variable `medv` using the rest of variables.

```
library(ISLR)
library(tree)
library(MASS)
```

We first divide the dataset into two part, training set (half the dataset) and test set (the other half the dataset).

```
set.seed (1)
train = sample (1: nrow(Boston ), nrow(Boston )/2)
data_train=Boston[train,]
data_test=Boston[-train,]
```

Fit a regression tree to the training set, plot the tree.

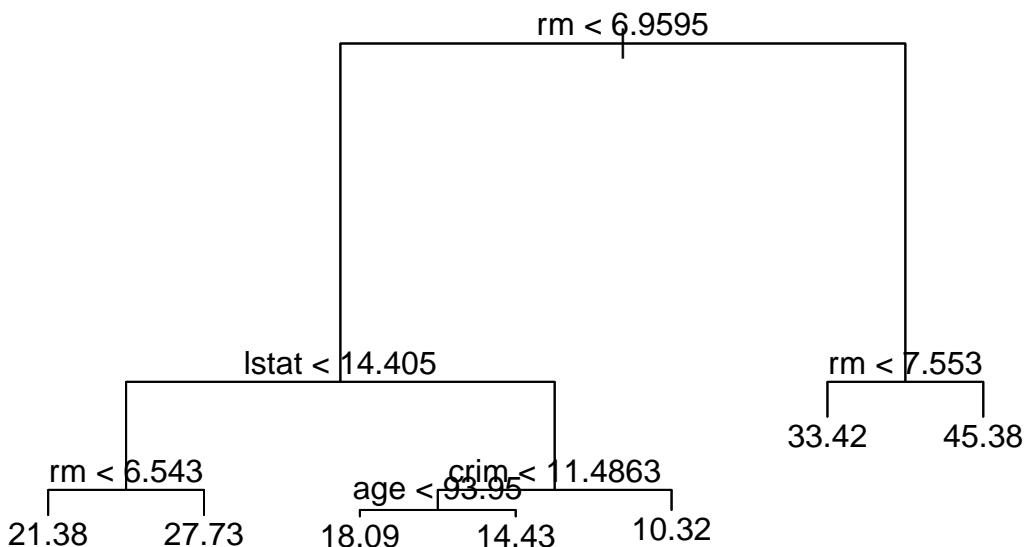
```
tree.boston=tree(medv~,data_train)
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = data_train)
## Variables actually used in tree construction:
## [1] "rm"      "lstat"   "crim"    "age"
```

```

## Number of terminal nodes: 7
## Residual mean deviance: 10.38 = 2555 / 246
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -10.1800 -1.7770 -0.1775  0.0000  1.9230 16.5800
plot(tree.boston)
text(tree.boston,pretty =0)

```



Predict `medv` using the test set and record the mean squared error.

```

yhat=predict(tree.boston,data_test)
mean((yhat -data_test$medv)^2)

```

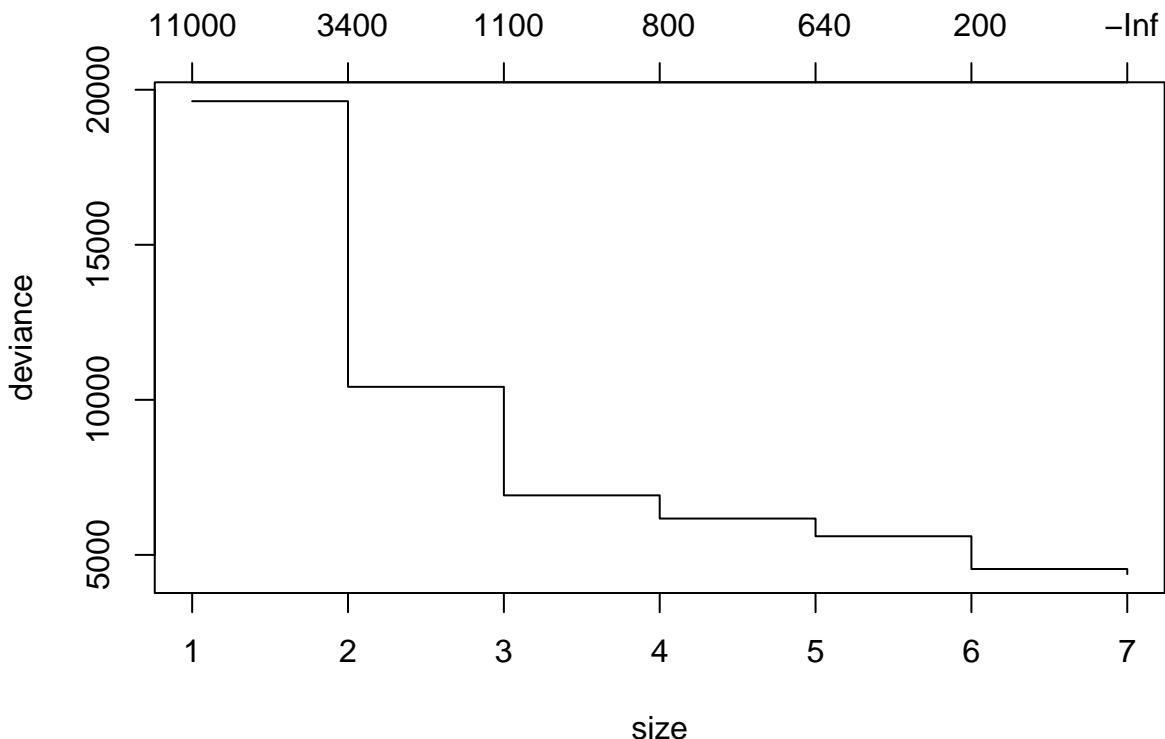
```
## [1] 35.28688
```

Prune the tree using `cv.tree()` and plot the pruned tree

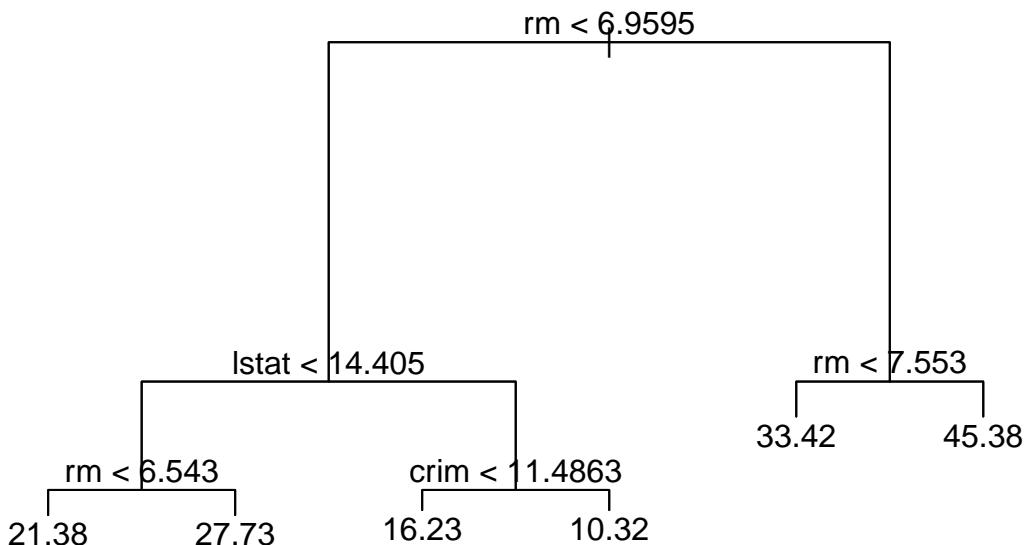
```

cv.boston =cv.tree(tree.boston)
plot(cv.boston)

```



```
prune.boston =prune.tree(tree.boston ,best =6)
plot(prune.boston)
text(prune.boston,pretty =0)
```



Predict `medv` using the test set and pruned tree, recorder the mean squared error and compare it with unpruned tree.

```
yhat=predict(prune.boston,data_test)
mean((yhat -data_test$medv)^2)
```

```
## [1] 35.16439
```

Here we apply bagging and random forests to the Boston data, using the `randomForest` package in R. Note that bagging is simply a special case of a random forest with `m` =

p. Therefore, the `randomForest()` function can be used to perform both random forests and bagging. We perform bagging as follows:

```
library (randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

## 
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
## 
##     margin

#you may need to install the randomForest library first
set.seed (472)
bag.boston =randomForest(medv~.,data=data_train,mtry=13, ntree=10, importance =TRUE)

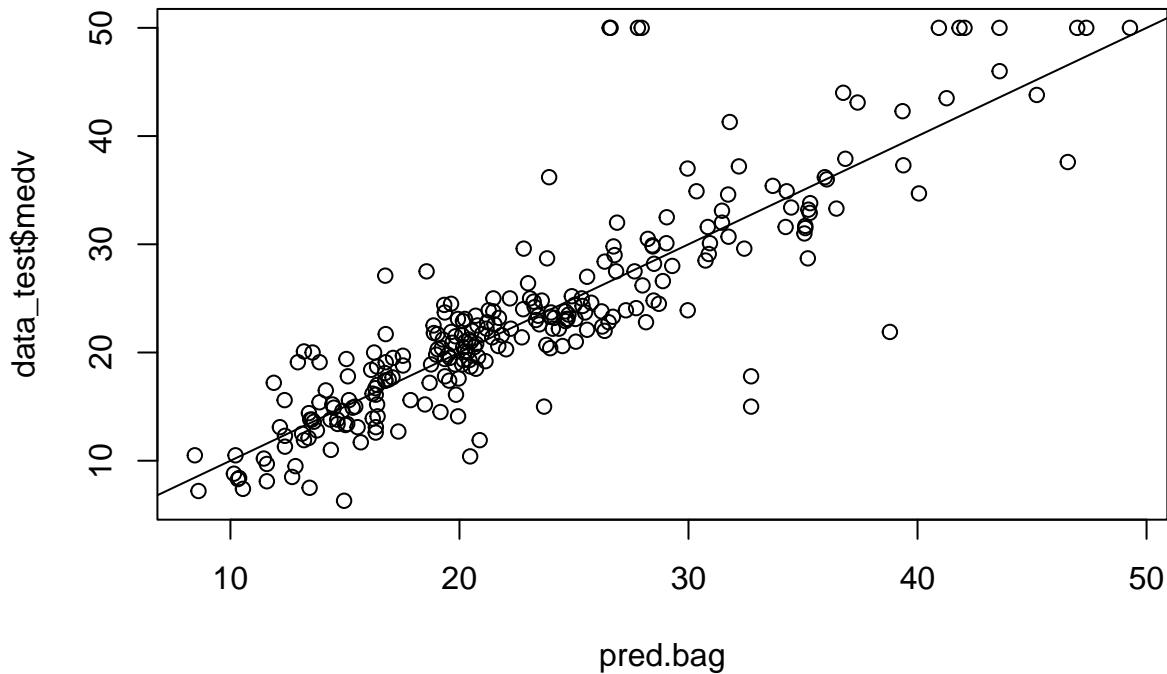
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
bag.boston

## 
## Call:
##   randomForest(formula = medv ~ ., data = data_train, mtry = 13,      ntree = 10, impo
## 
##           Type of random forest: regression
##           Number of trees: 10
## No. of variables tried at each split: 12
## 
##           Mean of squared residuals: 13.45607
##           % Var explained: 82.49
```

Note there are 13 predictors in the Boston data, the argument `mtry=13` indicates that all 13 predictors are considered for each split of the tree, in other words, bagging is conducted. `ntree=10` indicates 10 bootstrap trees are generated in this bagged model. The MSR and % variance explained are based on OOB (out-of-bag) estimates

We can also evaluate the performance of bagged model using the test set:

```
pred.bag = predict (bag.boston,newdata =data_test)
plot(pred.bag , data_test$medv)
abline (0,1)
```



```
mean(( pred.bag -data_test$medv)^2)
```

```
## [1] 22.44938
```

The test set MSE associated with the bagged regression tree is much smaller than that obtained using an optimally-pruned single CART.

We now Increase the number of bootstrap trees generated in the bagged model to 100 and 1000, record the test set MSE respectively and compare them to the bagged model with 10 bootstrap trees.

```
bag.boston =randomForest(medv~.,data=data_train,mtry=13, ntree=100, importance =TRUE)

## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range

pred.bag = predict (bag.boston,newdata =data_test)
mean(( pred.bag -data_test$medv)^2)

## [1] 23.13702

bag.boston =randomForest(medv~.,data=data_train,mtry=13, ntree=1000, importance =TRUE)

## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range

pred.bag = predict (bag.boston,newdata =data_test)
mean(( pred.bag -data_test$medv)^2)

## [1] 23.54552
```

We now Grow a random forest proceeds in exactly the same way, except that we use a smaller value of the `mtry` argument. By default, `randomForest()` uses  $p/3$  variables when building

a random forest of regression trees, we may also try  $\sqrt{p}$  variables when building a random forest of classification trees.

```
rf.boston =randomForest(medv~.,data=data_train,mtry=4, ntree=100, importance =TRUE)
pred.rf = predict (rf.boston,newdata =data_test)
mean(( pred.rf -data_test$medv)^2)
```

```
## [1] 20.41462
```

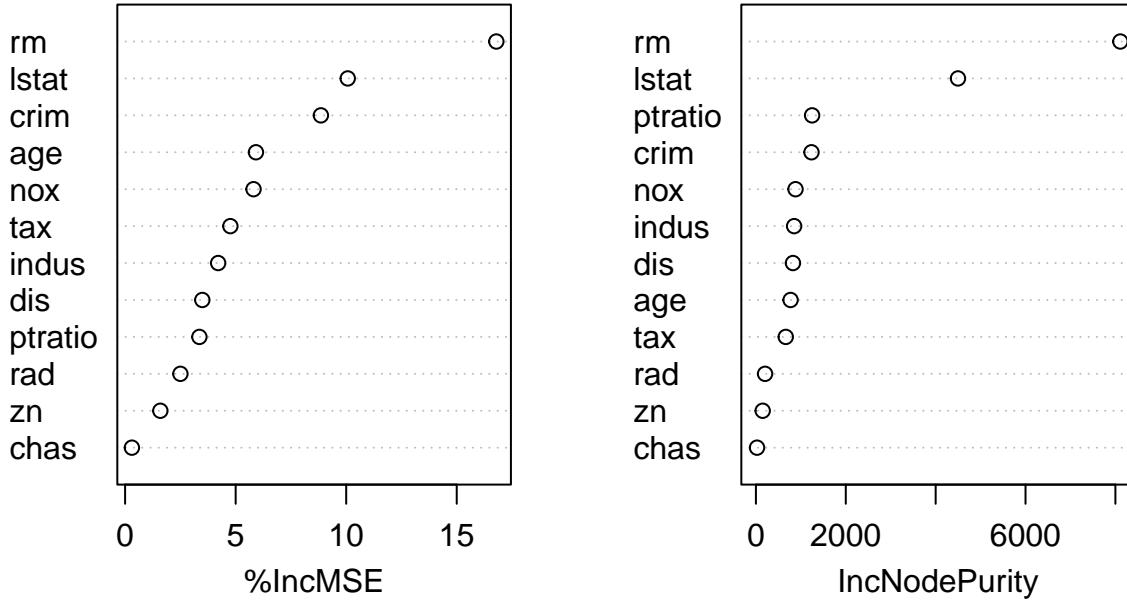
To view the importance of each variable, apply the `importance()` function. (type `?importance` in the console for detailed description of `importance()` function) And use the `varImpPlot()` to plot the importance measures and interpret the results.

```
importance(rf.boston)
```

```
##           %IncMSE IncNodePurity
## crim      8.8561695   1236.90238
## zn        1.5940368    150.70830
## indus     4.2096189    850.89234
## chas      0.3066732    24.94713
## nox       5.8090684    881.53719
## rm        16.7918585   8110.68177
## age       5.9184141    771.57077
## dis       3.4916527    824.61307
## rad       2.5007048    203.89994
## tax       4.7607576    664.49261
## ptratio   3.3599677   1250.70664
## lstat    10.0709051   4496.94612

varImpPlot(rf.boston)
```

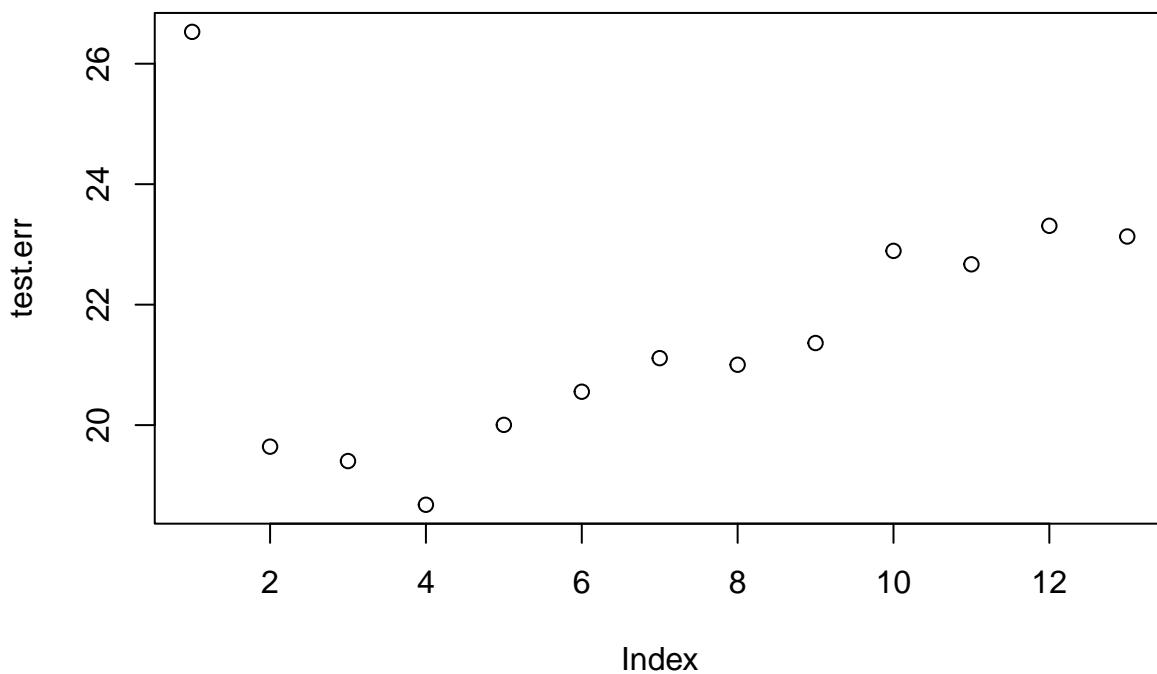
## rf.boston



We can write a `for` loop to record the test set prediction MSE for all 13 possible values of `mtry`.

```
test.err=double(13)
for(mtry_t in 1:13){
  fit=randomForest(medv~.,data=data_train,mtry=mtry_t,ntree=100)
  pred=predict(fit,data_test)
  test.err[mtry_t]=mean(( pred -data_test$medv)^2)
}
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
plot(test.err)
```

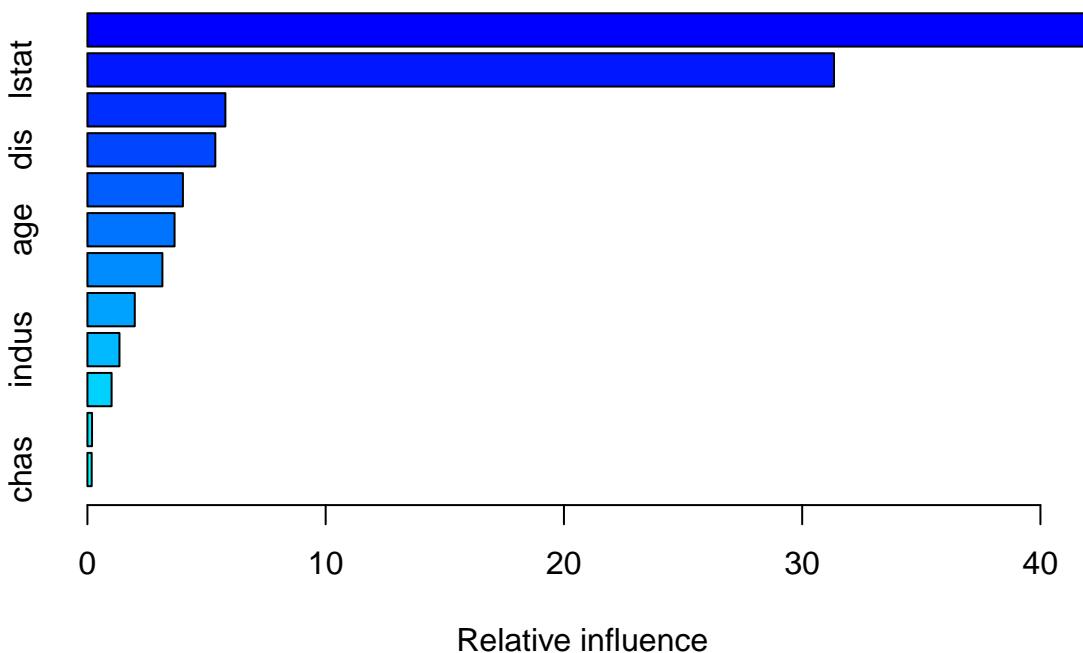


We now use the gbm package, and within it the gbm() function, to fit boosted regression trees to the Boston data set.

```
library (gbm)

## Loaded gbm 2.1.8

set.seed (517)
boost.boston =gbm(medv~.,data=data_train, distribution="gaussian",
n.trees =1000, interaction.depth =2)
summary(boost.boston)
```



```
##          var      rel.inf
```

```

## rm          rm 41.9696443
## lstat      lstat 31.3357911
## crim      crim  5.7915015
## dis        dis   5.3682171
## nox        nox   4.0082986
## age        age   3.6575000
## ptratio    ptratio 3.1472612
## tax        tax   1.9878973
## indus     indus  1.3455310
## rad        rad   1.0149034
## zn         zn    0.1938735
## chas      chas  0.1795809

```

You may look up the function gbm via ?gbm. The option distribution=“gaussian” since this is a regression problem; if it were a binary classification problem, we would use distribution=“bernoulli”. The argument n.trees=1000 indicates that we want 1000 trees, and the option interaction.depth=2 specifies the number of splits performed on each tree. The summary() function produces a relative influence plot and also outputs the relative influence statistics.

We now use the boosted model to predict medv on the test set, and record MSE. Note the boosted model outperfrom random forests model built early.

```

pred.boost=predict(boost.boston,newdata =data_test, n.trees =1000)
mean((pred.boost-data_test$medv)^2)

## [1] 15.91325

```

The default shrinkage parameter lambda is 0.001. We can specify its value by adding argument shrinkage in the gbm() function, for example shrinkage=0.05.

# Part I

## Problem Sheets



# Problem Sheets

## Problem Sheet 1

### Q1

Which of the following problem correspond to supervised learning? More than one correct answer can be chosen.

- A. Given a dataset of 60,000 small square  $28 \times 28$  pixel grayscale images of handwritten single digits between 0 and 9. Classify an image of a handwritten single digit into one of 10 classes representing integer values from 0 to 9.
- B. Find clusters of genes that interact with each other
- C. Find an investment portfolio that is likely to make a profit in the coming month
- D. Predict whether a website user will click on an ad

### Q2

Which of the following statement about parametric and non-parametric models is not true?  
Select all that apply:

- A. A parametric approach reduces the problem of estimating a functional form  $f$  down to the problem of estimating a set of parameters because it assumes a form for  $f$ .
- B. A non-parametric approach does not assume a functional form  $f$  and so does not require a large number of observations to estimate  $f$ .
- C. The advantages of a parametric approach to regression or classification are the simplifying of modelling  $f$  to a few parameters and therefore the estimations of the model parameters are more accurate compared to a non-parametric approach.
- D. The disadvantages of a parametric approach to regression or classification are a potential to inaccurately estimate  $f$  if the form of  $f$  assumed is wrong or to overfit the observations if more complex models are used.

### Q3

Why is linear regression important to understand? Select all that apply:

- A. Linear regression is very extensible and can be used to capture nonlinear effects

- B. The linear model is often correct
- C. The linear model is hardly ever a good representation of the underlying system, but it is an important piece in many more complex methods
- D. Understanding simpler methods sheds light on more complex ones

#### **Q4**

Suppose that we build a simple linear regression  $Y = \beta_0 + \beta_1 X + \epsilon$  based on  $n$  observations  $(x_1, y_1), \dots, (x_n, y_n)$ . Which of the following indicates a fairly strong relationship between  $X$  and  $Y$ ? More than one correct answer can be chosen.

- A. The  $p$ -value for the null hypothesis  $\beta_1 = 0$  is 0.0001
- B. The estimated parameters  $\hat{\beta}_1 \gg 1$  and  $\hat{\beta}_0 \approx 0$
- C.  $R^2 = 0.9$
- D. The  $t$ -statistic for the null hypothesis  $\beta_1 = 0$  is 40

#### **Q5**

In the case of simple linear regression, show that the least squares line always passes through the point  $(\bar{x}, \bar{y})$ .

#### **Q6**

Consider a simple linear regression problem, prove that under the standard four assumptions (listed in the lecture notes), maximum likelihood estimation (MLE) is equivalent to the least squares estimation (LSE) of the model parameters.

#### **Q7**

A company manufactures an electronic device to be used in a very wide temperature range. The company knows that increased temperature shortens the lifetime of the device, and a study is therefore performed in which the lifetime is determined as a function of temperature. The following data is found:

Temperature in Celcius	Lifetime in hours
10	420
20	365
30	285
40	220
50	176
60	117
70	69
80	34
90	5

Construct a linear regression model for this problem. Estimate the model parameters. Calculate the 95% confidence interval for the slope in the linear regression model.

## Q8

Consider a simple linear regression model, fit by least squares to a set of training data  $(x_1, y_1), \dots, (x_N, y_N)$  drawn at random from a population. Let  $\hat{\beta}_0$  and  $\hat{\beta}_1$  be the least squares estimates. Suppose we have some test data  $(x'_1, y'_1), \dots, (x'_M, y'_M)$  drawn at random from the same population as the training data. Prove that

$$E\left[\frac{1}{N} \sum_{i=1}^N (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2\right] \leq E\left[\frac{1}{M} \sum_{i=1}^M (y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2\right]$$

## Problem Sheet 1 Solution

### Q1

Which of the following problem correspond to supervised learning? More than one correct answer can be chosen.

- A. Given a dataset of 60,000 small square  $28 \times 28$  pixel grayscale images of handwritten single digits between 0 and 9. Classify an image of a handwritten single digit into one of 10 classes representing integer values from 0 to 9.
- B. Find clusters of genes that interact with each other
- C. Find an investment portfolio that is likely to make a profit in the coming month
- D. Predict whether a website user will click on an ad

Solution: A, C, D

### Q2

Which of the following statement about parametric and non-parametric models is not true?  
Select all that apply:

- A. A parametric approach reduces the problem of estimating a functional form  $f$  down to the problem of estimating a set of parameters because it assumes a form for  $f$ .
- B. A non-parametric approach does not assume a functional form  $f$  and so does not require a large number of observations to estimate  $f$ .
- C. The advantages of a parametric approach to regression or classification are the simplifying of modelling  $f$  to a few parameters and therefore the estimations of the model parameters are more accurate compared to a non-parametric approach.
- D. The disadvantages of a parametric approach to regression or classification are a potential to inaccurately estimate  $f$  if the form of  $f$  assumed is wrong or to overfit the observations if more complex models are used.

Solution: B, C

**Q3**

Why is linear regression important to understand? Select all that apply:

- A. Linear regression is very extensible and can be used to capture nonlinear effects
- B. The linear model is often correct
- C. The linear model is hardly ever a good representation of the underlying system, but it is an important piece in many more complex methods
- D. Understanding simpler methods sheds light on more complex ones

Solution: A, C, D

**Q4**

Suppose that we build a simple linear regression  $Y = \beta_0 + \beta_1 X + \epsilon$  based on  $n$  observations  $(x_1, y_1), \dots, (x_n, y_n)$ . Which of the following indicates a fairly strong relationship between  $X$  and  $Y$ ? More than one correct answer can be chosen.

- A. The  $p$ -value for the null hypothesis  $\beta_1 = 0$  is 0.0001
- B. The estimated parameters  $\hat{\beta}_1 \gg 1$  and  $\hat{\beta}_0 \approx 0$
- C.  $R^2 = 0.9$
- D. The  $t$ -statistic for the null hypothesis  $\beta_1 = 0$  is 40

Solution: C

**Q5**

In the case of simple linear regression, show that the least squares line always passes through the point  $(\bar{x}, \bar{y})$ .

Solution: Simply using the formula of the linear square estimate of  $\beta_0$ ,  $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$ .

**Q6**

Consider a simple linear regression problem, proof that under the standard four assumptions (listed in the lecture notes), maximum likelihood estimation (MLE) is equivalent to the least squares estimation (LSE) of the model parameters.

Solution: Given the IID and normality distribution, the log-likelihood function can be written as follows:

$$\begin{aligned}
 l(\beta) &= \sum \log p(y_i | x_i, \beta) \\
 l(\hat{\beta}) &= \sum \log p(y_i | x_i, \hat{\beta}) \\
 &= \sum \log \left[ \left( \frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left( -\frac{1}{2\sigma^2} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 \right) \right] \\
 &\propto -\frac{1}{2\sigma^2} SSE(\hat{\beta})
 \end{aligned} \tag{13.2}$$

**Q7**

A company manufactures an electronic device to be used in a very wide temperature range. The company knows that increased temperature shortens the lifetime of the device, and a study is therefore performed in which the lifetime is determined as a function of temperature. The following data is found:

Temperature in Celcius	Lifetime in hours
10	420
20	365
30	285
40	220
50	176
60	117
70	69
80	34
90	5

Construct a linear regression model for this problem. Estimate the model parameters. Calculate the 95% confidence interval for the slope in the linear regression model.

Solution:

$$\hat{\beta}_0 = 453.556 \text{ and } \hat{\beta}_1 = -5.313$$

Confidence interval:  $(-5.915, -4.705)$

**Q8**

Consider a simple linear regression model, fit by least squares to a set of training data  $(x_1, y_1), \dots, (x_N, y_N)$  drawn at random from a population. Let  $\hat{\beta}_0$  and  $\hat{\beta}_1$  be the least squares estimates. Suppose we have some test data  $(x'_1, y'_1), \dots, (x'_M, y'_M)$  drawn at random from the same population as the training data. Prove that

$$E\left[\frac{1}{N} \sum_{i=1}^N (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2\right] \leq E\left[\frac{1}{M} \sum_{i=1}^M (y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2\right]$$

Solution:

Note

$$E\left[\frac{1}{M} \sum_{i=1}^M (y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2\right] = E[(y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2]$$

since for IID random variables  $z_i$ ,

$$E\left[\frac{1}{n} \sum_{i=1}^n z_i\right] = E(z)$$

Therefore, it doesn't matter whether we have  $M$  test points or  $N$  test points, i.e.

$$E\left[\frac{1}{M} \sum_{i=1}^M (y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2\right] = E\left[\frac{1}{N} \sum_{i=1}^N (y'_i - \tilde{\beta}_0 - \tilde{\beta}_1 x'_i)^2\right]$$

Define the random variables:

$A = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$  and  $B = \frac{1}{N} \sum_{i=1}^N (y'_i - \tilde{\beta}_0 - \tilde{\beta}_1 x'_i)^2$  where  $\tilde{\beta}_0$  and  $\tilde{\beta}_1$  are the LS estimates of the linear regression parameters based on the TEST set. Note that  $\{y_i, x_i\}$  and  $\{y'_i, x'_i\}$  are samples from the same distribution, hence  $A$  and  $B$  have the same distribution and  $E(A) = E(B)$ .

As  $\tilde{\beta}_0$  and  $\tilde{\beta}_1$  are the LS estimates,

$$B = \frac{1}{N} \sum_{i=1}^N (y'_i - \tilde{\beta}_0 - \tilde{\beta}_1 x'_i)^2 \leq \frac{1}{N} \sum_{i=1}^N (y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2$$

Therefore

$$E\left[\frac{1}{N} \sum_{i=1}^N (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2\right] = E\left[\frac{1}{N} \sum_{i=1}^N (y'_i - \tilde{\beta}_0 - \tilde{\beta}_1 x'_i)^2\right] \leq E\left[\frac{1}{N} \sum_{i=1}^N (y'_i - \hat{\beta}_0 - \hat{\beta}_1 x'_i)^2\right]$$

## Problem Sheet 2

### Q1

Make a sketch of typical squared bias, variance, irreducible error, training error and test error curves (i.e. five curves) on a single graph, where the x-axis represents the complexity of the model and the y-axis represents the values for each curve. Explain why each of the five curves has the shape sketched.

### Q2

Which of the following statement about modelling is NOT true? Select all that apply:

- A. A fitted model with more predictors will necessarily have a lower training error than a model with fewer predictors.
- B. The advantage of building a very complex model for regression or classification is obtaining a better fit to the data which will lead to smaller test error.
- C. The disadvantages for a very complex approach for regression or classification are i) requires estimating a greater number of parameters; ii) follows the noise too closely which might lead to overfitting.
- D. A more complex model would usually be preferred to a less complex approach when we are interested in prediction and not the interpretability of the results.

**Q3**

You are fitting a linear regression model to a data set. The model has up to 10 predictors and 100 observations. Which of the following is NOT true in terms of using the model selection criteria to select a number of predictors to include?

- A. Mallows'  $C_p$  will likely select a model with more predictors than AIC
- B. Mallows'  $C_p$  will select the same model as AIC
- C. Mallows'  $C_p$  will likely select a model with fewer predictors than BIC
- D. Mallows'  $C_p$  will likely select a model with more predictors than BIC

**Q4**

Following Q3, which of the model selection approach will lead to the smallest test MSE?

- A. Forward Stepwise Selection
- B. Backward Stepwise Selection
- C. Best Subset Selection
- D. Not enough information

**Q5**

Following Q3, if you decide to use Best Subset Selection to decide which predictors to include, how many different models will you end up considering? How many if you use Forward Stepwise Selection?

**Q6**

Which of the following statement about multicollinearity is TRUE? Select all that apply:

- A. When predictors are correlated, it indicates that changes in one predictor are associated with shifts in another predictor. The stronger the correlation, the more difficult it is to change one predictor without changing another.
- B. When predictors are highly correlated, the coefficient estimates become very sensitive to small changes in the model.
- C. Multicollinearity reduces the precision of the estimated coefficients, which weakens the statistical power of the regression model. One might not be able to trust the p-values to identify independent variables that are statistically significant.
- D. Multicollinearity does not influence the predictions, precision of the predictions. If the primary goal is to make predictions without understanding the role of each predictor, there is no need to reduce multicollinearity.

**Q7**

Consider a simple linear regression problem, prove that under the standard four assumptions (listed in the lecture notes):

- The least square estimates  $b_0$  and  $b_1$  are unbiased estimates.
- The variances of the least squares estimators in simple linear regression are:

$$Var[b_0] = \sigma_{b_0}^2 = \sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right)$$

$$Var[b_1] = \sigma_{b_1}^2 = \frac{\sigma^2}{S_{xx}}$$

- The covariance of the least squares estimators in simple linear regression is:

$$Cov[b_0, b_1] = \sigma_{b_0, b_1} = -\sigma^2 \frac{\bar{x}}{S_{xx}}$$

**Q8**

Suppose that you work part-time at a bowling alley that is open daily from noon to midnight. Although business is usually slow from noon to 6 P.M., the owner has noticed that it is better on hotter days during the summer, perhaps because the premises are comfortably air-conditioned. The owner shows you some data that she gathered last summer. This data set includes the maximum temperature (in Fahrenheit) and the number of lines bowled between noon and 6 P.M. for each of 20 days. (The maximum temperatures ranged from  $77^{\circ}F$  to  $95^{\circ}F$  during this period.) The owner would like to know if she can estimate tomorrow's business from noon to 6 P.M. by looking at tomorrow's weather forecast. She asks you to analyze the data. Let  $x$  be the maximum temperature for a day and  $y$  the number of lines bowled between noon and 6 P.M. on that day. The computer output based on the data for 20 days provided the following results:

$$\hat{y} = -432 + 7.7x, \quad s_e = 28.17, \quad S_{xx} = 607, \text{ and } \bar{x} = 87.5$$

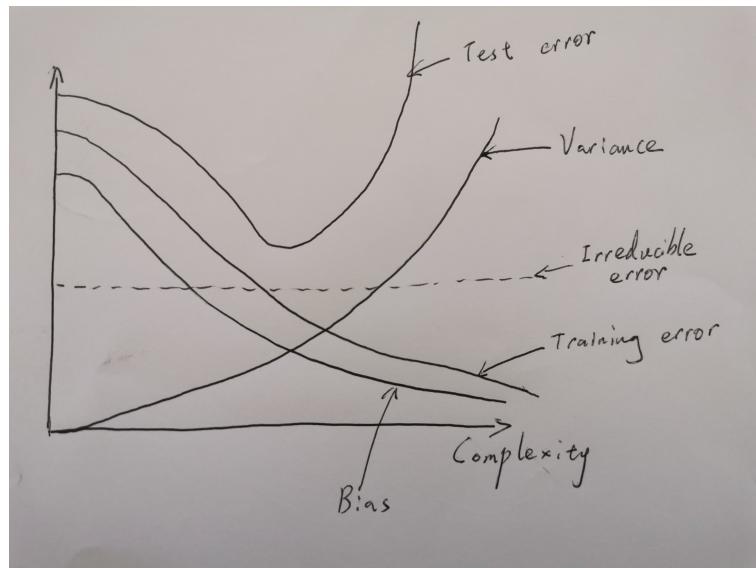
Assume that the weather forecasts are reasonably accurate.

- Does the maximum temperature seem to be a useful predictor of bowling activity between noon and 6 P.M.? Use an appropriate statistical procedure based on the information given. Use  $\alpha = 0.05$ .
- The owner wants to know how many lines of bowling she can expect, on average, for days with a maximum temperature of  $90^{\circ}F$ . Answer using a 95% confidence level.
- The owner has seen tomorrow's weather forecast, which predicts a high of  $90^{\circ}F$ . About how many lines of bowling can she expect? Answer using a 95% confidence level.
- The owner asks you how many lines of bowling she could expect if the high temperature were  $100^{\circ}F$ . Give a point estimate, together with an appropriate warning to the owner.

## Problem Sheet 2 Solution

### Q1

Make a sketch of typical squared bias, variance, irreducible error, training error and test error curves (i.e. five curves) on a single graph, where the x-axis represents the complexity of the model and the y-axis represents the values for each curve. Explain why each of the five curves has the shape sketched.



### Q2

Which of the following statement about modelling is NOT true? Select all that apply:

- A fitted model with more predictors will necessarily have a lower training error than a model with fewer predictors.
- The advantage of building a very complex model for regression or classification is obtaining a better fit to the data which will lead to smaller test error.
- The disadvantages for a very complex approach for regression or classification are i) requires estimating a greater number of parameters; ii) follows the noise too closely which might lead to overfitting.
- A more complex model would usually be preferred to a less complex approach when we are interested in prediction and not the interpretability of the results.

Solution: A, B For A, note the two models may be structurally different and do not share predictors at all. For B, not necessarily lead to small TEST error.

### Q3

You are fitting a linear regression model to a data set. The model has up to 10 predictors and 100 observations. Which of the following is NOT true in terms of using the model selection criteria to select a number of predictors to include?

- A. Mallows'  $C_p$  will likely select a model with more predictors than AIC
- B. Mallows'  $C_p$  will select the same model as AIC
- C. Mallows'  $C_p$  will likely select a model with fewer predictors than BIC
- D. Mallows'  $C_p$  will likely select a model with more predictors than BIC

Solution: A, C

#### **Q4**

Following Q3, which of the model selection approach will lead to the smallest test MSE?

- A. Forward Stepwise Selection
- B. Backward Stepwise Selection
- C. Best Subset Selection
- D. Not enough information

Solution: D

#### **Q5**

Following Q3, if you decide to use Best Subset Selection to decide which predictors to include, how many different models will you end up considering? How many if you use Forward Stepwise Selection?

Solution: i)  $2^{10} = 1024$  ii)  $1 + \frac{10 \times (10+1)}{2} = 56$

#### **Q6**

Which of the following statement about multicollinearity is TRUE? Select all that apply:

- A. When predictors are correlated, it indicates that changes in one predictor are associated with shifts in another predictor. The stronger the correlation, the more difficult it is to change one predictor without changing another.
- B. When predictors are highly correlated, the coefficient estimates become very sensitive to small changes in the model.
- C. Multicollinearity reduces the precision of the estimated coefficients, which weakens the statistical power of the regression model. One might not be able to trust the p-values to identify independent variables that are statistically significant.
- D. Multicollinearity does not influence the predictions, precision of the predictions. If the primary goal is to make predictions without understanding the role of each predictor, there is no need to reduce multicollinearity.

Solution: A, B, C, D

## Q7

Consider a simple linear regression problem, prove that under the standard four assumptions (listed in the lecture notes):

- The least square estimates  $b_0$  and  $b_1$  are unbiased estimates.
- The variances of the least squares estimators in simple linear regression are:

$$Var[b_0] = \sigma_{b_0}^2 = \sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right)$$

$$Var[b_1] = \sigma_{b_1}^2 = \frac{\sigma^2}{S_{xx}}$$

- The covariance of the least squares estimators in simple linear regression is:

$$Cov[b_0, b_1] = \sigma_{b_0, b_1} = -\sigma^2 \frac{\bar{x}}{S_{xx}}$$

Solution:

Note all the expectation and variance is conditioned on  $X$  is known.

- **Proof of  $E[b_1] = \beta_1$**  see lecture notes
- **Proof of  $E[b_0] = \beta_0$**

$$\begin{aligned} E(b_0) &= E(\bar{y} - b_1 \bar{x}) \\ &= \frac{1}{n} \sum E(y_i) - E(b_1 \bar{x}) \\ &= \frac{1}{n} \sum (\beta_0 + \beta_1 x_i) - E(b_1) \frac{1}{n} \sum x_i \\ &= \frac{1}{n} \sum (\beta_0 + \beta_1 x_i) - \beta_1 \frac{1}{n} \sum x_i \\ &= \beta_0 \end{aligned} \tag{13.3}$$

- **Proof of  $Var[b_0]$**  see lecture notes
- **Proof of  $Var[b_1]$**  see lecture notes
- **Proof of  $Cov[b_0, b_1]$**

$$\begin{aligned} Cov[b_0, b_1] &= Cov[\bar{y} - \bar{x}b_1, b_1] \\ &= Cov[\bar{y}, b_1] - \bar{x}Cov[b_1, b_1] \\ &= 0 - \bar{x}Var(b_1) \\ &= -\bar{x} \frac{\sigma^2}{S_{xx}} \end{aligned} \tag{13.4}$$

See the lecture notes of proof  $Var[b_1]$  for the proof of  $Cov[\bar{y}, b_1] = 0$ .

**Q8**

Suppose that you work part-time at a bowling alley that is open daily from noon to midnight. Although business is usually slow from noon to 6 P.M., the owner has noticed that it is better on hotter days during the summer, perhaps because the premises are comfortably air-conditioned. The owner shows you some data that she gathered last summer. This data set includes the maximum temperature (in Fahrenheit) and the number of lines bowled between noon and 6 P.M. for each of 20 days. (The maximum temperatures ranged from  $77^{\circ}\text{F}$  to  $95^{\circ}\text{F}$  during this period.) The owner would like to know if she can estimate tomorrow's business from noon to 6 P.M. by looking at tomorrow's weather forecast. She asks you to analyze the data. Let  $x$  be the maximum temperature for a day and  $y$  the number of lines bowled between noon and 6 P.M. on that day. The computer output based on the data for 20 days provided the following results:

$$\hat{y} = -432 + 7.7x, \quad s_e = 28.17, \quad S_{xx} = 607, \quad \bar{x} = 87.5$$

Assume that the weather forecasts are reasonably accurate.

- a. Does the maximum temperature seem to be a useful predictor of bowling activity between noon and 6 P.M.? Use an appropriate statistical procedure based on the information given. Use  $\alpha = 0.05$ .
- b. The owner wants to know how many lines of bowling she can expect, on average, for days with a maximum temperature of  $90^{\circ}\text{F}$ . Answer using a 95% confidence level.
- c. The owner has seen tomorrow's weather forecast, which predicts a high of  $90^{\circ}\text{F}$ . About how many lines of bowling can she expect? Answer using a 95% confidence level.
- d. The owner asks you how many lines of bowling she could expect if the high temperature were  $100^{\circ}\text{F}$ . Give a point estimate, together with an appropriate warning to the owner.

Solution:

- a.

$$\hat{y} = -432 + 7.7x, \quad s_e = 28.17, \quad S_{xx} = 607, \quad \bar{x} = 87.5$$

$$b_1 = 7.7, \quad s_{b_1} = s_e / \sqrt{S_{xx}} = 28.17 / \sqrt{607} = 1.1434$$

Test:  $H_0 : \beta_1 = 0$  vs  $H_1 : \beta_1 > 0$

Use  $t$  distribution with  $df = n - 2 = 20 - 2 = 18$

For  $\alpha = 0.05$ , the critical value  $t^* = 1.734$

$$t_{stats} = (7.7 - 0) / 1.1434 = 6.734 > t^*$$

Reject  $H_0$ , there is sufficient evidence to conclude  $\beta_1$  is positive, i.e. the maximum temperature and bowling activity between twelve noon and 6:00 pm have a positive association.

- b.

For  $x^* = 90$ ,  $\hat{y}^* = -432 + 7.7(90) = 261$

$$s_{ci} = s_e \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}} = 28.17 \sqrt{\frac{1}{20} + \frac{(90 - 87.5)^2}{607}} = 6.9172$$

The 95% confidence interval for  $u^*|x^* = 90$  is

$$\hat{y}^* \pm t_{\alpha/2}s_{ci} = 261 \pm 2.101(6.9172) - > (246.4670, 275.5330)$$

- c.

$$s_{pi} = s_e \sqrt{1 + \frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_x x}} = 28.17 \sqrt{1 + \frac{1}{20} + \frac{(90 - 87.5)^2}{607}} = 29.0068$$

The 95% prediction interval for  $y^*$  is

$$\hat{y}^* \pm t_{\alpha/2}s_{pi} = 261 \pm 2.101(29.0068) - > (200.0567, 321.9433)$$

- d.

$$y^* = -432 + 7.7(100) = 338 \text{ lines}$$

Our regression line is only valid for the range of x values in our sample ( $77^\circ F$  to  $95^\circ F$ ). We should interpret this estimate very cautiously and not attach too much value to it.

## Problem Sheet 3

### Q1

Suppose we estimate the regression coefficients in a linear regression model by minimizing

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

for a particular value of  $\lambda$ . Which of the following statement is TRUE? Select all that apply:

- A. As we increase  $\lambda$  from 0, the training RSS will steadily increase.
- B. As we increase  $\lambda$  from 0, the test RSS will steadily decrease.
- C. As we increase  $\lambda$  from 0, the (squared) bias will steadily decrease.
- D. As we increase  $\lambda$  from 0, the variance of fitted values will steadily decrease.

### Q2

Which of the following is a benefit of the sparsity imposed by the Lasso? Select all that apply:

- A. Sparse models are generally more easy to interpret.
- B. The Lasso does variable selection by default.
- C. Using the Lasso penalty helps to decrease the bias of the fits.
- D. Using the Lasso penalty helps to decrease the variance of the fits.

**Q3**

You perform a regression on a problem where your second predictor,  $x_2$ , is measured in meter. You decide to refit the model after changing  $x_2$  to be measured in kilometer. Which of the following is true? Select all that apply:

- A. If the model you performed is standard linear regression, then  $\hat{\beta}_2$  will change but  $\hat{y}$  will remain the same.
- B. If the model you performed is standard linear regression, then neither  $\hat{\beta}_2$  nor  $\hat{y}$  will change.
- C. If the model you performed is ridge regression, then  $\hat{\beta}_2$  and  $\hat{y}$  will both change.
- D. If the model you performed is Lasso regression, then neither  $\hat{\beta}_2$  nor  $\hat{y}$  will change.

**Q4**

You are working on a regression problem with many variables, so you decide to do Principal Components Analysis first and then fit the regression to the first 2 principal components. Which of the following would you expect to happen? Select all that apply:

- A. A subset of the features will be selected.
- B. Model Bias will decrease relative to the full least squares model.
- C. Variance of fitted values will decrease relative to the full least squares model.
- D. Model interpretability will improve relative to the full least squares model.

**Q5**

We compute the principal components of our  $p$  predictor variables. The RSS in a simple linear regression of  $Y$  onto the largest principal component will always be no larger than the RSS in a simple regression of  $Y$  onto the second largest principal component. True or False?

**Q6**

Suppose we estimate the regression coefficients for a lasso regression model of a single predictor  $x$  for by minimizing

$$\sum_{i=1}^n \left( y_i - \beta_0 - \beta_1 x_i \right)^2$$

subject to  $|\beta_1| \leq c$ . For a given  $c$ , the fitted coefficient for predictor  $x$  is  $\hat{\beta}_1 = \alpha$ . Suppose we now include an exact copy  $x^* = x$  and refit our lasso regression. Characterize the effect of this exact collinearity by describing the set of solutions for  $\hat{\beta}_1$  and  $\hat{\beta}_1^*$ , using the same value of  $c$ .

**Q7**

It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will now explore this property in a very simple setting. Suppose that  $n = 2$ ,  $p = 2$ ,  $x_{11} = x_{12}$ ,  $x_{21} = x_{22}$ . Furthermore, suppose that  $y_1 + y_2 = 0$  and  $x_{11} + x_{21} = 0$  and  $x_{12} + x_{22} = 0$ , so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero:  $\hat{\beta}_0 = 0$ .

- (a) Write out the ridge regression optimization problem in this setting.
- (b) Argue that in this setting, the ridge coefficient estimates satisfy  $\hat{\beta}_1 = \hat{\beta}_2$ .
- (c) Write out the lasso optimization problem in this setting.
- (d) Argue that in this setting, the lasso coefficients  $\hat{\beta}_1$  and  $\hat{\beta}_2$  are not unique—in other words, there are many possible solutions to the optimization problem in (c). Describe these solutions.

**Q8**

A principal component analysis of a 4-dimensional data set (containing the annual power generations of four power stations over a 20 years period) has been carried out. The standard deviation of each of the principal component (PC) is listed below:

	PC 1	PC 2	PC 3	PC 4
Standard deviation	2.056	0.492	0.279	0.154

The eigenvectors generated using principal component analysis are listed below:

	PC1	PC2	PC3	PC4
Station 1	0.361	-0.656	0.582	0.315
Station 2	-0.084	-0.730	-0.598	-0.319
Station 3	0.856	0.173	-0.076	-0.479
Station 4	0.358	0.075	-0.545	0.753

Let  $\hat{\Sigma} \in \mathbb{R}^{4 \times 4}$  denote the sample variance matrix of the data set.

- Outline the main goals of principal components analysis of data
- Provide the ordered eigenvalues of  $\hat{\Sigma}$ , and given the total variance of the data cloud.
- Draw a scree plot. How many components would you need in order to capture at least 95% of the total variance of the data cloud?
- Provide the first and the second eigenvector of  $\hat{\Sigma}$ , which we denote by  $\gamma_1$  and  $\gamma_2$ . Without carrying out calculations, give the numerical values of  $\gamma_1^T \gamma_1$ ,  $\gamma_2^T \gamma_1$  and  $\gamma_2^T \gamma_2$ , and explain your answer.

## Problem Sheet 3 Solution

**Q1**

Suppose we estimate the regression coefficients in a linear regression model by minimizing

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

for a particular value of  $\lambda$ . Which of the following statement is TRUE? Select all that apply:

- A. As we increase  $\lambda$  from 0, the training RSS will steadily increase.
- B. As we increase  $\lambda$  from 0, the test RSS will steadily decrease.
- C. As we increase  $\lambda$  from 0, the (squared) bias will steadily decrease.
- D. As we increase  $\lambda$  from 0, the variance of fitted values will steadily decrease.

Solution A, D

**Q2**

Which of the following is a benefit of the sparsity imposed by the Lasso? Select all that apply:

- A. Sparse models are generally more easy to interpret.
- B. The Lasso does variable selection by default.
- C. Using the Lasso penalty helps to decrease the bias of the fits.
- D. Using the Lasso penalty helps to decrease the variance of the fits.

Solution: A, B, D

**Q3**

You perform a regression on a problem where your second predictor,  $x\_2$ , is measured in meter. You decide to refit the model after changing  $x\_2$  to be measured in kilometer. Which of the following is true? Select all that apply:

- A. If the model you performed is standard linear regression, then  $\hat{\beta}_2$  will change but  $\hat{y}$  will remain the same.
- B. If the model you performed is standard linear regression, then neither  $\hat{\beta}_2$  nor  $\hat{y}$  will change.
- C. If the model you performed is ridge regression, then  $\hat{\beta}_2$  and  $\hat{y}$  will both change.
- D. If the model you performed is Lasso regression, then neither  $\hat{\beta}_2$  nor  $\hat{y}$  will change.

Solution: A, C

**Q4**

You are working on a regression problem with many variables, so you decide to do Principal Components Analysis first and then fit the regression to the first 2 principal components. Which of the following would you expect to happen? Select all that apply:

- A. A subset of the features will be selected.
- B. Model Bias will decrease relative to the full least squares model.
- C. Variance of fitted values will decrease relative to the full least squares model.
- D. Model interpretability will improve relative to the full least squares model.

Solution: C

**Q5**

We compute the principal components of our  $p$  predictor variables. The RSS in a simple linear regression of  $Y$  onto the largest principal component will always be no larger than the RSS in a simple regression of  $Y$  onto the second largest principal component. True or False?

Solution: False

**Q6**

Suppose we estimate the regression coefficients for a lasso regression model of a single predictor  $x$  for by minimizing

$$\sum_{i=1}^n \left( y_i - \beta_0 - \beta_1 x_i \right)^2$$

subject to  $|\beta_1| \leq c$ . For a given  $c$ , the fitted coefficient for predictor  $x$  is  $\hat{\beta}_1 = \alpha$ . Suppose we now include an exact copy  $x^* = x$  and refit our lasso regression. Characterize the effect of this exact collinearity by describing the set of solutions for  $\hat{\beta}_1$  and  $\hat{\beta}_1^*$ , using the same value of  $c$ .

Solution:

Apply the Lagrange multiplier, we minimize for the Lasso optimization is

$$\begin{aligned} & \sum_{i=1}^n \left( y_i - \beta_0 - \beta_1 x_i - \beta_1^* x_i^* \right)^2 + \lambda(|\beta_1| + |\beta_1^*|) \\ &= \sum_{i=1}^n \left( y_i - \beta_0 - (\beta_1 + \beta_1^*) x_i \right)^2 + \lambda|\beta_1 + \beta_1^*| + \lambda(|\beta_1| + |\beta_1^*| - |\beta_1 + \beta_1^*|) \end{aligned}$$

Note that the first two terms represent the same minimization problem for the original lasso regression (before we added a duplicate feature) and because

$$|\beta_1 + \beta_1^*| \leq |\beta_1| + |\beta_1^*|,$$

the third term is either positive or zero. It will be zero if  $\beta_1$  and  $\beta_1^*$  are either both positive or negative. We know that the minimum of the first two terms of this objective function happens for the solution  $\beta_1 + \beta_1^* = a$  as mentioned in the problem. Therefore the solution must satisfy  $\beta_1 + \beta_1^* = a$  and  $\beta_1 * \beta_1^* > 0$ .

## Q7

It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will now explore this property in a very simple setting. Suppose that  $n = 2$ ,  $p = 2$ ,  $x_{11} = x_{12}$ ,  $x_{21} = x_{22}$ . Furthermore, suppose that  $y_1 + y_2 = 0$  and  $x_{11} + x_{21} = 0$  and  $x_{12} + x_{22} = 0$ , so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero:  $\hat{\beta}_0 = 0$ .

- (a) Write out the ridge regression optimization problem in this setting.
- (b) Argue that in this setting, the ridge coefficient estimates satisfy  $\hat{\beta}_1 = \hat{\beta}_2$ .
- (c) Write out the lasso optimization problem in this setting.
- (d) Argue that in this setting, the lasso coefficients  $\hat{\beta}_1$  and  $\hat{\beta}_2$  are not unique—in other words, there are many possible solutions to the optimization problem in (c). Describe these solutions.

Solution:

- (a)  $(y_1 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_1 x_{21} - \beta_2 x_{22})^2 + \lambda(\beta_1^2 + \beta_2^2)$
- (b) Let  $x_{11} = x_{12} = x_1$ ,  $x_{21} = x_{22} = x_2$ . Take derivatives of the expression in (a) with respect to both  $\beta_1$  and  $\beta_2$  and setting them equal to zero gives

$$-2(y_1 x_1 + y_2 x_2) + 2(\hat{\beta}_1 + \hat{\beta}_2)x_1^2 + 2(\hat{\beta}_1 + \hat{\beta}_2)x_2^2 + \lambda\hat{\beta}_1 = 0$$

$$-2(y_1 x_1 + y_2 x_2) + 2(\hat{\beta}_1 + \hat{\beta}_2)x_1^2 + 2(\hat{\beta}_1 + \hat{\beta}_2)x_2^2 + \lambda\hat{\beta}_2 = 0$$

subtract one equation from the other givens  $\hat{\beta}_1 = \hat{\beta}_2$

- (c)  $(y_1 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_1 x_{21} - \beta_2 x_{22})^2 + \lambda(|\beta_1| + |\beta_2|)$
- (d) Note that minimizing the function in (c) is equivalent to minimizing the LS subject to the constraints  $|\beta_1| + |\beta_2| \leq c$ , which can be geometrically interpreted as a diamond centered at  $(0, 0)$ .

Also given that  $y_1 + y_2 = 0$  and  $x_{11} + x_{21} = 0$  and  $x_{12} + x_{22} = 0$ , the LS function  $(y_1 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_1 x_{21} - \beta_2 x_{22})^2 = 2(y_1 - (\beta_1 + \beta_2)x_1)^2$ .

Therefore the solution of minimizing LS is  $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_1}$ , which is in fact a line parallel to the edge of “Lasso-diamond”  $\hat{\beta}_1 + \hat{\beta}_2 = c$ . Note the solutions to the original Lasso optimization problem are contours of the function  $2(y_1 - (\hat{\beta}_1 + \hat{\beta}_2)x_1)^2$  that touch the Lasso-diamond  $\hat{\beta}_1 + \hat{\beta}_2 = c$ .

Since  $\hat{\beta}_1$  and  $\hat{\beta}_2$  are along the line  $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_1}$ , the entire edge  $\hat{\beta}_1 + \hat{\beta}_2 = c$  is a potential solution to the Lasso optimization problem! The same argument can be made for the opposite Lasso-diamond edge  $\hat{\beta}_1 + \hat{\beta}_2 = -c$ .

Therefore this Lasso problem does not have a unique solution. The general form of solution is given by two line segments:

$$\hat{\beta}_1 + \hat{\beta}_2 = c; \hat{\beta}_1 \geq 0 \text{ and } \hat{\beta}_2 \geq 0$$

$$\hat{\beta}_1 + \hat{\beta}_2 = -c; \hat{\beta}_1 \leq 0 \text{ and } \hat{\beta}_2 \leq 0$$

## Q8

A principal component analysis of a 4-dimensional data set (containing the annual power generations of four power stations over a 20 years period) has been carried out. The standard deviation of each of the principal component (PC) is listed below:

	PC 1	PC 2	PC 3	PC 4
Standard deviation	2.056	0.492	0.279	0.154

The eigenvectors generated using principal component analysis are listed below:

	PC1	PC2	PC3	PC4
Station 1	0.361	-0.656	0.582	0.315
Station 2	-0.084	-0.730	-0.598	-0.319
Station 3	0.856	0.173	-0.076	-0.479
Station 4	0.358	0.075	-0.545	0.753

Let  $\hat{\Sigma} \in \mathbb{R}^{4 \times 4}$  denote the sample variance matrix of the data set.

- Outline the main goals of principal components analysis of data
- Provide the ordered eigenvalues of  $\hat{\Sigma}$ , and given the total variance of the data cloud.
- Draw a scree plot. How many components would you need in order to capture at least 95% of the total variance of the data cloud?
- Provide the first and the second eigenvector of  $\hat{\Sigma}$ , which we denote by  $\gamma_1$  and  $\gamma_2$ . Without carrying out calculations, give the numerical values of  $\gamma_1^T \gamma_1$ ,  $\gamma_2^T \gamma_1$  and  $\gamma_2^T \gamma_2$ , and explain your answer.

Solution:

- To assess “true” dimensionality of multivariate data

To provide low-dimensional representations of data

To relate principal forms of variation to original variables

To provide principal components that are uncorrelated with each other.

- $\lambda_1 = 2.056^2 = 4.227136$

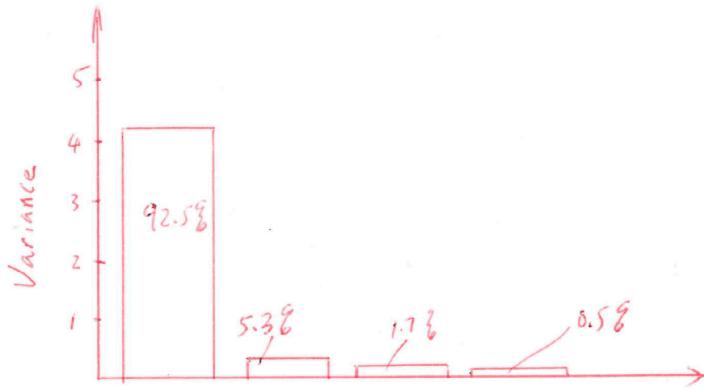
$$\lambda_2 = 0.492^2 = 0.242064$$

$$\lambda_3 = 0.279^2 = 0.077841$$

$$\lambda_4 = 0.154^2 = 0.023716$$

$$TV = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 4.570757$$

- We need the first two components to capture at least 95% of the total variance.



- $\gamma_1 = (0.361, -0.084, 0.856, 0.358)^T; \gamma_2 = (-0.656, -0.730, 0.173, 0.075)^T$ .

eigenvectors are unit vectors, i.e  $\gamma_1^T \gamma_1 = \gamma_2^T \gamma_2 = 1$  and mutually orthogonal,  
i.e.  $\gamma_2^T \gamma_1 = 0$ .

## Problem Sheet 4

### Q1

Which of the following statement about smoothing splines is TRUE? Select all that apply:

- A. Smoothing splines are natural cubic splines with an infinite number of knots.
- B. Letting tuning parameter  $\lambda$  tend to infinity leads to the standard (first-order) linear model situation.
- C. Too small values of  $\lambda$  could lead to overfitting.
- D.  $\lambda$  can be tuned via cross-validation.

### Q2

Let  $I\{x \leq c\}$  denote a function which is 1 if  $x \leq c$  and 0 otherwise.

Which of the following is a basis for linear splines with a knot at  $c$ ? Select all that apply:

- A. 1,  $x$ ,  $(x - c)I\{x > c\}$
- B. 1,  $x$ ,  $(x - c)I\{x \leq c\}$
- C. 1,  $x$ ,  $I\{x \leq c\}$
- D. 1,  $(x - c)I\{x > c\}$ ,  $(x - c)I\{x \leq c\}$ .

**Q3**

Suppose we have a random sample from  $n$  individuals. For each individual we have measurements on  $p$  predictor variables ( $X_1, \dots, X_p$ ) and a binary response variable  $Y$ . A logistic regression model is fitted to the data. Choose all TRUE statements about logistic regression.

- A. Logistic regression is based on a conditional model for  $(X_1, \dots, X_p)|Y = 0$  and  $(X_1, \dots, X_p)|Y = 1$
- B. If the coefficient of the  $j$ -th predictor variable,  $\beta_j$ , is positive, the probability that the response variable  $Y$  is equal to 1 increases as  $x_j$  increases whilst all other predictor variables remain the same.
- C. Logistic regression cannot use categorical variables as predictor variables.
- D. Logistic regression cannot be extended to handle categorical response variables which take more than two possible outcomes.

**Q4**

In terms of model complexity, which is more similar to a smoothing spline with 100 knots and 5 effective degrees of freedom?

- A. A natural cubic spline with 5 knots
- B. A natural cubic spline with 100 knots

**Q5**

In the GAM  $y \sim f_1(X_1) + f_2(X_2) + \epsilon$ , as we make  $f_1$  and  $f_2$  more and more complex we can approximate any regression function to arbitrary precision. True or False?

**Q6**

Is the following function a cubic spline? Why or why not?

$$f(x) = \begin{cases} 0, & x < 0, \\ x^3, & 0 \leq x < 1, \\ x^3 + (x-1)^3, & 1 \leq x < 2, \\ -(x-3)^3 - (x-4)^3, & 2 \leq x < 3, \\ -(x-4)^3, & 3 \leq x < 4, \\ 0, & 4 \leq x. \end{cases}$$

**Q7**

A cubic regression spline with one knot at  $\xi$  can be obtained using a basis of the form  $x, x^2, x^3, (x-\xi)_+^3$ , where  $(x-\xi)_+^3 = (x-\xi)^3$  if  $x > \xi$  and equals 0 otherwise. Show that a function of the form

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)_+^3$$

is indeed a cubic regression spline, regardless of the values of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

- a) Find a cubic polynomial

$$f_1(x) = a_1 + b_1x + c_1x^2 + d_1x^3$$

such that  $f(x) = f_1(x)$  for all  $x \leq \xi$ . Express  $a_1, b_1, c_1, d_1$  in terms of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

- b) Find a cubic polynomial

$$f_2(x) = a_2 + b_2x + c_2x^2 + d_2x^3$$

such that  $f(x) = f_2(x)$  for all  $x > \xi$ . Express  $a_2, b_2, c_2, d_2$  in terms of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

- c) Show that  $f_1(\xi) = f_2(\xi)$ .  
d) Show that  $f'_1(\xi) = f'_2(\xi)$ .  
e) Show that  $f''_1(\xi) = f''_2(\xi)$ .

## Q8

Prove that  $p(X) = \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}$  is equivalent to  $\ln \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X$ . In other words, the logistic function representation and logit transformation are equivalent.

## Q9

Consider a simple logistic regression model for a binary classification problem,  $p(X) = P(Y = 1|X) = \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}$ . Given the data of response  $\{y_i\}, i = 1, \dots, n$  and the corresponding data of single predictor  $\{x_i\}, i = 1, \dots, n$ , derive the log-likelihood of the parameters  $\beta_0, \beta_1$  and state any assumptions required for the derivation.

# Problem Sheet 4 Solution

## Q1

Which of the following statement about smoothing splines is TRUE? Select all that apply:

- A. Smoothing splines are natural cubic splines with an infinite number of knots.
- B. Letting tuning parameter  $\lambda$  tend to infinity leads to the standard (first-order) linear model situation.
- C. Too small values of  $\lambda$  could lead to overfitting.
- D.  $\lambda$  can be tuned via cross-validation.

Solution: B, C, D

**Q2**

Let  $I\{x \leq c\}$  denote a function which is 1 if  $x \leq c$  and 0 otherwise.

Which of the following is a basis for linear splines with a knot at  $c$ ? Select all that apply:

- A. 1,  $x$ ,  $(x - c)I\{x > c\}$
- B. 1,  $x$ ,  $(x - c)I\{x \leq c\}$
- C. 1,  $x$ ,  $I\{x \leq c\}$
- D. 1,  $(x - c)I\{x > c\}$ ,  $(x - c)I\{x \leq c\}$ .

Solution: A, B, D

**Q3**

Suppose we have a random sample from  $n$  individuals. For each individual we have measurements on  $p$  predictor variables  $(X_1, \dots, X_p)$  and a binary response variable  $Y$ . A logistic regression model is fitted to the data. Choose all TRUE statements about logistic regression.

- A. Logistic regression is based on a conditional model for  $(X_1, \dots, X_p)|Y = 0$  and  $(X_1, \dots, X_p)|Y = 1$
- B. If the coefficient of the  $j$ -th predictor variable,  $\beta_j$ , is positive, the probability that the response variable  $Y$  is equal to 1 increases as  $x_j$  increases whilst all other predictor variables remain the same.
- C. Logistic regression cannot use categorical variables as predictor variables.
- D. Logistic regression cannot be extended to handle categorical response variables which take more than two possible outcomes.

Solution: B

**Q4**

In terms of model complexity, which is more similar to a smoothing spline with 100 knots and 5 effective degrees of freedom?

- A. A natural cubic spline with 5 knots
- B. A natural cubic spline with 100 knots

Solution: A

Even though the smoothing spline has 100 knots, it is penalized to be smooth, so it is about as complex as a model with 5 variables (effective degree of freedom). The natural cubic spline with 5 knots has 5 degrees of freedom is such a model. Note a cubic spline with 5 knots has  $5+3+1$  degrees of freedom, the natural cubic spline adds two more linear constraints which frees 2 df at each side (a cubic polynomial model  $\rightarrow$  linear model)

**Q5**

In the GAM  $y \sim f_1(X_1) + f_2(X_2) + \epsilon$ , as we make  $f_1$  and  $f_2$  more and more complex we can approximate any regression function to arbitrary precision. True or False?

Solution: False

Note, this additive model can not capture interaction behaviors.

**Q6**

Is the following function a cubic spline? Why or why not?

$$f(x) = \begin{cases} 0, & x < 0, \\ x^3, & 0 \leq x < 1, \\ x^3 + (x-1)^3, & 1 \leq x < 2, \\ -(x-3)^3 - (x-4)^3, & 2 \leq x < 3, \\ -(x-4)^3, & 3 \leq x < 4, \\ 0, & 4 \leq x. \end{cases}$$

Solution: At  $x = 2$ ,  $f'$  is discontinuous (15 on one side and -15 on the other side), so this is not a cubic spline.

**Q7**

A cubic regression spline with one knot at  $\xi$  can be obtained using a basis of the form  $x, x^2, x^3, (x-\xi)_+^3$ , where  $(x-\xi)_+^3 = (x-\xi)^3$  if  $x > \xi$  and equals 0 otherwise. Show that a function of the form

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)_+^3$$

is indeed a cubic regression spline, regardless of the values of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

a) Find a cubic polynomial

$$f_1(x) = a_1 + b_1 x + c_1 x^2 + d_1 x^3$$

such that  $f(x) = f_1(x)$  for all  $x \leq \xi$ . Express  $a_1, b_1, c_1, d_1$  in terms of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

b) Find a cubic polynomial

$$f_2(x) = a_2 + b_2 x + c_2 x^2 + d_2 x^3$$

such that  $f(x) = f_2(x)$  for all  $x > \xi$ . Express  $a_2, b_2, c_2, d_2$  in terms of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

c) Show that  $f_1(\xi) = f_2(\xi)$ .

d) Show that  $f'_1(\xi) = f'_2(\xi)$ .

e) Show that  $f''_1(\xi) = f''_2(\xi)$ .

Solution:

- a) For  $x \leq \xi$ ,  $f_1(x)$  has coefficients  $a_1 = \beta_0$ ,  $b_1 = \beta_1$ ,  $c_1 = \beta_2$  and  $d_1 = \beta_3$   
b) For  $x > \xi$ , we have

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)^3$$

$$= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\xi^2 \beta_4) x + (\beta_2 - 3\beta_4 \xi) x^2 + (\beta_3 + \beta_4) x^3$$

so we take  $a_2 = \beta_0 - \beta_4 \xi^3$ ,  $b_2 = \beta_1 + 3\xi^2 \beta_4$ ,  $c_2 = \beta_2 - 3\beta_4 \xi$ ,  $d_2 = \beta_3 + \beta_4$ .

c)

$$f_1(\xi) = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3$$

and

$$f_2(\xi) = (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\xi^2 \beta_4) \xi + (\beta_2 - 3\beta_4 \xi) \xi^2 + (\beta_3 + \beta_4) \xi^3 = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3$$

d)

$$f'_1(\xi) = \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2$$

and

$$f'_2(\xi) = \beta_1 + 3\xi^2 \beta_4 + 2(\beta_2 - 3\beta_4 \xi) \xi + 3(\beta_3 + \beta_4) \xi^2 = \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2$$

e)

$$f''_1(\xi) = 2\beta_2 + 6\beta_3 \xi$$

and

$$f''_2(\xi) = 2(\beta_2 - 3\beta_4 \xi) + 6(\beta_3 + \beta_4) \xi = 2\beta_2 + 6\beta_3 \xi$$

## Q8

Prove that  $p(X) = \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}$  is equivalent to  $\ln \frac{p(X)}{1-p(X)} = \beta_0 + \beta_1 X$ . In other words, the logistic function representation and logit transformation are equivalent.

Solution:

$$\begin{aligned} \frac{p(X)}{1-p(X)} &= \frac{\frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}}{1 - \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}} \\ &= \frac{\frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}}{\frac{1 + \exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)} - \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}} \\ &= \frac{\frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}}{\frac{1}{1 + \exp(\beta_0 + \beta_1 X)}} \\ &= \exp(\beta_0 + \beta_1 X) \end{aligned} \tag{13.5}$$

therefore  $\ln \frac{p(X)}{1-p(X)} = \beta_0 + \beta_1 X$

**Q9**

Consider a simple logistic regression model for a binary classification problem,  $p(X) = P(Y = 1|X) = \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}$ . Given the data of response  $\{y_i\}, i = 1, \dots, n$  and the corresponding data of single predictor  $\{x_i\}, i = 1, \dots, n$ , derive the log-likelihood of the parameters  $\beta_0, \beta_1$  and state any assumptions required for the derivation.

Solution: The probability of the observed class was either  $p$ , if  $y_i = 1$ , or  $1 - p$ , if  $y_i = 0$ . The likelihood is then

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

The log-likelihood turns products into sums:

$$\begin{aligned} l(\beta_0, \beta_1) &= \sum_{i=1}^n y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i)) \\ &= \sum_{i=1}^n \log(1 - p(x_i)) + \sum_{i=1}^n y_i \log \frac{p(x_i)}{1 - p(x_i)} \\ &= \sum_{i=1}^n \log(1 - p(x_i)) + \sum_{i=1}^n y_i (\beta_0 + \beta_1 x_i) \\ &= \sum_{i=1}^n -\log(1 + \exp(\beta_0 + \beta_1 x_i)) + \sum_{i=1}^n y_i (\beta_0 + \beta_1 x_i) \end{aligned} \tag{13.6}$$

Note the assumptions required are

- independence of observations, in order to use the  $\prod$
- $p(X) = P(Y = 1|X) = \frac{\exp(\beta_0 + \beta_1 X)}{1 + \exp(\beta_0 + \beta_1 X)}$  suggests that we assume a linear relationship between the predictor  $X$  and the logit transformation of the response variable  $Y$ .

# **Part II**

# **Practical Classes**



# Practical Class Sheets 1

In this practical class, we will use R to fit linear regression models by the method of least squares (LS). We will make use of R's `lm` function simplify the process of model fitting and parameter estimation, and use information from its output to construct confidence intervals. Finally, we will assess the quality of the regression and use residual diagnostic plots to assess the validity of the regression assumptions.

*To start, create a new R script by clicking on the corresponding menu button (in the top left); and save it somewhere in your computer. You can now write all your code into this file, and then execute it in R using the menu button `Run` or by simple pressing `ctrl + enter`.*

## The `faithful` data set

The dataset we will be using in this practical class concerns the behaviour of the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. For a natural phenomenon, the Old Faithful geyser is highly predictable and has erupted every 44 to 125 minutes since 2000. There are two variables in the data set:

1. the waiting time between eruptions of the geyser, and
2. the duration of the eruptions.

Before we start fitting any models, the first step should always be to perform some exploratory data analysis of the data to gain some insight into the behaviour of the variables and suggest any interesting relationships or hypotheses to explore further.

### Exercise 1.1

- Load the `faithful` dataset by typing `data(faithful)`
- To save typing later, let us extract the columns into two variables. Create a vector `w` that contains the vector of waiting times, and a second vector `d` that contains the durations of eruptions.
- Plot the waiting times (y-axis) against the durations (x-axis).
- Compute the Pearson correlation coefficient between `w` and `d` using the `cor` function.
- Is there evidence of a linear relationship?

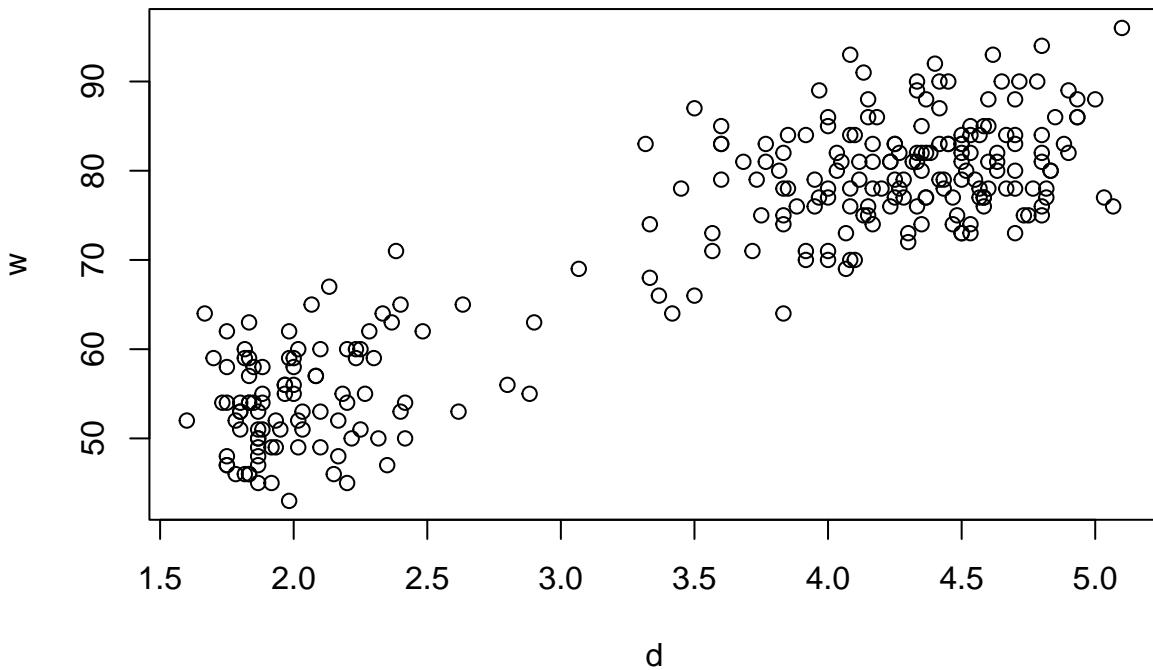
Click for solution

```
## SOLUTION  
data(faithful)
```

```
w <- faithful$waiting
d <- faithful$erruptions
cor(w, d)
```

```
## [1] 0.9008112
```

```
plot(y=w, x=d)
```



## Fitting the simple linear regression model

The model we're interested in fitting to these data is a simple linear relationship between the waiting times,  $w$ , and the eruption durations,  $d$ :

$$w = \beta_0 + \beta_1 d + \varepsilon.$$

R provides the `lm` function to compute the least-squares linear regression for us, and it returns an object which summarises all aspects of the regression fit.

### Technique

In R, we can fit linear models by the method of least squares using the function `lm`. Suppose our response variable is  $Y$ , and we have a predictor variable  $X$ , with observations stored in R vectors  $y$  and  $x$  respectively. To fit  $y$  as a linear function of  $x$ , we use the following `lm` command:

```
model <- lm(y ~ x)
```

Alternatively, if we have a data frame called `dataset` with variables in columns `a` and `b` then we could fit the linear regression of `a` on `b` without having to extract the columns into vectors by specifying the `data` argument

```
model <- lm(a ~ b, data=dataset)
```

The  $\sim$  (tilde) symbol in this expression should be read as “is modelled as”. Note that *R* always automatically include a constant term, so we only need to specify the  $X$  and  $Y$  variables.

We can inspect the fitted regression object returned from `lm` to get a simple summary of the estimated model parameters. To do this, simply evaluate the variable which contains the information returned from `lm`: `model`

### Exercise 1.2

- Use `lm` to fit waiting times `w` as a linear function of eruption durations `d`.
- Save the result of the regression function to `model`.
- Inspect the value of `model` to find the fitted least-squares coefficients.

Click for solution

```
## SOLUTION
# lm(y ~ x) use only x variable
# lm(y ~ .) use all predictors
# lm(y ~ 1) intercept only

model <- lm(w~d)
model
```

```
##
## Call:
## lm(formula = w ~ d)
##
## Coefficients:
## (Intercept)           d
##            33.47       10.73
```

### Technique

*R* also has a number of functions that, when applied to the results of a linear regression, will return key quantities such as residuals and fitted values.

- `coef(model)` and `coefficients(model)` – returns the estimated model coefficients as a vector  $(b_0, b_1)$
- `fitted(model)` and `fitted.values(model)` – returns the vector of fitted values,  $\hat{y}_i = b_0 + b_1 x_i$
- `resid(model)` and `residuals(model)` – returns the vector of residual values,  $e_i = y_i - \hat{y}_i$

### Exercise 1.3

- a. Use the `coef` function to extract the coefficients of the fitted linear regression model as a vector.

- b. Extract the vector of residuals from `model`, and use this to compute the sum of squares of the residuals as `lsq.Q`.

Click for solution

```
## SOLUTION
# (a)
coef(model)

## (Intercept)      d
##     33.47440   10.72964

beta0hat <- coef(model)[1]
beta0hat

## (Intercept)
##     33.4744

beta1hat <- coef(model)[2]
beta1hat

##      d
## 10.72964

# (b)
lsq.resid <- resid(model)
lsq.Q <- sum(lsq.resid^2)
lsq.Q

## [1] 9443.387
```

## The regression summary

### Technique

We can easily extract and inspect the coefficients and residuals of the linear model, but to obtain a more complete statistical summary of the model we use the `summary` function.:

```
summary(model)
```

There is a lot of information in this output, but the key quantities to focus on are:

- **Residuals:** simple summary statistics of the residuals from the regression.
- **Coefficients:** a table of information about the fitted coefficients. Its columns are:
  - *The label of the fitted coefficient:* The first will usually be `(Intercept)`, and subsequent rows will be named after the other predictor variables in the model.
  - The `Estimate` column gives the least squares estimates of the coefficients.
  - The `Std. Error` column gives the corresponding standard error for each coefficient.
  - The `t value` column contains the  $t$  test statistics.

- The  $\text{Pr}(>|t|)$  column is then the  $p$ -value associated with each test, where a low  $p$ -value indicates that we have evidence to reject the null hypothesis.
- **Residual standard error:** This gives  $s_e$  the square root of the unbiased estimate of the residual variance.
- **Multiple R-Squared:** This is the  $R^2$  value defined in lectures as the squared correlation coefficient and is a measure of goodness of fit.

We can also use the `summary` to access the individual elements of the regression summary output. If we save the results of a call to the `summary` function of a `lm` object as `summ`:

- `summ$residuals` – extracts the regression residuals as `resid` above
- `summ$coefficients` – returns the  $p \times 4$  coefficient summary table with columns for the estimated coefficient, its standard error, t-statistic and corresponding (two-sided) p-value.
- `summ$sigma` – extracts the regression standard error
- `summ$r.squared` – returns the regression  $R^2$

#### Exercise 1.4

- Apply the `summary` function to `model` to produce the regression summary for our example.
- Make sure you are able to locate:
  - The coefficient estimates, and their standard errors,
  - The residual standard error,  $s_e$ ;
  - The regression  $R^2$ .
- Extract the regression standard error from the regression summary, and save it as `se`. (We'll need this later!)
- What do the entries in the `t value` column correspond to? What significance test (and what hypotheses) do these statistics correspond to?
- Are the coefficients significant or not? What does this suggest about the regression?
- What is the  $R^2$  value for the regression and what is its interpretation? Can you extract its numerical value from the output as a new variable?
- Does the  $R^2$  value ‘agree’ with the Pearson correlation coefficient between `w` and `d` you calculated early.

Click for solution

```
## SOLUTION
# (a) & (b)
summary(model)
```

```
##
## Call:
## lm(formula = w ~ d)
##
## Residuals:
```

```

##      Min       1Q   Median      3Q      Max
## -12.0796 -4.4831  0.2122  3.9246 15.9719
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept) 33.4744    1.1549  28.98 <0.0000000000000002 ***
## d           10.7296    0.3148  34.09 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.914 on 270 degrees of freedom
## Multiple R-squared:  0.8115, Adjusted R-squared:  0.8108
## F-statistic: 1162 on 1 and 270 DF,  p-value: < 0.0000000000000022

# (c)
se <- summary(model)$sigma
se

## [1] 5.914009

# (d) & (e)
# The column t value shows you the t-test associated with testing the significance
# of the parameter listed in the first column.
# Both coefficients are significant, as p-values are very small.

# (f)
rsq <- summary(model)$r.squared
rsq

## [1] 0.8114608

# (g)
cor(w, d)^2

## [1] 0.8114608

```

## Inference on the coefficients

Given a fitted regression model, we can perform various hypothesis tests and construct confidence intervals to perform the usual kinds of statistical inference. To do this, we require the Normality assumption to hold for our inference to be valid.

Theory tells us that the standard errors for the parameter estimates are defined as

$$s_{b_0} = s_e \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}, \quad s_{b_1} = \frac{s_e}{\sqrt{S_{xx}}},$$

where  $s_{b_0}$  and  $s_{b_1}$  are unbiased estimates for  $\sigma_{b_0}$  and  $\sigma_{b_1}$ , the variances of  $b_0$  and  $b_1$ .

Given these standard errors and the assumption of normality, we can say for coefficient  $\beta_i$ ,  $i = 0, 1$  that:

$$\frac{b_i - \beta_i}{s_{b_i}} \sim t_{n-2},$$

and hence an  $\alpha$  level confidence interval for  $\beta_i$  can be written as

$$b_i \pm t_{n-2, \alpha/2}^* s_{b_i}.$$

### Exercise 1.5

- Consider the regression slope. Use the regression summary to extract the standard error of  $b_1$ . Assign its value to the variable `se.beta1`.
- Compute the test statistic defined above using `beta1hat` and `se.beta1` for a test under the null hypothesis  $H_0 : \beta_1 = 0$ . Does it agree with the summary output from R?
- Use the `pt` function to find the probability of observing a test statistic as extreme as this.
- Referring to your  $t$ -test statistic and the `t` values in the regression summary, what do you conclude about the significance of the regression coefficients?
- Find a 95% confidence interval for the slope  $\beta_1$ . You'll need to use the `qt` function to find the critical value of the  $t$ -distribution.
- An easy way to find the confidence interval in (e) is by using the function `confint`.

Click for solution

```
## SOLUTION
# (a)
# extract the standard error from the second column of the coefficient summary table
se.beta1 <- summary(model)$coefficients[2,2]
se.beta1

## [1] 0.3147534

# (b)
## compute the t statistic
t.beta1 <- (beta1hat-0)/se.beta1
t.beta1

##          d
## 34.08904

t.beta1<-unname(t.beta1)
t.beta1

## [1] 34.08904

# (c)
## p-value
n <- length(w)
2*(1-pt(t.beta1, n-2)) # df =n-2

## [1] 0
```

```

# (e)
## confidence interval
beta1hat + c(-1,1) * qt(0.975,n-2) * se.beta1

## [1] 10.10996 11.34932

# (f)
confint(model, level = 0.95)

##               2.5 %   97.5 %
## (Intercept) 31.20069 35.74810
## d          10.10996 11.34932

# for the slope parameter
confint(model, parm = "d", level = 0.95)

##               2.5 %   97.5 %
## d 10.10996 11.34932

# or
confint(model, level = 0.95)[2,]

##               2.5 %   97.5 %
## d 10.10996 11.34932

```

## Estimation and prediction

Suppose now that we are interested in some new point  $X = x^*$ , and at this point we want to predict: (a) the location of the regression line, and (b) the value of  $Y$ .

In the first problem, we are interested in the value  $\mu^* = \beta_0 + \beta_1 x^*$ , i.e. the location of the regression line, at some new point  $X = x^*$ .

In the second problem, we are interested in the value of  $Y^* = \beta_0 + \beta_1 x^* + \varepsilon$ , i.e. the actual observation value, at some new  $X = x^*$ .

The difference between the two is that the first simply concerns the location of the regression line, whereas the inference for the new observation  $Y^*$  concerns both the position of the regression line and the regression error about that point.

### Exercise 1.6

- Find a 95% confidence interval for the estimated waiting time  $\widehat{W}$  when the eruption duration is 3 minutes using the `predict` function.
- Find a 95% prediction interval for the actual waiting time,  $W$ .
- Compare the two intervals.

[Click for solution](#)

```

## SOLUTION
newdata1<- data.frame(d = 3)
predict(model, newdata = newdata1, interval = "confidence", level = 0.95)

##          fit      lwr      upr
## 1 65.66332 64.89535 66.4313

predict(model, newdata = newdata1, interval = "prediction", level = 0.95)

##          fit      lwr      upr
## 1 65.66332 53.99458 77.33206

```

## Residual Analysis

The residuals from our linear regression are the values  $e_i = \hat{\varepsilon}_i = y_i - \hat{y}_i$  for  $i = 1, \dots, n$ , where  $\hat{y}_i = b_0 + b_1 x_i$ . Analysis of the residuals allow us to perform diagnostic tests to investigate the goodness of fit of our regression under our modelling assumptions.

Under the assumption of linearity, given the LS estimates, the observed values of  $X$  are uncorrelated with the residuals. Which, in the case of simple linear regression, implies that the residuals are uncorrelated with any linear combination of  $X$ , in particular the fitted values  $\hat{y}_i = b_0 + b_1 x_i$ . Therefore, our diagnostics are based on the scatterplots of  $e$  against  $\hat{y}$ . In the case of simply linear regression will look the same as plotting  $e$  against  $x$ .

To assess the SLR assumptions, we inspect the residual plot for particular features:

- **Even and random scatter of the residuals about 0** is the expected behaviour when our SLR assumptions are satisfied.
- **Residuals shown evidence of a trend or pattern** — The presence of a clear pattern or trend in the residuals suggests that  $\mathbb{E}[\epsilon]_i \neq 0$  and  $\text{Cov}[\epsilon_i, \epsilon_j] \neq 0$ . There is clearly structure in the data that is not explained by the regression model, and so a simple linear model is not adequate for explaining the behaviour of  $Y$ .
- **Spread of the residuals is not constant** — if the spread of the residuals changes substantially as  $x$  (or  $\hat{y}$ ) changes then clearly our assumption of constant variance is not upheld.
- **A small number of residuals are very far from the others and 0** — observations with very large residuals are known as *outliers*. Sometimes these points can be explained through points with particularly high measurement error, or the effect of another variable which should be included in the model. Their presence could signal problems with the data, a linear regression being inadequate, or a violation of Normality.

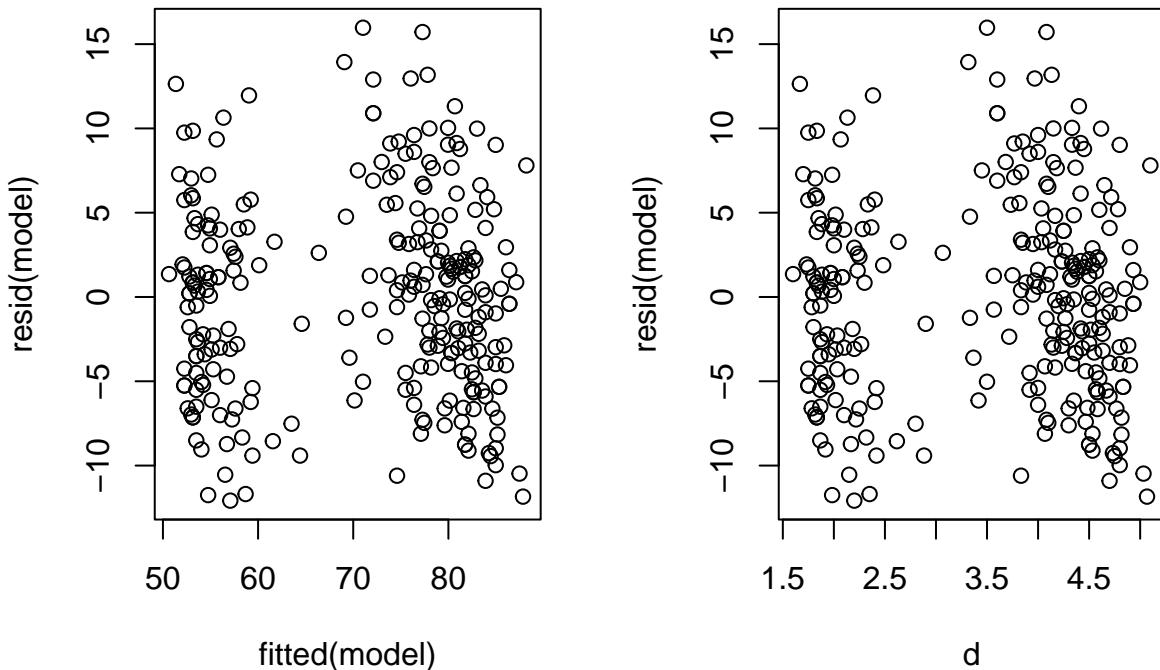
### Exercise 1.7

- Plot the residuals against the fitted values, and the residuals against the eruption durations side-by-side.
- What do you see? Why should these plots be similar?

- Use your residual plot to assess whether the SLR assumptions are valid for these data.

Click for solution

```
par(mfrow=c(1, 2))
plot(y=resid(model), x=fitted(model))
plot(y=resid(model), x=d)
```



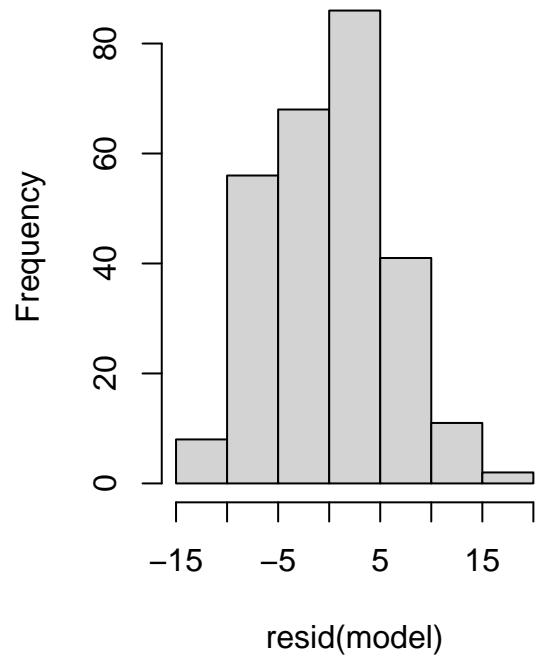
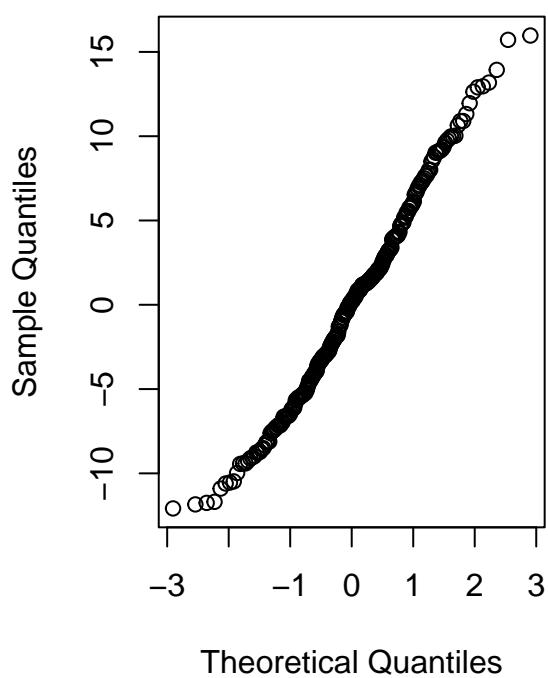
In order to state confidence intervals for the coefficients and for predictions made using this model, in addition to the assumptions tested above, we also require that the regression errors are normally distributed. We can check the normality assumption using quantile plots.

### Exercise 1.8

- Produce a side-by-side plot of the histogram of the residuals and a normal quantile plot of the residuals.
- Do the residuals appear to be normally distributed?

Click for solution

```
par(mfrow=c(1, 2))
hist(resid(model))
qqnorm(resid(model))
```

**Histogram of resid(model)****Normal Q-Q Plot**



# Practical Class Sheets 2

In this practical class, we will use R to fit linear regression models, make inference of model parameters, construct confidence intervals, assess the quality of the model, use residual diagnostic plots to assess the validity of the regression assumptions and conduct model selection for multiple linear regression.

*To start, create a new R script by clicking on the corresponding menu button (in the top left); and save it somewhere in your computer. You can now write all your code into this file, and then execute it in R using the menu button **Run** or by simple pressing **ctrl + enter**.*

## Body Fat and Body Measurements Data

In this practical class we will investigate the dataset **fat** from the library **faraway**, which consists of 252 observations concerning percentage of body fat and body measurements. We are interested in predicting body fat (variable **brozek**) using the other variables available, except for **siri** (another way of computing body fat), **density** (it is used in the **brozek** and **siri** formula) and **free** (it is computed using the **brozek** formula).

First, we need to install the **faraway** package, using the command `install.packages("faraway")`. We only need to do this once.

Now, load the package and the data set and read the help file for more information about the variables:

```
library("faraway")
```

## Exploratory data analysis

To get more information about the data set:

```
names(fat)

## [1] "brozek"   "siri"      "density"    "age"       "weight"     "height"    "adipos"
## [8] "free"      "neck"      "chest"      "abdom"     "hip"        "thigh"     "knee"
## [15] "ankle"     "biceps"    "forearm"   "wrist"

?fat
str(fat)
```

```

## 'data.frame': 252 obs. of 18 variables:
## $ brozek : num 12.6 6.9 24.6 10.9 27.8 ...
## $ siri   : num 12.3 6.1 25.3 10.4 28.7 ...
## $ density: num 1.07 1.09 1.04 1.08 1.03 ...
## $ age    : int 23 22 22 26 24 24 26 25 25 ...
## $ weight : num 154 173 154 185 184 ...
## $ height : num 67.8 72.2 66.2 72.2 71.2 ...
## $ adipos : num 23.7 23.4 24.7 24.9 25.6 ...
## $ free   : num 135 161 116 165 133 ...
## $ neck   : num 36.2 38.5 34 37.4 34.4 39 36.4 ...
## $ chest  : num 93.1 93.6 95.8 101.8 97.3 ...
## $ abdom  : num 85.2 83 87.9 86.4 100 94.4 90.7 ...
## $ hip    : num 94.5 98.7 99.2 101.2 101.9 ...
## $ thigh  : num 59 58.7 59.6 60.1 63.2 66 58.4 ...
## $ knee   : num 37.3 37.3 38.9 37.3 42.2 42 38.3 ...
## $ ankle  : num 21.9 23.4 24 22.8 24 25.6 22.9 ...
## $ biceps : num 32 30.5 28.8 32.4 32.2 35.7 31.9 ...
## $ forearm: num 27.4 28.9 25.2 29.4 27.7 30.6 27.8 ...
## $ wrist  : num 17.1 18.2 16.6 18.2 17.7 18.8 17.7 ...
# Look at the first few rows
head(fat)

```

```

##   brozek siri density age weight height adipos  free neck chest abdom  hip
## 1    12.6 12.3  1.0708 23  154.25  67.75  23.7 134.9 36.2 93.1 85.2 94.5
## 2     6.9  6.1  1.0853 22  173.25  72.25  23.4 161.3 38.5 93.6 83.0 98.7
## 3    24.6 25.3  1.0414 22  154.00  66.25  24.7 116.0 34.0 95.8 87.9 99.2
## 4    10.9 10.4  1.0751 26  184.75  72.25  24.9 164.7 37.4 101.8 86.4 101.2
## 5    27.8 28.7  1.0340 24  184.25  71.25  25.6 133.1 34.4 97.3 100.0 101.9
## 6    20.6 20.9  1.0502 24  210.25  74.75  26.5 167.0 39.0 104.5 94.4 107.8
##   thigh knee ankle biceps forearm wrist
## 1  59.0 37.3 21.9  32.0   27.4  17.1
## 2  58.7 37.3 23.4  30.5   28.9  18.2
## 3  59.6 38.9 24.0  28.8   25.2  16.6
## 4  60.1 37.3 22.8  32.4   29.4  18.2
## 5  63.2 42.2 24.0  32.2   27.7  17.7
## 6  66.0 42.0 25.6  35.7   30.6  18.8

```

```

# Summary statistics
summary(fat)

```

```

##      brozek        siri       density        age
## Min.   : 0.00   Min.   : 0.00   Min.   :0.995   Min.   :22.00
## 1st Qu.:12.80  1st Qu.:12.47  1st Qu.:1.041   1st Qu.:35.75
## Median :19.00  Median :19.20  Median :1.055   Median :43.00
## Mean   :18.94  Mean   :19.15  Mean   :1.056   Mean   :44.88
## 3rd Qu.:24.60  3rd Qu.:25.30  3rd Qu.:1.070   3rd Qu.:54.00
## Max.   :45.10  Max.   :47.50  Max.   :1.109   Max.   :81.00

```

```

##      weight       height      adipos       free
##  Min.   :118.5   Min.   :29.50   Min.   :18.10   Min.   :105.9
##  1st Qu.:159.0   1st Qu.:68.25   1st Qu.:23.10   1st Qu.:131.3
##  Median :176.5   Median :70.00   Median :25.05   Median :141.6
##  Mean   :178.9   Mean   :70.15   Mean   :25.44   Mean   :143.7
##  3rd Qu.:197.0   3rd Qu.:72.25   3rd Qu.:27.32   3rd Qu.:153.9
##  Max.   :363.1   Max.   :77.75   Max.   :48.90   Max.   :240.5
##      neck        chest       abdom       hip
##  Min.   :31.10   Min.   : 79.30   Min.   : 69.40   Min.   : 85.0
##  1st Qu.:36.40   1st Qu.: 94.35   1st Qu.: 84.58   1st Qu.: 95.5
##  Median :38.00   Median : 99.65   Median : 90.95   Median : 99.3
##  Mean   :37.99   Mean   :100.82   Mean   : 92.56   Mean   : 99.9
##  3rd Qu.:39.42   3rd Qu.:105.38   3rd Qu.: 99.33   3rd Qu.:103.5
##  Max.   :51.20   Max.   :136.20   Max.   :148.10   Max.   :147.7
##      thigh        knee       ankle      biceps      forearm
##  Min.   :47.20   Min.   :33.00   Min.   :19.1    Min.   :24.80   Min.   :21.00
##  1st Qu.:56.00   1st Qu.:36.98   1st Qu.:22.0    1st Qu.:30.20   1st Qu.:27.30
##  Median :59.00   Median :38.50   Median :22.8    Median :32.05   Median :28.70
##  Mean   :59.41   Mean   :38.59   Mean   :23.1    Mean   :32.27   Mean   :28.66
##  3rd Qu.:62.35   3rd Qu.:39.92   3rd Qu.:24.0    3rd Qu.:34.33   3rd Qu.:30.00
##  Max.   :87.30   Max.   :49.10   Max.   :33.9    Max.   :45.00   Max.   :34.90
##      wrist
##  Min.   :15.80
##  1st Qu.:17.60
##  Median :18.30
##  Mean   :18.23
##  3rd Qu.:18.80
##  Max.   :21.40

# Check for missing values
sum(is.na(fat))

## [1] 0
# zero here, so no missing values

```

Create a new dataframe called `fat1` which is equivalent to `fat`, but with the variables `siri`, `density` and `free` removed.

```

# Remove variables 2, 3 and 8.
fat1 <- fat[,-c(2,3,8)]
# siri is variable 2, density variable 3 and free variable 8.

```

## Exercise 2.1

- How many rows are in this data set? How many columns? What do the rows and columns represent?
- Make some pairwise scatterplots of the predictors (columns) in this data set. Describe

your findings.

- c. Are any of the predictors associated with percentage of body fat? If so, explain the relationship.

Click for solution

```
## SOLUTION
```

```
# (a)
```

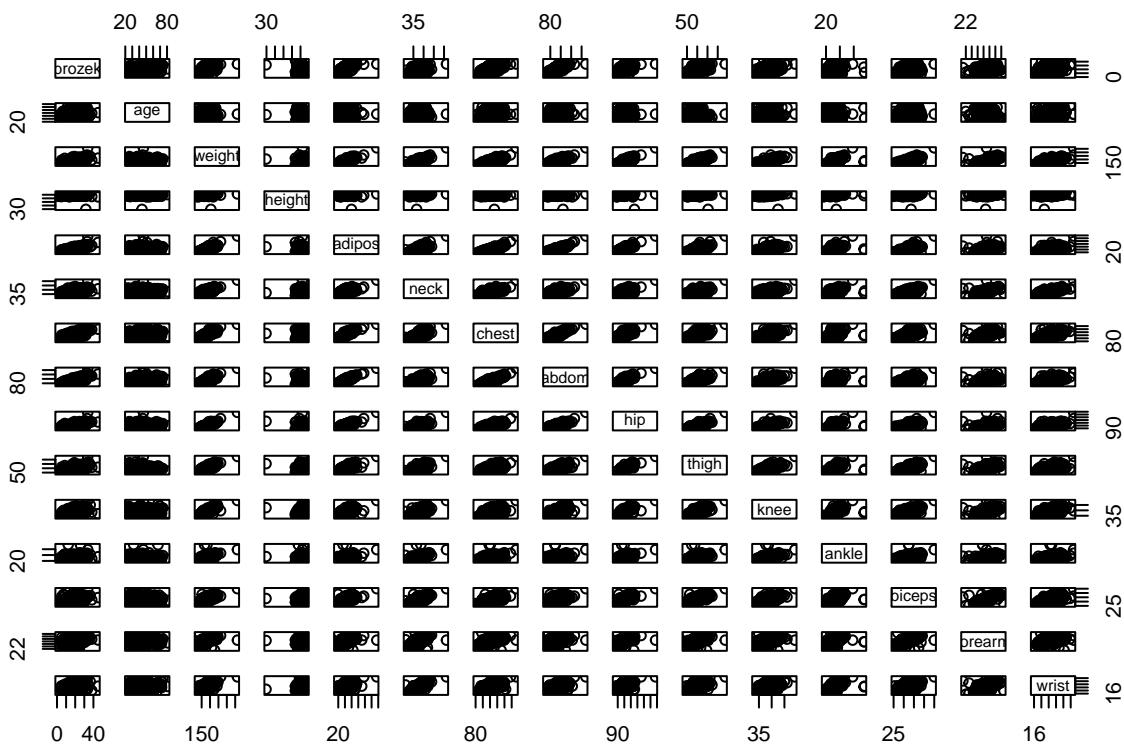
```
dim(fat1)
```

```
## [1] 252 15
```

```
# 252 observations and 15 variables
```

```
# (b)
```

```
pairs(fat1)
```



```
# (c)
```

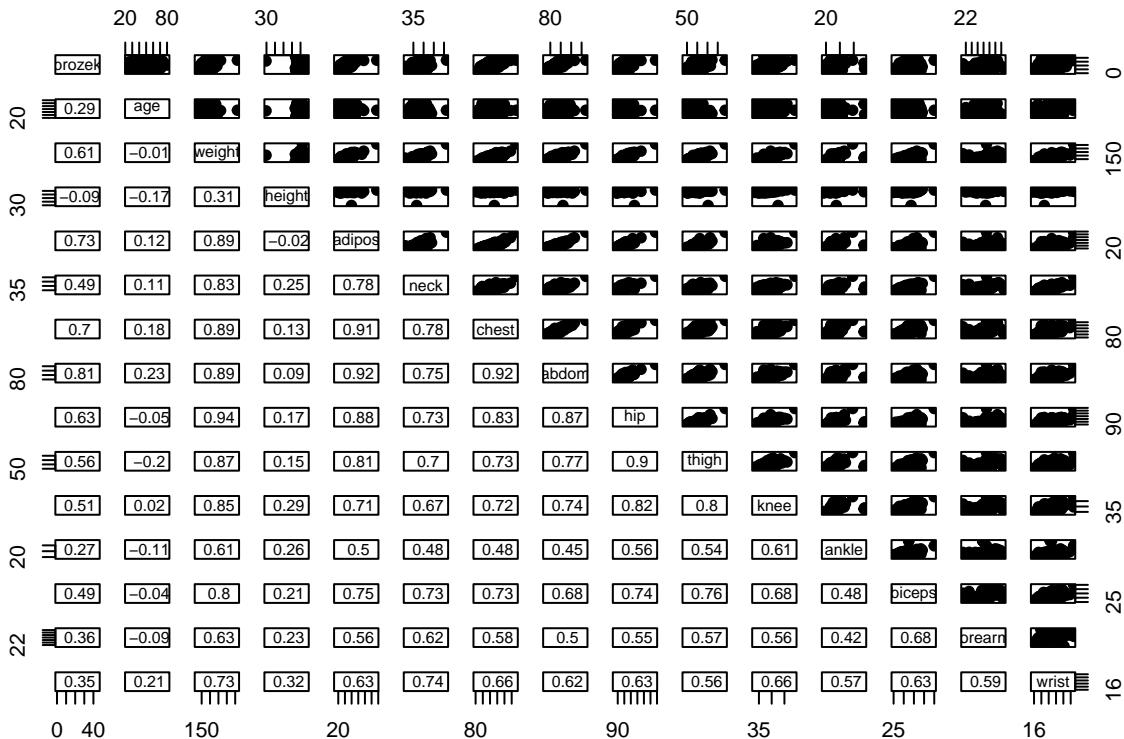
```
# Add correlation coefficients
```

```
# Correlation panel
```

```
panel.cor <- function(x, y){
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- round(cor(x, y), digits=2)
  txt <- paste0(" ", r)
  text(0.5, 0.5, txt, cex = 0.8)
}
```

```
# Customize upper panel
upper.panel<-function(x, y){
  points(x,y, pch = 19)
}

# Create the plots
pairs(fat1,
      lower.panel = panel.cor,
      upper.panel = upper.panel)
```



# A nicer plot using the corrplot package.

```
#install.packages("corrplot")
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
##
## Attaching package: 'corrplot'
```

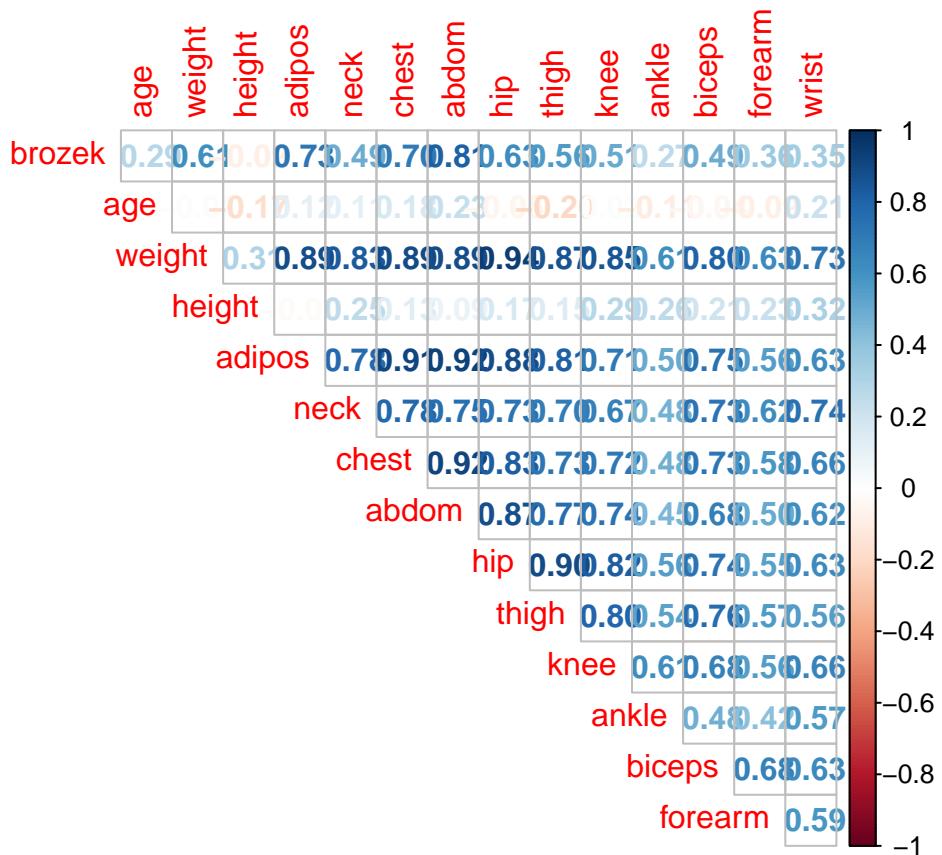
```
## The following object is masked from 'package:pls':
```

```
##
```

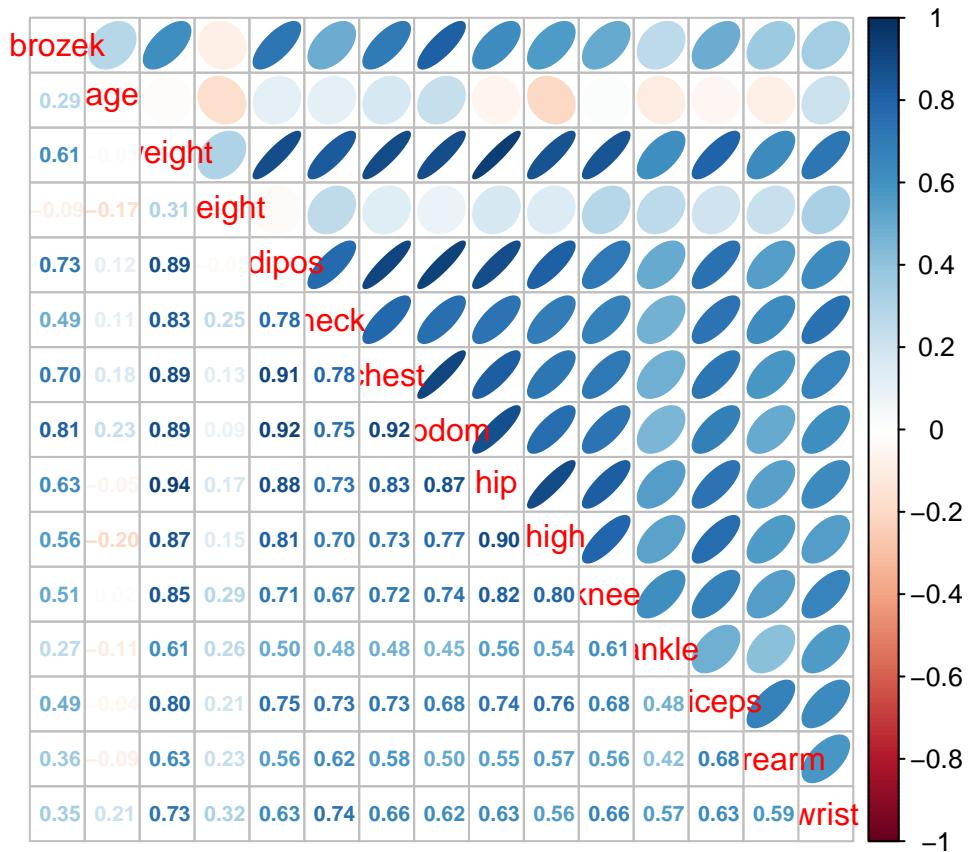
```
##     corrplot
```

*#checking correlation between variables*

```
corrplot(cor(fat1), method = "number", type = "upper", diag = FALSE)
```



```
corrplot.mixed(cor(fat1), upper = "ellipse", lower = "number", number.cex = .7)
```



# Simple Linear Regression

Following practical class 01, conduct simple linear regression modelling.

## Linear regression model fitting

### Exercise 2.2

- Use the `lm()` function to fit a simple linear regression model, with `brozek` as the response variable and `adipos` as the predictor variable. Save the result of the regression function to `reg1`.
- Use `summary(reg1)` command to get information about the model. This gives us the estimated coefficients, t-tests, p-values and standard errors as well as the R-square and F-statistic for the model.
- Use `names(reg1)` function to find what types of information are stored in `reg1`. For example, we can get or extract the estimated coefficients using the function `coef(reg1)` or `reg1$coefficients`.

Click for solution

```
## SOLUTION
# (a)
reg1 <- lm(brozek~adipos,data=fat1)

# (b)
summary(reg1)

##
## Call:
## lm(formula = brozek ~ adipos, data = fat1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.4292  -3.4478   0.2113   3.8663  11.7826
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept) -20.40508   2.36723  -8.62 0.0000000000000778 ***
## adipos        1.54671   0.09212  16.79 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.324 on 250 degrees of freedom
## Multiple R-squared:  0.53, Adjusted R-squared:  0.5281
## F-statistic: 281.9 on 1 and 250 DF,  p-value: < 0.0000000000000022
```

```
# (c)
names(reg1)

## [1] "coefficients"   "residuals"      "effects"       "rank"
## [5] "fitted.values" "assign"         "qr"           "df.residual"
## [9] "xlevels"        "call"          "terms"         "model"

coef(reg1)

## (Intercept) adipos
## -20.405082 1.546712

reg1$coefficients

## (Intercept) adipos
## -20.405082 1.546712
```

## Confidence and prediction intervals

### Exercise 2.3

- Obtain a 95% confidence interval for the coefficient estimates. *Hint:* use the `confint()` command.
- Obtain a 95% confidence and prediction intervals of `brozek` for a given value of `adipos`. Assume this given value of `adipos` is equal to 22.

Click for solution

```
## SOLUTION
# (a)
confint(reg1, level = 0.95)

##               2.5 %    97.5 %
## (Intercept) -25.067331 -15.74283
## adipos       1.365275   1.72815

# (b)
newdata= data.frame(adipos=22)
predict(reg1,newdata=newdata, interval="confidence", level = 0.95)

##       fit     lwr      upr
## 1 13.62259 12.71416 14.53101

predict(reg1,data.frame(adipos=22), interval="prediction", level = 0.95)

##       fit     lwr      upr
## 1 13.62259 3.096772 24.14841
```

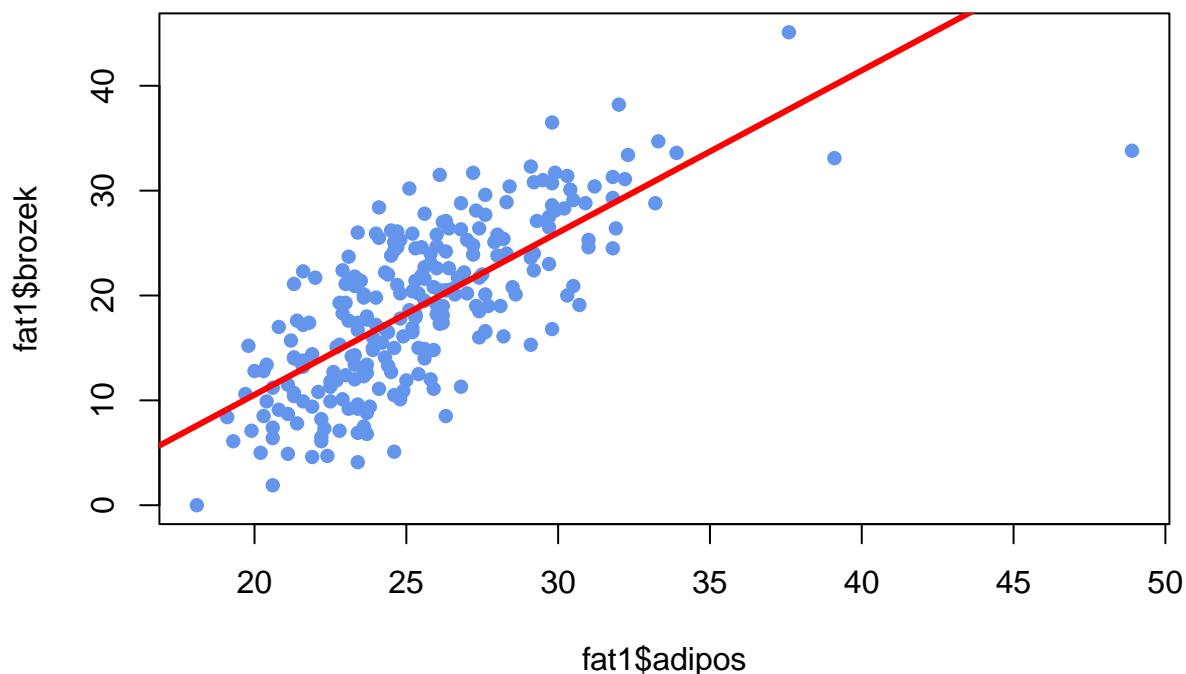
## Regression diagnostics

### Exercise 2.4

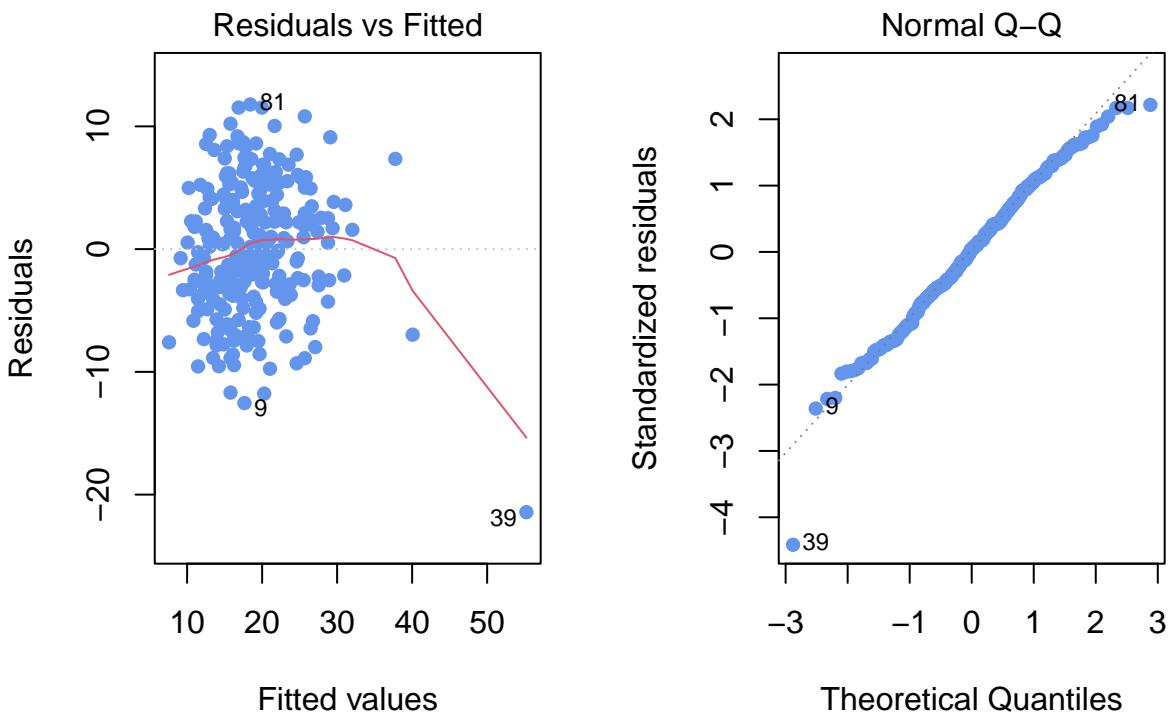
- Create a scatterplot of `adipos` and `brozek`. Is there a relationship between the two variables?
- Plot `brozek` and `adipos` along with the least squares regression line using the `plot()` and `abline()` functions.
- Use residual plots to check the assumptions of the model.

Click for solution

```
## SOLUTION
# (a) & (b)
plot(fat1$adipos, fat1$brozek, pch=16, col="cornflowerblue")
abline(reg1, lwd=3, col="red")
```



```
# (c)
par(mfrow=c(1,2)) # to have two plots side-by-side
plot(reg1, which=1, pch=16, col="cornflowerblue")
plot(reg1, which=2, pch=16, col="cornflowerblue")
```



```
par(mfrow=c(1,1)) # to return to one plot per window
```

## Multiple Linear Regression

### Linear regression model fitting

#### Exercise 2.5

- Use the `lm()` function again to fit a linear regression model, with `brozek` as the response variable and both `adipos` and `age` as predictors. Save the result of the regression function to `reg2`.
- Now use the `lm()` function again to fit a linear regression model, with `brozek` as the response variable and all other 14 variables as predictors. Save the result of the regression function to `reg3`. *Hint:* Use `lm(brozek~., data=fat1)`.
- Fit the same linear regression as in (b) but now exclude the `age`. Save the result of the regression function to `reg4`. *Hint:* Use `lm(brozek~.-age, data=fat1)`.
- Compare the R-squares of these models. Notice we can extract the R-square value from the summary object as `summary(reg)$r.sq`. Type `?summary.lm` to see what else is available.
- Use Variance Inflation Factor (VIF) to assess if there is any relationship between the independent variables. *Hint:* use the `vif()` function from the `car` package.

Click for solution

```
## SOLUTION
# (a)
reg2 <- lm(brozek~adipos+age,data=fat1)
```

```

summary(reg2)

##
## Call:
## lm(formula = brozek ~ adipos + age, data = fat1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -20.3523  -3.6914  -0.0805   3.5328  11.6982
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept) -24.75937   2.43119 -10.184 < 0.0000000000000002 ***
## adipos        1.49481   0.08875  16.842 < 0.0000000000000002 ***
## age          0.12643   0.02569   4.921      0.00000157 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.093 on 249 degrees of freedom
## Multiple R-squared:  0.5716, Adjusted R-squared:  0.5682
## F-statistic: 166.1 on 2 and 249 DF,  p-value: < 0.0000000000000022

```

# (b)

```

reg3 <- lm(brozek~., data=fat1)
summary(reg3)

```

```

##
## Call:
## lm(formula = brozek ~ ., data = fat1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.2573  -2.5919  -0.1031   2.9040   9.2754
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept) -15.1901414 16.1089106 -0.943      0.3467
## age          0.0568795  0.0300278  1.894      0.0594 .
## weight       -0.0813022  0.0498853 -1.630      0.1045
## height       -0.0530707  0.1034409 -0.513      0.6084
## adipos        0.0610056  0.2779614  0.219      0.8265
## neck         -0.4449644  0.2184007 -2.037      0.0427 *
## chest        -0.0308670  0.0977939 -0.316      0.7526
## abdom         0.8789769  0.0854535 10.286 <0.0000000000000002 ***
## hip          -0.2030661  0.1370728 -1.481      0.1398
## thigh         0.2273768  0.1355589  1.677      0.0948 .
## knee        -0.0009927  0.2298076 -0.004      0.9966

```

```

## ankle      0.1572066  0.2075680  0.757      0.4496
## biceps    0.1485112  0.1600277  0.928      0.3543
## forearm   0.4296681  0.1848602  2.324      0.0210 *
## wrist     -1.4792526  0.4966669 -2.978      0.0032 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.996 on 237 degrees of freedom
## Multiple R-squared:  0.749, Adjusted R-squared:  0.7342
## F-statistic: 50.52 on 14 and 237 DF, p-value: < 0.00000000000000022

```

### # (c)

```

reg4 <- lm(brozek~.-age,data=fat1)
summary(reg4)

```

```

##
## Call:
## lm(formula = brozek ~ . - age, data = fat1)
##
## Residuals:
##    Min      1Q  Median      3Q      Max 
## -9.5851 -2.7683 -0.0017  2.8892  9.0632 
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)    
## (Intercept) -16.44809  16.18249 -1.016     0.3105    
## weight      -0.10069  0.04909 -2.051     0.0413 *  
## height      -0.07693  0.10323 -0.745     0.4568    
## adipos       0.05355  0.27944  0.192     0.8482    
## neck        -0.40682  0.21865 -1.861     0.0640 .  
## chest       -0.02395  0.09826 -0.244     0.8076    
## abdom        0.94825  0.07765 12.212 <0.0000000000000002 *** 
## hip         -0.22302  0.13741 -1.623     0.1059    
## thigh        0.13510  0.12719  1.062     0.2892    
## knee         0.10451  0.22417  0.466     0.6415    
## ankle        0.11744  0.20762  0.566     0.5722    
## biceps       0.17082  0.16046  1.065     0.2881    
## forearm      0.37262  0.18338  2.032     0.0433 *  
## wrist        -1.15979  0.46969 -2.469     0.0142 *  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.018 on 238 degrees of freedom
## Multiple R-squared:  0.7452, Adjusted R-squared:  0.7313
## F-statistic: 53.55 on 13 and 238 DF, p-value: < 0.00000000000000022

```

### # (d)

```

summary(reg1)$r.sq

```

```

## [1] 0.5299755
summary(reg2)$r.sq

## [1] 0.5716312
summary(reg3)$r.sq

## [1] 0.7490309
summary(reg4)$r.sq

## [1] 0.7452313
summary(reg4)$r.squared

## [1] 0.7452313
# or the adjusted R Squared
summary(reg4)$adj.r.squared

## [1] 0.7313154
# (e)
#install.packages("car") # you need to do this once
library(car)
vif(reg3)

##      age    weight    height    adipos     neck    chest    abdom     hip
## 2.250902 33.786851 2.256593 16.163444 4.430734 10.684562 13.346689 15.158277
##      thigh     knee    ankle   biceps   forearm     wrist
## 7.961508 4.828828 1.945514 3.674508 2.193390 3.379612

```

## Interaction between variables (Additional)

It is straightforward to include the interaction between variables in the linear model using the `lm()` function. For example, the syntax `chest*abdom` will include `chest`, `abdom` and the interaction term `chest x abdom` as predictors.

```

summary(lm(brozek~chest*abdom, data=fat1))

##
## Call:
## lm(formula = brozek ~ chest * abdom, data = fat1)
##
## Residuals:
##      Min      1Q  Median      3Q      Max
## -9.9568 -3.2831  0.1953  2.8313 11.0567
##
## Coefficients:

```

```

##             Estimate Std. Error t value      Pr(>|t|)
## (Intercept) -95.206407  18.306007 -5.201 0.00000041538850 ***
## chest        0.404148   0.187669   2.154 0.032243 *
## abdom        1.498560   0.204429   7.330 0.00000000000323 ***
## chest:abdom -0.006936   0.001820  -3.810 0.000175 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.332 on 248 degrees of freedom
## Multiple R-squared:  0.6913, Adjusted R-squared:  0.6876
## F-statistic: 185.1 on 3 and 248 DF,  p-value: < 0.0000000000000022

```

We can also include a non-linear transformation of the predictors, using the function `I()`. For example, we can regress the response variable `brozek` on both `adipos` and `adipos^2` as follows.

```
summary(lm(brozek~adipos+I(adipos^2), data=fat1))
```

```

##
## Call:
## lm(formula = brozek ~ adipos + I(adipos^2), data = fat1)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -12.8991 -3.4058 -0.1397  3.7930 11.3518
##
## Coefficients:
##             Estimate Std. Error t value      Pr(>|t|)
## (Intercept) -55.688716  7.963036 -6.993 0.0000000002460 ***
## adipos       4.111738  0.561764   7.319 0.0000000000343 ***
## I(adipos^2) -0.045378  0.009814  -4.624 0.0000605491752 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.12 on 249 degrees of freedom
## Multiple R-squared:  0.5671, Adjusted R-squared:  0.5637
## F-statistic: 163.1 on 2 and 249 DF,  p-value: < 0.0000000000000022

```

We can also apply other transformations, e.g. taking the log of `adipos`.

```
summary(lm(brozek~log(adipos), data=fat1))
```

```

##
## Call:
## lm(formula = brozek ~ log(adipos), data = fat1)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -13.5263 -3.3776  0.0751  3.8273 11.4306

```

```

## 
## Coefficients:
##              Estimate Std. Error t value      Pr(>|t|)    
## (Intercept) -119.233     7.813  -15.26 <0.0000000000000002 *** 
## log(adipos)   42.820     2.419   17.70 <0.0000000000000002 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 5.174 on 250 degrees of freedom
## Multiple R-squared:  0.5562, Adjusted R-squared:  0.5544 
## F-statistic: 313.3 on 1 and 250 DF,  p-value: < 0.0000000000000022

```

## Variable Selection

Please review the practical demonstration 5.4) presented in the lecture.

### Forward Stepwise Regression

#### Exercise 2.6

- Use `regsubsets()` to perform forward stepwise selection, assigning the result to a variable called `fwd`. Note that you will need to (install and) load library `leaps`.
- Look at the summary of the resulting object `fwd` and extract the following statistics for the model of each size: RSS,  $R^2$ ,  $C_p$ , BIC and adjusted  $R^2$ . Combine them into a single matrix.
- How many predictors are included in the models with i) minimum  $C_p$ , ii) minimum BIC, and iii) maximum adjusted  $R^2$ ? Which predictors are included in each case?
- In a single plotting window, generate the following three plots, showing the relationship between the two variables as lines (as seen in the lecture practical demonstration 5.4): i)  $C_p$  against number of predictors, ii) BIC against number of predictors, and iii) adjusted  $R^2$  against number of predictors. Make sure to add relevant labels to the plots. Highlight the optimal model appropriately on each plot using a big red point.
- Use the special `plot()` function for `regsubsets` to visualise the models obtained by  $C_p$ .

Click for solution

```

## SOLUTION
# (a)
library("leaps") # required library for regsubsets() function.
fwd = regsubsets(brozek~, data = fat1, method = 'forward', nvmax = 14)

# (b)

```

```

results = summary(fwd)
results

## Subset selection object
## Call: regsubsets.formula(brozek ~ ., data = fat1, method = "forward",
##      nvmax = 14)
## 14 Variables (and intercept)
##      Forced in Forced out
## age      FALSE      FALSE
## weight   FALSE      FALSE
## height   FALSE      FALSE
## adipos   FALSE      FALSE
## neck     FALSE      FALSE
## chest    FALSE      FALSE
## abdom   FALSE      FALSE
## hip      FALSE      FALSE
## thigh    FALSE      FALSE
## knee     FALSE      FALSE
## ankle   FALSE      FALSE
## biceps  FALSE      FALSE
## forearm FALSE      FALSE
## wrist   FALSE      FALSE
## 1 subsets of each size up to 14
## Selection Algorithm: forward
##      age weight height adipos neck chest abdom hip thigh knee ankle biceps
## 1  ( 1 )   " " " "   " "   " "   " "   " "   "*"   " " " "   " "   " "
## 2  ( 1 )   " " "*"   " "   " "   " "   " "   "*"   " " " "   " "   " "
## 3  ( 1 )   " " "*"   " "   " "   " "   " "   "*"   " " " "   " "   " "
## 4  ( 1 )   " " "*"   " "   " "   " "   " "   "*"   " " " "   " "   " "
## 5  ( 1 )   " " "*"   " "   " "   " "   "*"   " "   "*"   " " " "   " "   " "
## 6  ( 1 )   "*" "*"   " "   " "   " "   "*"   " "   "*"   " " " "   " "   " "
## 7  ( 1 )   "*" "*"   " "   " "   " "   "*"   " "   "*"   " " "*"   " "   " "
## 8  ( 1 )   "*" "*"   " "   " "   " "   "*"   " "   "*"   " *"   " "   " "
## 9  ( 1 )   "*" "*"   " "   " "   " "   "*"   " "   "*"   " *"   " "   " "
## 10 ( 1 )  "*" "*"   " "   " "   " "   "*"   " "   "*"   " *"   " "   "*"   "*"
## 11 ( 1 )  "*" "*"   "*"   " "   " "   "*"   " "   "*"   " *"   " "   "*"   "*"
## 12 ( 1 )  "*" "*"   "*"   " "   " "   "*"   " *"   "*"   " *"   " "   "*"   "*"
## 13 ( 1 )  "*" "*"   "*"   "*"   " "   "*"   " *"   "*"   " *"   " "   "*"   "*"
## 14 ( 1 )  "*" "*"   "*"   "*"   " "   "*"   " *"   "*"   " *"   " *"   "*"   "*"
##      forearm wrist
## 1  ( 1 )   " "   " "
## 2  ( 1 )   " "   " "
## 3  ( 1 )   " "   "*"
## 4  ( 1 )   "*"   "*"
## 5  ( 1 )   "*"   "*"
## 6  ( 1 )   "*"   "*"
## 7  ( 1 )   "*"   "*"

```

```

## 8  ( 1 )   "*"   "*"
## 9  ( 1 )   "*"   "*"
## 10 ( 1 )  "*"   "*"
## 11 ( 1 )  "*"   "*"
## 12 ( 1 )  "*"   "*"
## 13 ( 1 )  "*"   "*"
## 14 ( 1 )  "*"   "*"

RSS = results$rss
r2 = results$rsq
Cp = results$cp
BIC = results$bic
Adj_r2 = results$adjr2

# Combine the calculated criteria values above into a single matrix.
criteria_values <- cbind(RSS, r2, Cp, BIC, Adj_r2)
criteria_values

```

	RSS	r2	Cp	BIC	Adj_r2
[1,]	5094.931	0.6621178	71.075501	-262.3758	0.6607663
[2,]	4241.328	0.7187265	19.617727	-303.0555	0.7164672
[3,]	4108.183	0.7275563	13.279341	-305.5638	0.7242606
[4,]	3994.311	0.7351080	8.147986	-307.1180	0.7308182
[5,]	3950.628	0.7380049	7.412297	-304.3597	0.7326798
[6,]	3913.377	0.7404753	7.079431	-301.2177	0.7341196
[7,]	3853.214	0.7444651	5.311670	-299.5925	0.7371342
[8,]	3819.985	0.7466688	5.230664	-296.2456	0.7383287
[9,]	3805.075	0.7476576	6.296894	-291.7018	0.7382730
[10,]	3793.873	0.7484005	7.595324	-286.9153	0.7379607
[11,]	3786.200	0.7489093	9.114827	-281.8960	0.7374010
[12,]	3785.179	0.7489771	11.050872	-276.4346	0.7363734
[13,]	3784.367	0.7490309	13.000019	-270.9592	0.7353225
[14,]	3784.367	0.7490309	15.000000	-265.4298	0.7342058

# (c)

```
which.min(Cp) # i) 8
```

```
## [1] 8
```

```
which.min(BIC) # ii) 4
```

```
## [1] 4
```

```
which.max(Adj_r2) # iii) 8
```

```
## [1] 8
```

```
coef(fwd, 4)
```

```

## (Intercept)      weight      abdom      forearm      wrist
## -31.2967858   -0.1255654    0.9213725    0.4463824   -1.3917662
# Looking at the summary above, the model with 4 predictors (minimum BIC)
# includes the predictors weight, abdom, forearm and wrist.

coef(fwd, 8)

```

```

## (Intercept)          age      weight      neck      abdom      hip
## -20.06213373  0.05921577 -0.08413521 -0.43189267  0.87720667 -0.18641032
##      thigh      forearm      wrist
##  0.28644340  0.48254563 -1.40486912

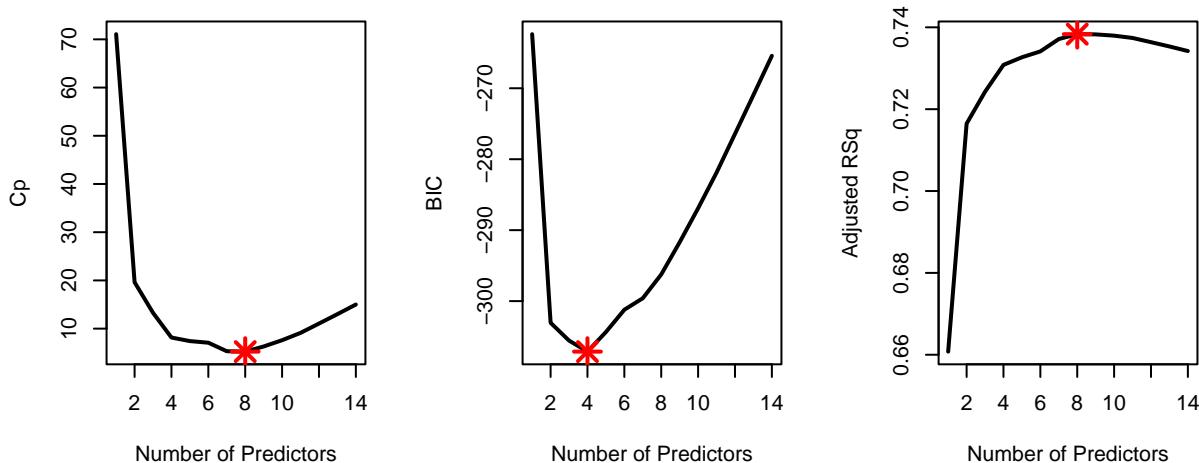
```

# The model with 8 predictors (minimum Cp and maximum adjusted R-squared)  
# includes additionally the predictors age, neck, hip and thigh.

```

# (d)
par(mfrow = c(1, 3))
plot(Cp, xlab = "Number of Predictors", ylab = "Cp", type = 'l', lwd = 2)
points(8, Cp[8], col = "red", cex = 2, pch = 8, lwd = 2)
plot(BIC, xlab = "Number of Predictors", ylab = "BIC", type = 'l', lwd = 2)
points(4, BIC[4], col = "red", cex = 2, pch = 8, lwd = 2)
plot(Adj_r2, xlab = "Number of Predictors", ylab = "Adjusted RSq",
     type = "l", lwd = 2)
points(8, Adj_r2[8], col = "red", cex = 2, pch = 8, lwd = 2)

```



```

par(mfrow = c(1, 1))
plot(fwd, scale = "Cp")

```



## Best Subset and Backward Selection

### Exercise 2.7

Do the best models (as determined by  $C_p$ , BIC and adjusted- $R^2$ ) from best subset and backward stepwise selections have the same number of predictors? If yes, are the predictors the same as those from forward selection? *Hint:* if it is not obvious how to perform backward selection type `?regsubsets` and see information about argument `method`.

Click for solution

```
## SOLUTION
best = regsubsets(brozek~, data = fat1, nvmax = 14)
bwd = regsubsets(brozek~, data = fat1, method = 'backward', nvmax = 14)

which.min(summary(best)$cp)

## [1] 8
which.min(summary(best)$bic)

## [1] 4
which.max(summary(best)$adjr2)

## [1] 8
which.min(summary(bwd)$cp)

## [1] 8
which.min(summary(bwd)$bic)

## [1] 4
which.max(summary(bwd)$adjr2)
```

```

## [1] 8

# Yes, the three optimal models (under each of the criteria Cp, BIC and
# adj-R-squared) for each of forward stepwise, backward stepwise and
# best subset selections all have the same number of predictors.

# Cp
coef(fwd,8)

## (Intercept)      age      weight      neck      abdom      hip
## -20.06213373  0.05921577 -0.08413521 -0.43189267  0.87720667 -0.18641032
##      thigh      forearm      wrist
##  0.28644340   0.48254563 -1.40486912

coef(best,8)

## (Intercept)      age      weight      neck      abdom      hip
## -20.06213373  0.05921577 -0.08413521 -0.43189267  0.87720667 -0.18641032
##      thigh      forearm      wrist
##  0.28644340   0.48254563 -1.40486912

coef(bwd,8)

## (Intercept)      age      weight      neck      abdom      hip
## -20.06213373  0.05921577 -0.08413521 -0.43189267  0.87720667 -0.18641032
##      thigh      forearm      wrist
##  0.28644340   0.48254563 -1.40486912

# BIC
coef(fwd,4)

## (Intercept)      weight      abdom      forearm      wrist
## -31.2967858 -0.1255654   0.9213725   0.4463824 -1.3917662

coef(best,4)

## (Intercept)      weight      abdom      forearm      wrist
## -31.2967858 -0.1255654   0.9213725   0.4463824 -1.3917662

coef(bwd,4)

## (Intercept)      weight      abdom      forearm      wrist
## -31.2967858 -0.1255654   0.9213725   0.4463824 -1.3917662

# In addition, the predictors are also the same.

# adjusted R squared is not needed because it has also 8 predictors

```

# Practical Class Sheets 3

In this practical class, we will apply ridge and lasso regression for multiple linear regression and explore principal component analysis.

*To start, create a new R script by clicking on the corresponding menu button (in the top left); and save it somewhere in your computer. You can now write all your code into this file, and then execute it in R using the menu button **Run** or by simple pressing **ctrl + enter**.*

In this practical class we will investigate the dataset **seatpos** from the library **faraway**.

## Initial Data Analysis

### Exercise 3.1

This exercise focuses on generally improving your data analytical skills which you have been practicing since the first practical class.

- a) Load library **faraway** and check the help file for dataset **seatpos**.
- b) Looking at the help file, which variable might the researchers at the HuMoSim laboratory consider to be a good response variable?
- c) Looking at the help document only for the moment (that is, without performing any analysis on the dataset), are there any predictor variables that we think may be likely to be highly correlated?
- d) What is the dimension of the dataset? Check whether there is any missing data.
- e) Use exploratory data analysis commands and plots to investigate the correlation of **hipcenter** with the other (possible predictor) variables. With which predictor variables is **hipcenter** most correlated? Are those predictor variables correlated with each other? Does this exploratory analysis support your initial thoughts from part (c)? What do your findings here suggest about the fixed location referred to in the help document for description of **hipcenter** (is it in front or behind the driver)?

Click for solution

**## SOLUTION**

```
# (a)
library(faraway)
?seatpos
```

```

# (b)
# hipcenter would be a good response variable, as it is the only variable
# in the dataset that considers the location of a person within the car.
# This can be used as a proxy for where different drivers will position the
# seat, which is the quantity of interest for car designers.

# (c)
# Many of the predictors could be highly correlated.
# Examples include:
# - height and weight (a whole analysis could be done on this
# pairing),
# - lower leg length and thigh length - I would suspect people with
# longer lower legs also have longer thighs, but maybe I'm wrong.
# - Ht and HtShoes - these two variables measure height bare foot and in
# shoes - given that people's heights are in general not going to increase
# much by putting on a pair of shoes (and possibly by roughly a similar
# amount for each person), these variables are likely to be (very) highly
# correlated. I would question including both of these variables
# in an analysis, although our computational analyses will confirm whether
# this is the case.
#
# Perhaps Age is unlikely to be too highly correlated with any of the other
# variables.
# Note that the aim of this question is to get you to think about the
# dataset before we go headlong into applying computational methods, since
# this is what a statistician/data scientist/machine learner should do!

# (d)
dim( seatpos ) # 38 samples, 9 variables.

## [1] 38 9
sum(is.na(seatpos)) # no missing data.

## [1] 0
# (e)
# We can calculate the correlation of `hipcenter` to the other variables.
# Note that the response is the ninth variable in this dataset.
cor(seatpos[,9], seatpos[,-9])

##          Age    Weight   HtShoes        Ht    Seated      Arm    Thigh
## [1,] 0.2051722 -0.640333 -0.7965964 -0.7989274 -0.7312537 -0.585095 -0.5912015
##          Leg
## [1,] -0.7871685

```

```
# From this we see that `hipcenter` has a reasonable negative correlation
# with most of the predictors, apart from age.

# We display the correlation matrix across the eight predictors as follows.
cor( seatpos[,-9] )
```

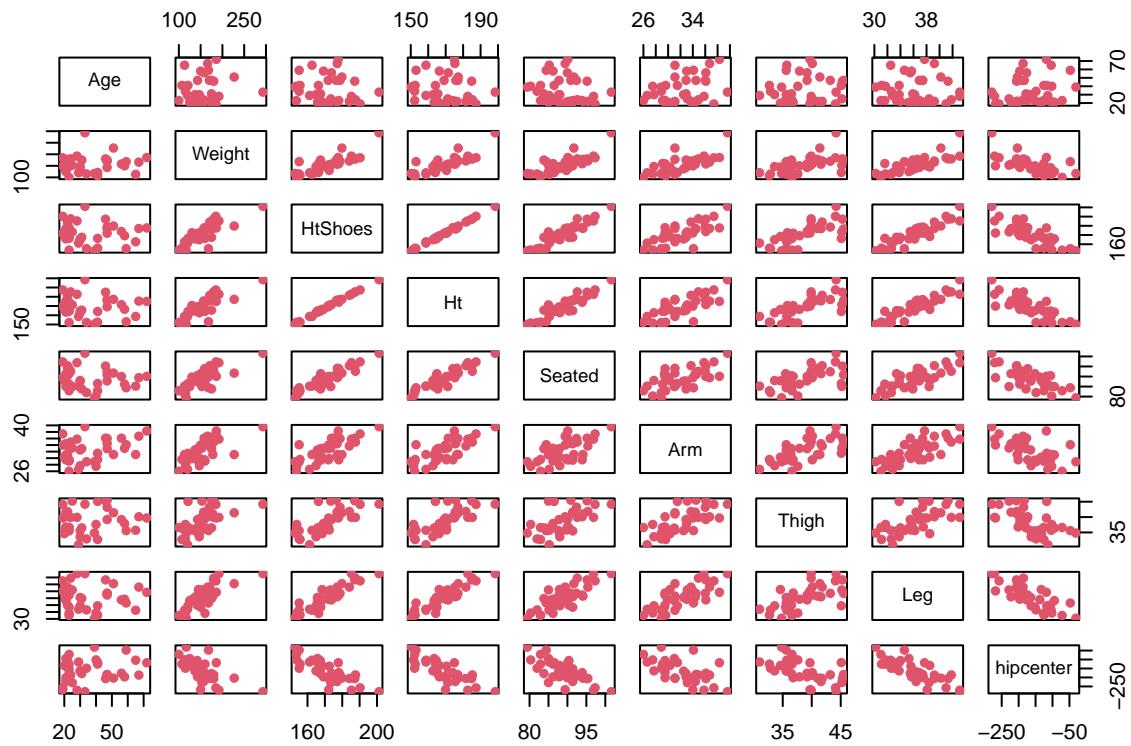
	Age	Weight	HtShoes	Ht	Seated	Arm
## Age	1.00000000	0.08068523	-0.07929694	-0.09012812	-0.1702040	0.3595111
## Weight	0.08068523	1.00000000	0.82817733	0.82852568	0.7756271	0.6975524
## HtShoes	-0.07929694	0.82817733	1.00000000	0.99814750	0.9296751	0.7519530
## Ht	-0.09012812	0.82852568	0.99814750	1.00000000	0.9282281	0.7521416
## Seated	-0.17020403	0.77562705	0.92967507	0.92822805	1.0000000	0.6251964
## Arm	0.35951115	0.69755240	0.75195305	0.75214156	0.6251964	1.0000000
## Thigh	0.09128584	0.57261442	0.72486225	0.73496041	0.6070907	0.6710985
## Leg	-0.04233121	0.78425706	0.90843341	0.90975238	0.8119143	0.7538140
	Thigh	Leg				
## Age	0.09128584	-0.04233121				
## Weight	0.57261442	0.78425706				
## HtShoes	0.72486225	0.90843341				
## Ht	0.73496041	0.90975238				
## Seated	0.60709067	0.81191429				
## Arm	0.67109849	0.75381405				
## Thigh	1.00000000	0.64954120				
## Leg	0.64954120	1.00000000				

# Many of the predictors are reasonably highly correlated with each other,
# except for Age.

# As suspected the pairing with greatest correlation is Ht and HtShoes.

# We could obtain a pairs plot across the dataset.

```
pairs( seatpos, pch = 16, col = 2 )
```



```
# These support the comments and correlations seen above.
```

```
# I would hazard a guess that the fixed location in the car is behind the
# driver, since the distance from it to the drivers hips gets smaller as
# the driver's various dimensions (in general) get larger. Larger drivers
# are likely to move their seat back, so to make the distance shorter, the
# fixed position should be behind the driver.
# Note: these are my interpretations of the dataset, but again the purpose
# of this question is about getting you to think logically about the data
# you have been presented with.
# This can sometimes be harder than the computational side of things.
# In particular it may help to catch any coding errors (and potential
# illogical reasonings and conclusions) later!
```

## Ridge Regression

### Exercise 3.2

- Define a vector  $y$  containing all the values of the response variable `hipcenter`.
- Stack the predictors column-wise into a matrix, and define this matrix as  $x$ .
- Load library `glmnet`. Fit a model of all of the remaining variables for `hipcenter` using ridge regression over a grid of 200  $\lambda$  values. *Hint:* type `?glmnet` and look at argument `nlambda`.
- Plot the regularisation paths of the ridge regression coefficients as a function of  $\log \lambda$ .

Click for solution

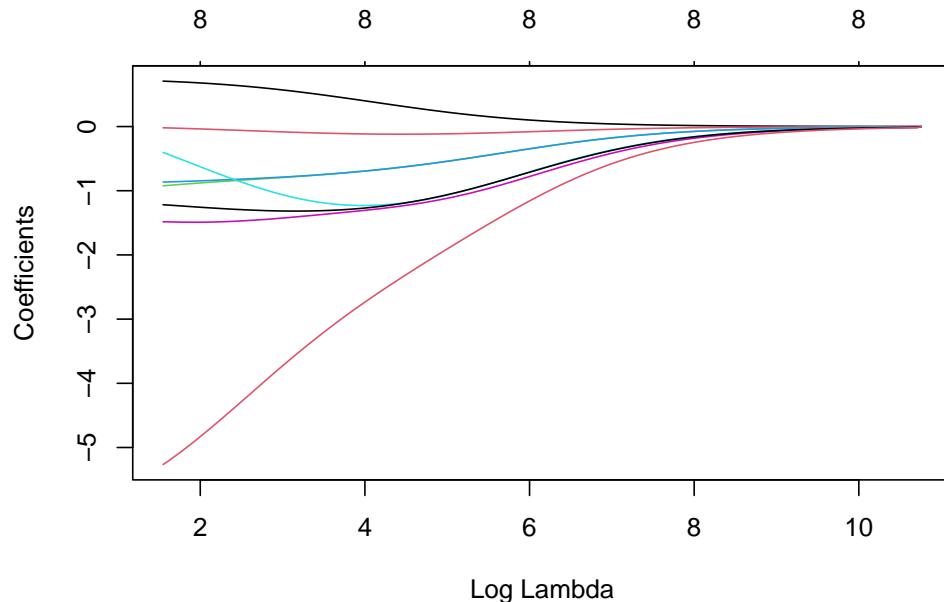
```
## SOLUTION

#(a)
y = seatpos$hipcenter

#(b)
x = model.matrix(hipcenter~., seatpos) [,-1]

#(c)
library("glmnet")
ridge = glmnet(x, y, alpha = 0, nlambda = 200)

#(d)
plot(ridge, xvar = 'lambda')
```



## Lasso Regression

### Exercise 3.3

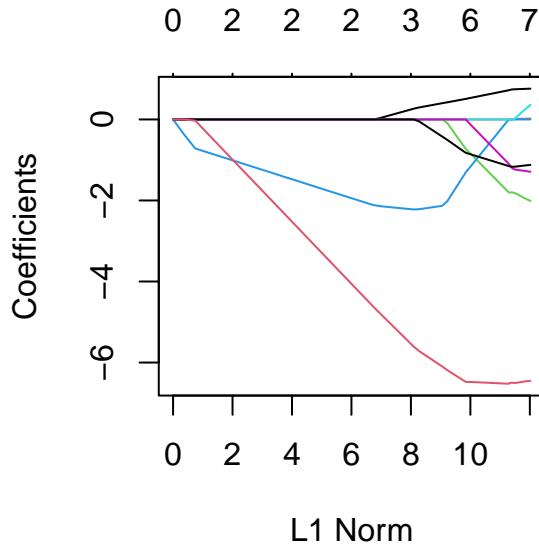
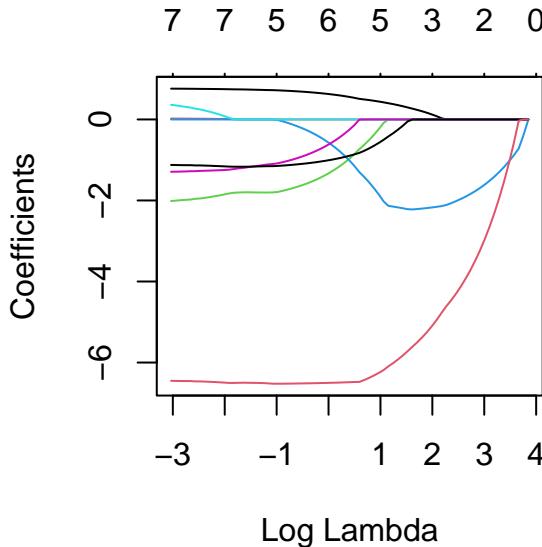
Fit a model of all of the remaining variables for `hipcenter` using lasso regression. Plot regularisation paths against  $\log \lambda$  and  $\ell_1$  norm values. Investigate which variables are included under various values of  $\lambda$ . Do you notice anything interesting about the trends for the predictors `Ht` and `HtShoes`? Do the findings here support exploratory analysis of Exercise 3.1?

Click for solution

```
## SOLUTION
```

```
lasso = glmnet(x, y)
```

```
par(mfrow = c(1,2))
plot(lasso, xvar = 'lambda')
plot(lasso)
```



*# We can see which values of lambda were in the grid.*

```
lasso$lambda
```

```
## [1] 47.02272566 42.84535631 39.03909294 35.57096752 32.41094081 29.53164215
## [7] 26.90813246 24.51768813 22.33960429 20.35501542 18.54673195 16.89909140
## [13] 15.39782269 14.02992256 12.78354291 11.64788819 10.61312191 9.67028141
## [19] 8.81120026 8.02843751 7.31521325 6.66534987 6.07321856 5.53369056
## [25] 5.04209274 4.59416712 4.18603397 3.81415825 3.47531895 3.16658119
## [31] 2.88527084 2.62895133 2.39540254 2.18260158 1.98870527 1.81203418
## [37] 1.65105806 1.50438261 1.37073740 1.24896487 1.13801027 1.03691258
## [43] 0.94479612 0.86086304 0.78438634 0.71470362 0.65121132 0.59335950
## [49] 0.54064708 0.49261748 0.44885470 0.40897969 0.37264706 0.33954212
## [55] 0.30937814 0.28189383 0.25685116 0.23403321 0.21324235 0.19429849
## [61] 0.17703754 0.16131002 0.14697968 0.13392241 0.12202511 0.11118474
## [67] 0.10130739 0.09230752 0.08410718 0.07663533 0.06982726 0.06362399
## [73] 0.05797181 0.05282176 0.04812922
```

*# We can see which variables were included under each of those.*

```
lasso$beta
```

```
## 8 x 75 sparse Matrix of class "dgCMatrix"
## [[ suppressing 75 column names 's0', 's1', 's2' ... ]]
##
## Age     .   .
## Weight  .   .
## HtShoes .   .
## Ht      . -0.3788888 -0.71690000 -0.8815363 -1.031066 -1.168389 -1.292464
## Seated .   .
```

```

## Arm      . . . . .
## Thigh    . . . . .
## Leg     . . -0.03001398 -0.5709448 -1.065255 -1.512438 -1.923026
##
## Age     . . . . .
## Weight   . . . . .
## HtShoes  . . . . .
## Ht       -1.406576 -1.509502 -1.604347 -1.689716 -1.768569 -1.839363 -1.904943
## Seated   . . . . .
## Arm     . . . . .
## Thigh    . . . . .
## Leg     -2.293971 -2.635099 -2.942743 -3.226199 -3.481281 -3.716848 -3.928280
##
## Age     . . . . 0.03026313 0.07914329
## Weight   . . . . .
## HtShoes  . . . . .
## Ht       -1.963641 -2.017036 -2.066746 -2.111083 -2.13956040 -2.15657249
## Seated   . . . . .
## Arm     . . . . .
## Thigh    . . . . .
## Leg     -4.124080 -4.302752 -4.462389 -4.610703 -4.77565830 -4.94857151
##
## Age     0.123651 0.164252 0.2011991 0.2349099 0.2656318 0.29557296
## Weight   . . . . .
## HtShoes  . . . . .
## Ht       -2.172738 -2.186422 -2.1999306 -2.2112247 -2.2213847 -2.21537008
## Seated   . . . . .
## Arm     . . . . .
## Thigh    . . . . -0.04904616
## Leg     -5.104146 -5.249008 -5.3779072 -5.4983751 -5.6085302 -5.71786751
##
## Age     0.3254539 0.3526374 0.3774918 0.4000568 0.42215302 0.4425705
## Weight   . . . . .
## HtShoes  . . . . -0.08747275 -0.2410930
## Ht       -2.1919535 -2.1710512 -2.1509204 -2.1336034 -2.02086133 -1.8495163
## Seated   . . . . .
## Arm     . . . . .
## Thigh    -0.1550600 -0.2512560 -0.3396447 -0.4194801 -0.50353183 -0.5808560
## Leg     -5.8252121 -5.9220281 -6.0129224 -6.0932133 -6.18702261 -6.2568841
##
## Age     0.4611742 0.4781344 0.4935876 0.507660431 0.5325412 0.5538978
## Weight   . . . . .
## HtShoes -0.3810765 -0.5094711 -0.6264423 -0.732315652 -0.8541160 -0.9612630
## Ht       -1.6933835 -1.5502583 -1.4198654 -1.301776165 -1.1523513 -1.0218632
## Seated   . . . . .
## Arm     . . . -0.003724438 -0.1356874 -0.2445963
## Thigh   -0.6513109 -0.7155662 -0.7741120 -0.825232452 -0.8617522 -0.8945363

```

```

## Leg      -6.3205372 -6.3785468 -6.4314027 -6.478380251 -6.4798559 -6.4847698
##
## Age      0.5733715  0.5910974  0.6072458  0.6219709  0.6353911  0.6476155
## Weight   .
## HtShoes  -1.0599721 -1.1485746 -1.2290941 -1.3032866 -1.3710911 -1.4325446
## Ht       -0.9018626 -0.7938899 -0.6957256 -0.6054368 -0.5229599 -0.4481416
## Seated   .
## Arm      -0.3438619 -0.4342699 -0.5166409 -0.5917202 -0.6601388 -0.7224752
## Thigh    -0.9244800 -0.9516743 -0.9764387 -0.9990584 -1.0196829 -1.0384553
## Leg      -6.4892510 -6.4933288 -6.4970433 -6.5004312 -6.5035202 -6.5063374
##
## Age      0.6587657  0.6689108  0.6781749  0.6865932  0.6942714  0.7012737
## Weight   .
## HtShoes  -1.4893806 -1.5398858 -1.5875253 -1.6288279 -1.6673273 -1.7029099
## Ht       -0.3791066 -0.3175022 -0.2597176 -0.2091840 -0.1622689 -0.1190003
## Seated   .
## Arm      -0.7793014 -0.8310560 -0.8782503 -0.9212114 -0.9603579 -0.9960311
## Thigh    -1.0556179 -1.0711783 -1.0854622 -1.0983601 -1.1101614 -1.1209538
## Leg      -6.5089124 -6.5112740 -6.5134288 -6.5154515 -6.5172794 -6.5189702
##
## Age      0.70765527 0.71346592 0.7187507 0.7228005 0.7250685 0.7283275
## Weight   .
## HtShoes  -1.73553063 -1.76517488 -1.7918480 -1.8012793 -1.8042393 -1.8036976
## Ht       -0.07936592 -0.04332879 -0.0108381 .
## Seated   .
## Arm      -1.02852223 -1.05809285 -1.0849764 -1.1006536 -1.1107097 -1.1284351
## Thigh    -1.13080691 -1.13978543 -1.1479486 -1.1525898 -1.1535925 -1.1565048
## Leg      -6.52053931 -6.52200063 -6.5233662 -6.5216161 -6.5131065 -6.5068898
##
## Age      0.731506  0.734511  0.737307  0.739810222 0.741186412 0.742529970
## Weight   .
## HtShoes  -1.802075 -1.800008 -1.797816 -1.798603166 -1.803557433 -1.808553585
## Ht       .
## Seated   .
## Arm      -1.145882 -1.162368 -1.177701 -1.194759877 -1.207604764 -1.224531973
## Thigh    -1.159510 -1.162575 -1.165548 -1.166929365 -1.167273424 -1.167232842
## Leg      -6.503318 -6.501111 -6.499640 -6.497567833 -6.499442317 -6.503591032
##
## Age      0.745354326 0.74742495 0.7490595 0.75051743 0.75188196 0.75309161
## Weight   0.008513292 0.01013531 0.0117728 0.01326849 0.01463865 0.01587222
## HtShoes  -1.827269779 -1.85186185 -1.8757287 -1.89667200 -1.91673747 -1.93407617
## Ht       .
## Seated   0.040515474 0.08294766 0.1225952 0.15754489 0.19096305 0.21992109
## Arm      -1.239936898 -1.24773622 -1.2530833 -1.25803785 -1.26232015 -1.26641490
## Thigh    -1.163875042 -1.15825820 -1.1526247 -1.14780561 -1.14306697 -1.13909162
## Leg      -6.502812684 -6.49475353 -6.4872853 -6.48122990 -6.47512351 -6.47010118
##

```

```

## Age      0.75422793  0.75592912  0.75628200  0.75774078  0.75870481  0.75951416
## Weight   0.01700937  0.01736582  0.01888635  0.01919551  0.01980897  0.02057163
## HtShoes -1.95075883 -1.95798264 -1.97774056 -1.98463485 -1.99295752 -2.00426952
## Ht       .
## Seated   0.24771599  0.26649454  0.29335360  0.31078418  0.32692789  0.34634309
## Arm      -1.26997742 -1.27709493 -1.27745387 -1.28286904 -1.28666067 -1.28881721
## Thigh    -1.13514900 -1.13506802 -1.12899125 -1.12874195 -1.12739244 -1.12473708
## Leg      -6.46501672 -6.46519803 -6.45672861 -6.45697888 -6.45593029 -6.45313255
##
## Age      0.76009998
## Weight   0.02118573
## HtShoes -2.01275176
## Ht       .
## Seated   0.36049535
## Arm      -1.29085662
## Thigh    -1.12283674
## Leg      -6.45077390

```

# In particular, high values of lambda leads to inclusion of Ht only.  
# Other variables start to come in; leg, followed by Age,  
# followed by Thigh, as lambda gets smaller.  
# It is interesting that Age comes in third, despite having a reasonably  
# small correlation with the response.  
# However, this may be as expected, since the other variables also have  
# high correlations with each other (collinearity), whereas Age may  
# contain alternative predictive information.  
# Interestingly, HtShoes enters only for small enough values of lambda, at  
# which point its coefficients increase whilst the Ht coefficients  
# decrease, eventually to zero.  
# This is to do with the ridiculously high correlation between Ht  
# and HtShoes (spotted in Exercise 3.1) – even for really low values of lambda,  
# we don't need both of these variables.

## Principal Component Analysis

### Exercise 3.4

- Conduct Principal Component Analysis to all the predictors  $\mathbf{x}$  as in Exercise 3.2 with all variables being scaled to have unit variance.
- Report the variance of each of the principal component and the eigenvectors that corresponding to the first principal component.
- Draw a scree plot. How many components would you need in order to capture at least 95% of the total variance of the data cloud?
- Compress the data using the number of principal components as in (c), then reconstruct the data and compare the reconstruct data with the original data by looking at Age

and weight.

Click for solution

```
## SOLUTION

(a)
s <- apply(x, 2, sd)      # calculates the column standard deviations
x.s <- sweep(x, 2, s, "/") # divides all columns by their standard deviations
seatpos.pr <- prcomp(x.s)

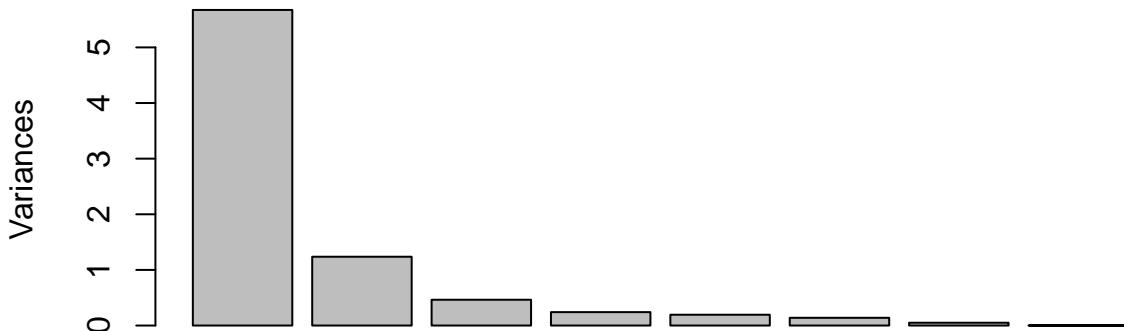
(b)
seatpos.pr

## Standard deviations (1, ..., p=8):
## [1] 2.38184501 1.11210881 0.68098711 0.49087508 0.44070349 0.37306059 0.22437586
## [8] 0.03985271
##
## Rotation (n x k) = (8 x 8):
##          PC1       PC2       PC3       PC4       PC5       PC6
## Age     -0.007219379  0.8763467  0.16383976 -0.16522774 -0.3349932 -0.25464449
## Weight  -0.366979122  0.0448877  0.42981137 -0.60025209  0.5537489  0.09798202
## HtShoes -0.411460536 -0.1055831  0.03375209  0.02577245 -0.2204816 -0.05101900
## Ht      -0.412057421 -0.1119799  0.01116858  0.02294603 -0.1887759 -0.04369735
## Seated  -0.381270226 -0.2178995  0.17138740 -0.15033847 -0.6171009  0.23019712
## Arm    -0.348771387  0.3742641 -0.01670980  0.55358297  0.2380225  0.60781701
## Thigh   -0.327523319  0.1251793 -0.86246173 -0.31151283  0.1038969 -0.06344739
## Leg    -0.389747512 -0.0555930  0.11688322  0.43024468  0.2205229 -0.70326773
##          PC7       PC8
## Age     0.02269849 -0.015528966
## Weight -0.04435483  0.008082356
## HtShoes 0.53650776  0.691876699
## Ht      0.50884054 -0.721493879
## Seated -0.56689080 -0.002844309
## Arm    -0.07347462  0.007539785
## Thigh  -0.14492761  0.018814564
## Leg    -0.32092126  0.005274142

seatpos.pr$rotation[,1]

##          Age       Weight       HtShoes        Ht       Seated       Arm
## -0.007219379 -0.366979122 -0.411460536 -0.412057421 -0.381270226 -0.348771387
##          Thigh       Leg
## -0.327523319 -0.389747512

(c)
plot(seatpos.pr)
```

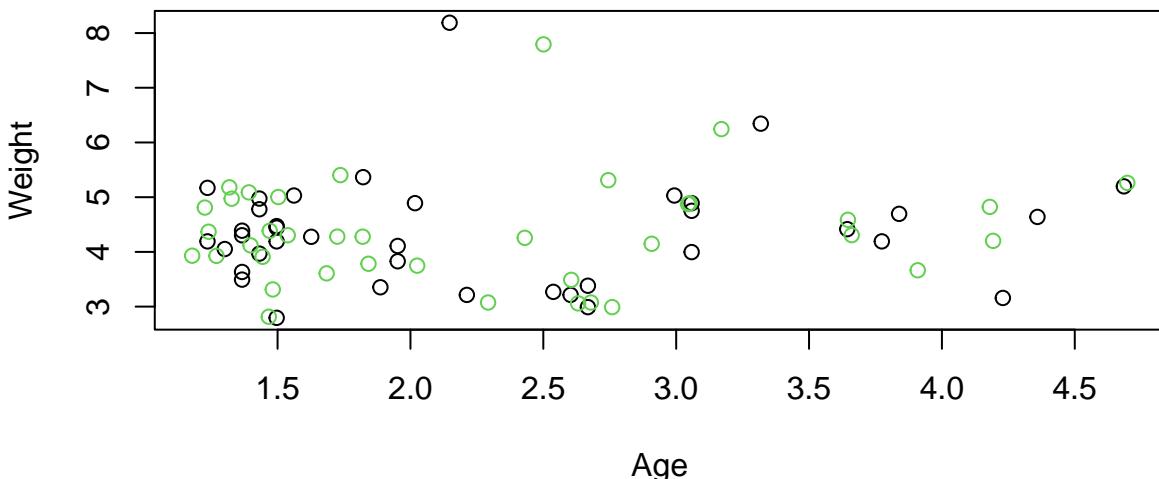
**seatpos.pr**

```
summary(seatpos.pr)
```

```
## Importance of components:
##                               PC1      PC2      PC3      PC4      PC5      PC6      PC7
## Standard deviation     2.3818  1.1121  0.68099 0.49088 0.44070 0.3731  0.22438
## Proportion of Variance 0.7091  0.1546  0.05797 0.03012 0.02428 0.0174  0.00629
## Cumulative Proportion  0.7091  0.8638  0.92171 0.95183 0.97611 0.9935  0.99980
##                               PC8
## Standard deviation     0.03985
## Proportion of Variance 0.00020
## Cumulative Proportion 1.00000

# 4 components are needed to capture at least
# 95% of the total variance of the data cloud
```

```
#(d)
T <- t(seatpos.pr$x[,c(1,2,3,4)]) #compressed using 4 PCs
ms <- colMeans(x.s)
R <- t(ms + seatpos.pr$rot[,c(1,2,3,4)]%*% T) #reconstruction
plot(rbind(x.s[,1:2], R[,1:2]), col=c(rep(1,38),rep(3,38)))
```



# Principal Component Regression

## Exercise 3.5

- Load library `pls` (required for PCR). Use the command `pcr` to fit a principal component regression model to `hipcenter` based on the remaining variables. Select the number of principal components using cross-validation. Remember to standardise the variables. How many principal components are recommended, according to the cross-validation error?
- Appropriately plot the mean squared validation error.
- Use the command `coef()` to find corresponding coefficient estimates in terms of the original  $\hat{\beta}$ 's. How do these estimates coincide with the exploratory analysis of Exercise 3.1?

Click for solution

```
## SOLUTION

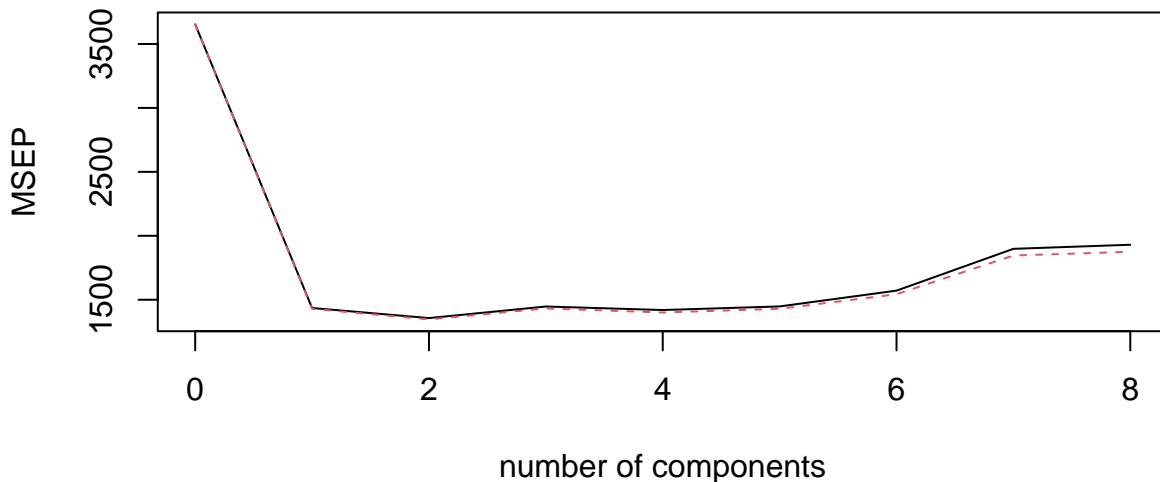
#(a)
library("pls")
pcr.fit=pcr(hipcenter~., data=seatpos, scale = TRUE, validation = "CV" )
summary(pcr.fit)

## Data:      X dimension: 38 8
##  Y dimension: 38 1
## Fit method: svdpc
## Number of components considered: 8
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##              (Intercept) 1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           60.45    37.89    36.83    38.04    37.68    38.05    39.64
## adjCV        60.45    37.80    36.69    37.84    37.42    37.81    39.28
##              7 comps  8 comps
## CV           43.57    43.93
## adjCV        42.96    43.30
##
## TRAINING: % variance explained
##              1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X            70.91    86.37    92.17    95.18    97.61    99.35    99.98
## hipcenter   61.89    66.34    66.48    67.82    68.02    68.58    68.63
##              8 comps
## X            100.00
## hipcenter   68.66

# The summary suggests use of 2 principal components.
# Notice this is different from the number in Exercise 3.4c
```

```
#(b)
validationplot( pcr.fit, val.type = 'MSEP' )
```

### hipcenter



```
#(c)
coef(pcr.fit, ncomp = 2)
```

```
## , , 2 comps
##
##          hipcenter
## Age      9.779064
## Weight -6.721580
## HtShoes -9.301406
## Ht      -9.385585
## Seated -9.978190
## Arm     -2.633940
## Thigh   -5.035274
## Leg     -8.307696
```

# Since Age had a positive correlation with hipcenter, the coefficient  
# is positive. For the remaining variables the correlation was negative,  
# so the coefficients are negative.  
# Notice how the size of the coefficients is relatively similar across  
# all predictor variables.  
# This is to do with how Principal Component Analysis works.

## Predictive Performance of Methods

### Exercise 3.6

Use 50 repetitions of data-splitting with 28 samples for training (10 for testing) to compare the predictive performance (e.g. Correlation and MSE) of the following models of the remaining predictors for **hipcenter**:

- Best subset selection with  $C_p$
- Ridge with min-CV  $\lambda$  (5 folds)
- Lasso with min-CV  $\lambda$  (5 folds)
- PCR with min-CV number of principal components.

Click for solution

```
# Load R libraries.
library(leaps)

# Predict function for regsubsets
predict.regsubsets = function(object, newdata, id, ...){
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id = id)
  xvars = names(coefi)
  mat[, xvars] %*% coefi
}

repetitions = 50
cor.bss = c()
cor.ridge = c()
cor.lasso = c()
cor.pcr = c()

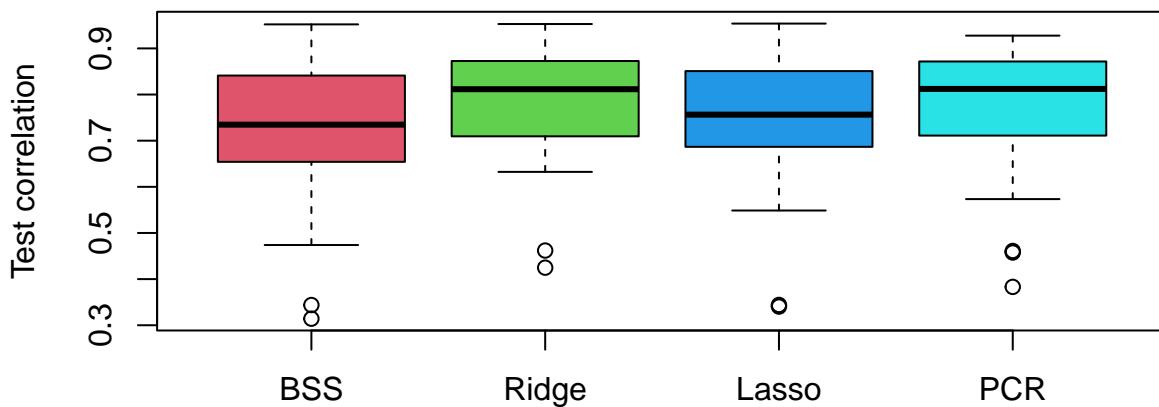
set.seed(1)
for(i in 1:repetitions){
  # Step (i) data splitting
  training.obs = sample(1:38, 28)
  y.train = seatpos$hipcenter[training.obs]
  x.train = model.matrix(hipcenter~, seatpos[training.obs, ])[,-1]
  y.test = seatpos$hipcenter[-training.obs]
  x.test = model.matrix(hipcenter~, seatpos[-training.obs, ])[,-1]

  # Step (ii) training phase
  bss.train = regsubsets(hipcenter~, data=seatpos[training.obs,], nvmax=8)
  min.cp = which.min(summary(bss.train)$cp)
  ridge.train = cv.glmnet(x.train, y.train, alpha = 0, nfolds = 5)
  lasso.train = cv.glmnet(x.train, y.train, nfold = 5)
  pcr.train = pcr(hipcenter~, data = seatpos[training.obs, ],
                  scale = TRUE, validation="CV")
  min.pcr = which.min(MSEP(pcr.train)$val[1,1, ] ) - 1

  # Step (iii) generating predictions
  predict.bss = predict.regsubsets(bss.train, seatpos[-training.obs, ], min.cp)
  predict.ridge = predict(ridge.train, x.test, s = 'lambda.min')
  predict.lasso = predict(lasso.train, x.test, s = 'lambda.min')
  predict.pcr = predict(pcr.train, seatpos[-training.obs, ], ncomp = min.pcr )
```

```
# Step (iv) evaluating predictive performance
cor.bss[i] = cor(y.test, predict.bss)
cor.ridge[i] = cor(y.test, predict.ridge)
cor.lasso[i] = cor(y.test, predict.lasso)
cor.pcr[i] = cor(y.test, predict.pcr)
}

# Plot the resulting correlations as boxplots.
boxplot(cor.bss, cor.ridge, cor.lasso, cor.pcr,
        names = c('BSS', 'Ridge', 'Lasso', 'PCR'),
        ylab = 'Test correlation', col = 2:5)
```





# Practical Class Sheets 4

In this practical class, we will explore polynomial regression, step-wise functions, splines, generalised additive models and logistic regression.

*To start, create a new R script by clicking on the corresponding menu button (in the top left); and save it somewhere in your computer. You can now write all your code into this file, and then execute it in R using the menu button **Run** or by simple pressing **ctrl + enter**.*

In this practical class we will investigate two datasets for modelling. Firstly the dataset **seatpos** from the library **faraway**, followed by the **Boston** data analysed in the lecture practical demonstrations. Lastly we will apply logistic regression to address a binary classification problem.

Following practical class 3, we continue investigating the dataset **seatpos**.

```
library("MASS")
library("faraway")
```

## Polynomial and Step-wise Function Regression

### Exercise 4.1

This exercise models the **seatpos** data by polynomial and step-wise function regression.

We will analyse the effects of predictor variable **Ht** on the response variable **hipcenter**. Assign the **hipcenter** values to a vector **y**, and the **Ht** variable values to a vector **x**.

```
y = seatpos$hipcenter
x = seatpos$Ht
```

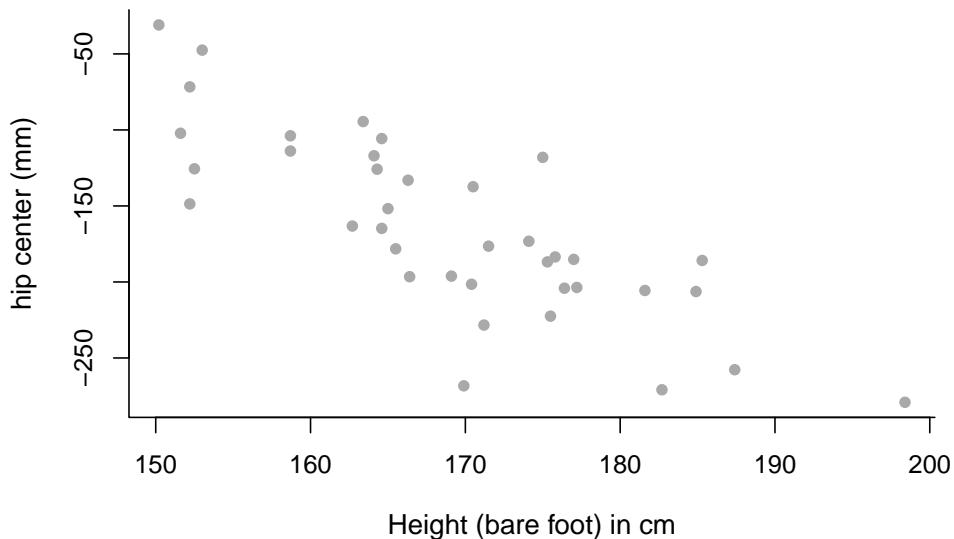
- a) Plot variable **Ht** and **hipcenter** against each other. Remember to include suitable axis labels. From visual inspection of this plot, what sort of polynomial might be appropriate for this data?
- b) Fit a first and second order polynomial to the data using the commands **lm** and **poly**. Look at the corresponding summary objects. Do these back up your answer to part (a)?
- c) Plot the first and second polynomial fits to the data, along with  $\pm 2$  standard deviation confidence intervals, similar to those generated in the lecture practical demonstrations. What do you notice about the degree-2 polynomial plot?

- d) Use step function regression with 5 cut-points to model **hipcenter** based on **Ht**. Plot the results.

Click for solution

```
## SOLUTION
```

```
# (a)
y.lab = 'hip center (mm)'
x.lab = 'Height (bare foot) in cm'
plot( x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "", bty = 'l', pch = 16 )
```



```
# mean trend in the data looks almost linear - perhaps only a
# first-order polynomial model is needed.
```

```
# (b)
poly1 = lm(y ~ poly(x, 1))
summary(poly1)

##
## Call:
## lm(formula = y ~ poly(x, 1))
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -99.956 -27.850   5.656  20.883  72.066 
## 
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)    
## (Intercept) -164.88      5.90 -27.95 < 0.0000000000000002 ***
## poly(x, 1)   -289.87     36.37  -7.97 0.000000000183 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```

## 
## Residual standard error: 36.37 on 36 degrees of freedom
## Multiple R-squared:  0.6383, Adjusted R-squared:  0.6282
## F-statistic: 63.53 on 1 and 36 DF,  p-value: 0.000000001831

poly2 = lm(y ~ poly(x, 2))
summary(poly2)

## 
## Call:
## lm(formula = y ~ poly(x, 2))
## 
## Residuals:
##      Min       1Q   Median       3Q      Max
## -96.068 -25.018   4.418  22.790  75.322
## 
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept) -164.885     5.919 -27.855 < 0.0000000000000002 ***
## poly(x, 2)1 -289.868    36.490  -7.944 0.000000000242 ***
## poly(x, 2)2   31.822    36.490   0.872  0.389
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 36.49 on 35 degrees of freedom
## Multiple R-squared:  0.646, Adjusted R-squared:  0.6257
## F-statistic: 31.93 on 2 and 35 DF,  p-value: 0.00000001282

# p-value for second order term is high, suggesting that it is not necessary in the model. This backs up our conclusion to part (a).

# (c)
sort.x = sort(x) # sorted values of x.

# Predicted values.
pred1 = predict(poly1, newdata = list(x = sort.x), se = TRUE)
pred2 = predict(poly2, newdata = list(x = sort.x), se = TRUE)

# Confidence interval bands.
se.bands1 = cbind( pred1$fit - 2*pred1$se.fit, pred1$fit + 2*pred1$se.fit )
se.bands2 = cbind( pred2$fit - 2*pred2$se.fit, pred2$fit + 2*pred2$se.fit )

# Plot both plots on a single graphics device.
par(mfrow = c(1,2))

# Degree-1 polynomial plot.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Degree-1 polynomial", bty = 'l')

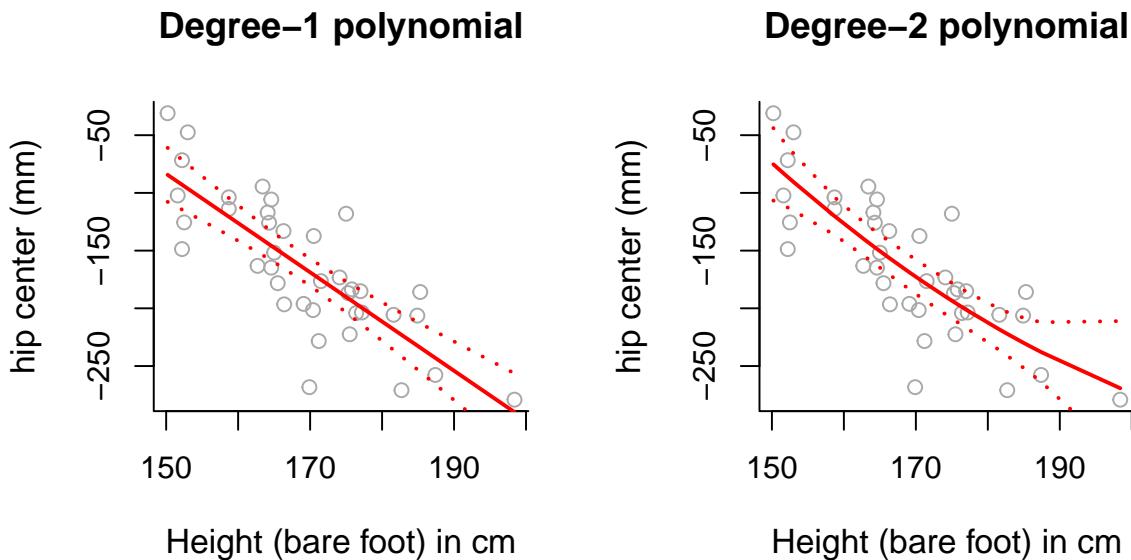
```

```

lines(sort.x, pred1$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands1, lwd = 2, col = "red", lty = 3)

# Degree-2 polynomial plot.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Degree-2 polynomial", bty = 'l')
lines(sort.x, pred2$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands2, lwd = 2, col = "red", lty = 3)

```



# Degree-2 polynomial plot looks relatively close to a straight line as  
# well - perhaps this suggests the second order component isn't necessary?

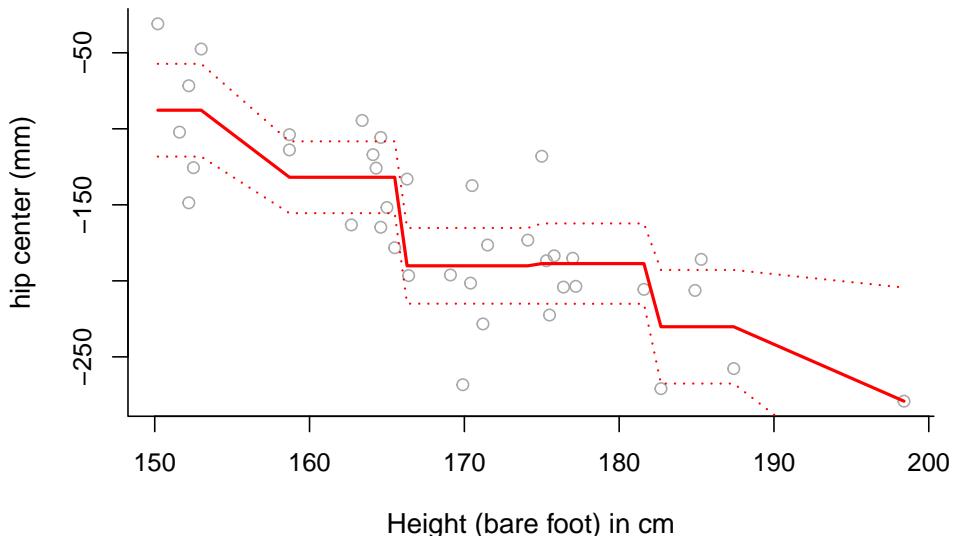
```

# (d)
# Remember to define the number of intervals (one more than the number of
# cut-points).
step6 = lm(y ~ cut(x, 6))
pred6 = predict(step6, newdata = list(x = sort(x)), se = TRUE)
se.bands6 = cbind(pred6$fit + 2*pred6$se.fit, pred6$fit-2*pred6$se.fit)

# Plot the results.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "5 cutpoints", bty = 'l')
lines(sort(x), pred6$fit, lwd = 2, col = "red")
matlines(sort(x), se.bands6, lwd = 1.4, col = "red", lty = 3)

```

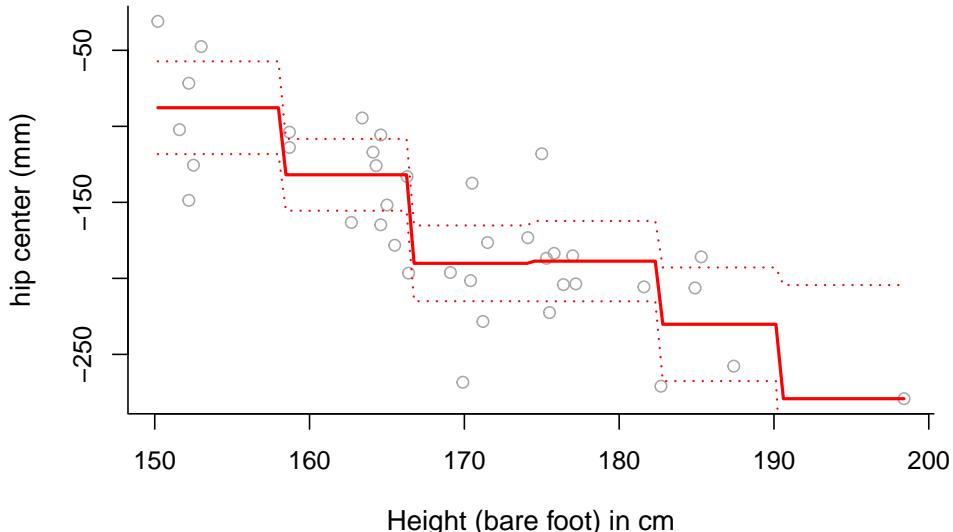
### 5 cutpoints



```
# Note that the slopes in this plot are an artifact of the way in which we
# have plotted the results.
# Use summary to see where the different intervals start and finish.
summary(step6)
```

```
##
## Call:
## lm(formula = y ~ cut(x, 6))
##
## Residuals:
##    Min     1Q     Median      3Q     Max 
## -78.209 -25.615    0.936   22.425   70.623 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -87.76     15.25  -5.753 0.00000222 ***
## cut(x, 6)(158,166] -44.11     19.29  -2.286  0.029 *  
## cut(x, 6)(166,174] -102.35    19.69  -5.197 0.00001119 *** 
## cut(x, 6)(174,182] -100.91    20.18  -5.001 0.00001983 *** 
## cut(x, 6)(182,190] -142.44    24.12  -5.906 0.00000143 *** 
## cut(x, 6)(190,198] -191.39    40.36  -4.742 0.00004197 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 37.36 on 32 degrees of freedom
## Multiple R-squared:  0.6606, Adjusted R-squared:  0.6076 
## F-statistic: 12.46 on 5 and 32 DF,  p-value: 0.0000009372
#
# The fitted model can be more accurately fitted over the data as follows:
newx <- seq(from = min(x), to = max(x), length = 100)
pred6 = predict(step6, newdata = list(x = newx), se = TRUE)
```

```
se.bands6 = cbind(pred6$fit + 2*pred6$se.fit, pred6$fit-2*pred6$se.fit)
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
     main = "5 cutpoints", bty = 'l')
lines(newx, pred6$fit, lwd = 2, col = "red")
matlines(newx, se.bands6, lwd = 1.4, col = "red", lty = 3)
```

**5 cutpoints**

```
# Whilst there is still some sloping artifact to the plot, it is more
# negligible now - that is, the plot of the fitted values more accurately
# represents the step-wise nature of the model.
# Note that this was also present in the examples in
# Section 9.1 and 9.2 - but less noticeable due to the amount of data.
# Change length = 1000 in the command seq for newx and you will see the
# steps become more step-like still!
```

## Splines

In this exercise we continue modeling the `seatpos` data by splines. We will again analyse the effects of predictor variable `Ht` on the response variable `hipcenter`.

### Exercise 4.2

- Find the 25th, 50th and 75th percentiles of `x`, storing them in a vector `cuts`.
- Use a linear spline to model `hipcenter` as a function of `Ht`, putting knots at the 25th, 50th and 75th percentiles of `x`.
- Plot the fitted linear spline from part (b) over the data, along with  $\pm 2$  standard deviation confidence intervals.
- Use a smoothing spline to model `hipcenter` as a function of `Ht`, selecting  $\lambda$  with cross-validation, and generate a relevant plot. What do you notice?

Click for solution

```
## SOLUTION

## (a)
summary(x)

##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.
##    150.2    163.6   169.5    169.1   175.7   198.4

cuts <- summary(x)[c(2,3,5)]

## (b)
library("splines")
spline1 = lm(y ~ bs(x, degree = 1, knots = cuts))

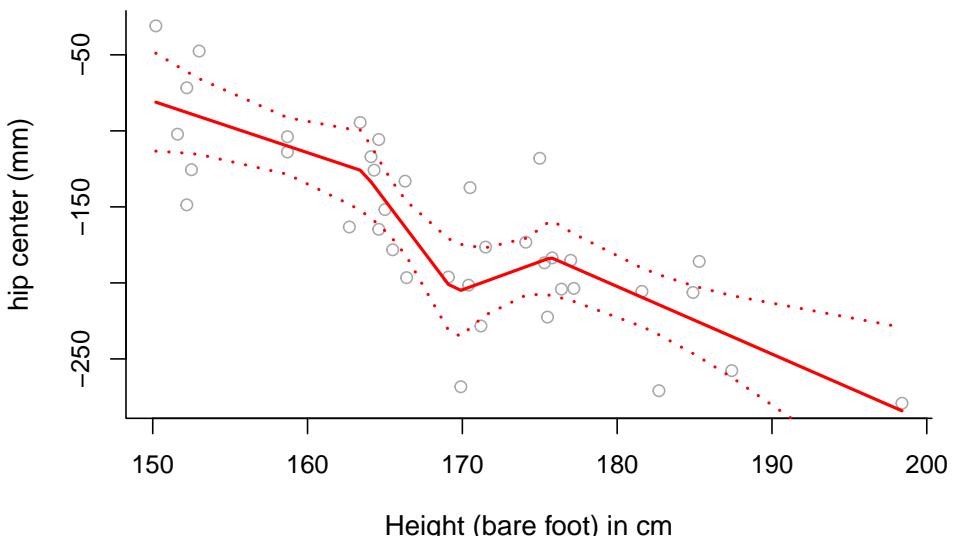
## (c)

# Sort the values of x from low to high.
sort.x <- sort(x)

# Obtain predictions for the fitted values along with confidence intervals.
pred1 = predict(spline1, newdata = list(x = sort.x), se = TRUE)
se.bands1 = cbind(pred1$fit + 2 * pred1$se.fit, pred1$fit - 2 * pred1$se.fit)

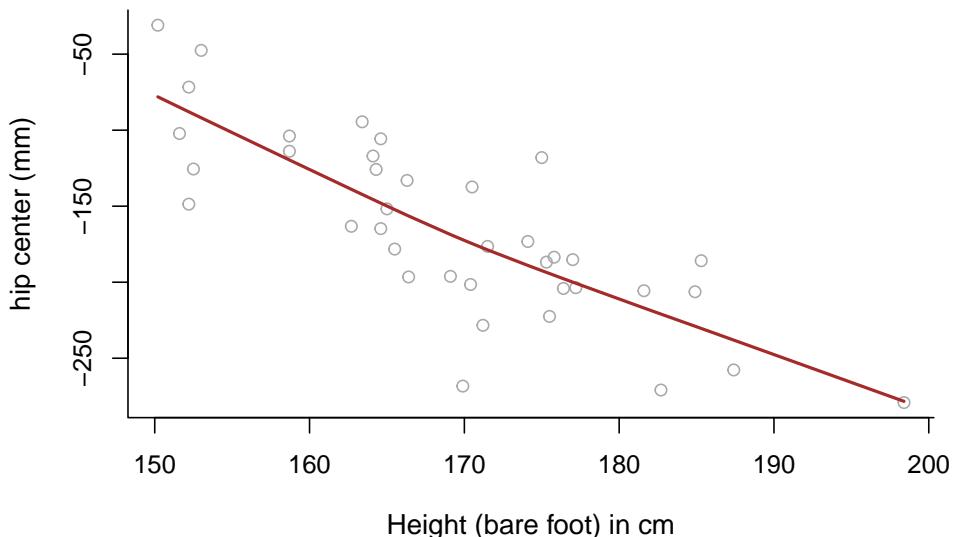
# Plot the fitted linear spline model over the data.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Linear Spline", bty = 'l')
lines(sort.x, pred1$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands1, lwd = 2, col = "red", lty = 3)
```

Linear Spline



```
#(d)
smooth1 = smooth.spline(x, y, df = 3)

plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Smoothing Spline (3df)", bty = 'l')
lines(smooth1, lwd = 2, col = "brown")
```

**Smoothing Spline (3df)**

```
# The fitted model is almost completely linear.
```

## GAMs

In this exercise we consider multiple predictors for modeling the `seatpos` data by GAMs.

### Exercise 4.3

- Fit a GAM for `hipcenter` that consists of three terms:
  - a natural spline with 5 degrees of freedom for `Age`,
  - a smoothing spline with 3 degrees of freedom for `Thigh`, and
  - a simple linear model term for `Ht`.
- Plot the resulting contributions of each term to the GAM, and compare them with plots of `hipcenter` against each of the three variables `Age`, `Thigh` and `Ht`.
- Does the contribution of each term of the GAM make sense in light of these pair-wise plots. Is the GAM fitting the data well?

Click for solution

```
## SOLUTION
```

```
#(a)
# Require library gam.
```

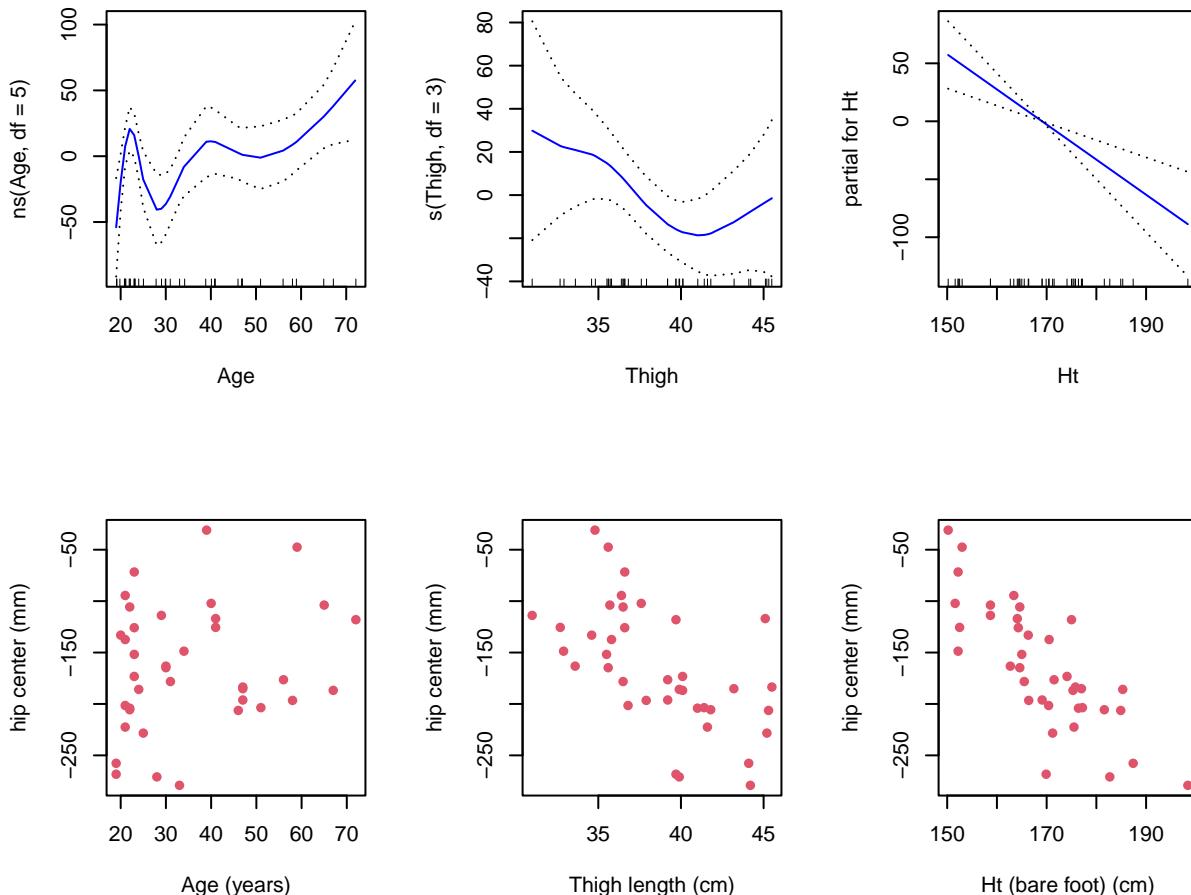
```

library(gam)

# Fit a GAM.
gam = gam( hipcenter ~ ns( Age, df = 5 ) + s( Thigh, df = 3 ) + Ht,
           data = seatpos )

#(b)
# Plot the contributions.
par( mfrow = c(2,3) )
plot( gam, se = TRUE, col = "blue" )
# Compare with the following plots.
plot( seatpos$Age, seatpos$hipcenter, pch = 16, col = 2,
      ylab = y.lab, xlab = "Age (years)" )
plot( seatpos$Thigh, seatpos$hipcenter, pch = 16, col = 2,
      ylab = y.lab, xlab = "Thigh length (cm)" )
plot( seatpos$Ht, seatpos$hipcenter, pch = 16, col = 2,
      ylab = y.lab, xlab = "Ht (bare foot) (cm)" )

```



```

#(c)
# The linear fit of Ht is quite clear. Perhaps the fits for Age and Thigh
# are less clear, although I think we can see that the curves are fitting
# the data somewhat.
# Perhaps they are overfitting, capturing variation across the predictor

```

```
# ranges (in the sample) that isn't really there (in the population).
```

## Boston Data

In this exercise we will investigate the Boston Housing Data.

```
library("MASS")
library(splines)
```

### Exercise 4.4

- Investigate modelling `medv` using regression, natural and smoothing splines, taking `indus` as a single predictor.
- Choose a set of predictor variables to use in order to fit a GAM for `medv`. Feel free to experiment with different modelling options.

Click for solution

```
## SOLUTION
#(a)

y = Boston$medv
x = Boston$indus
y.lab = 'Median Property Value'
x.lab = 'Non-retail business acres per town'

# Use summary on x to find 25th, 50th and 75th percentiles. We will use
# these for the positions of the knots for regression and natural splines.
summary(x)

##      Min. 1st Qu. Median   Mean 3rd Qu.    Max.
##      0.46    5.19   9.69  11.14  18.10  27.74

cuts <- summary(x)[c(2,3,5)]
cuts

## 1st Qu. Median 3rd Qu.
## 5.19    9.69  18.10

# sort x for later.
sort.x = sort(x)

# Fit a cubic spline
spline.bs = lm(y ~ bs(x, knots = cuts))
pred.bs = predict(spline.bs, newdata = list(x = sort.x), se = TRUE)
se.bands.bs = cbind(pred.bs$fit + 2 * pred.bs$se.fit,
                     pred.bs$fit - 2 * pred.bs$se.fit)

# Fit a natural cubic spline.
```

```

spline.ns = lm(y ~ ns(x, knots = cuts))
pred.ns = predict(spline.ns, newdata = list(x = sort.x), se = TRUE)
se.bands.ns = cbind(pred.ns$fit + 2 * pred.ns$se.fit,
                     pred.ns$fit - 2 * pred.ns$se.fit)

# Fit a smoothing spline, with 3 effective degrees of freedom.
spline.smooth = smooth.spline(x, y, df = 3)

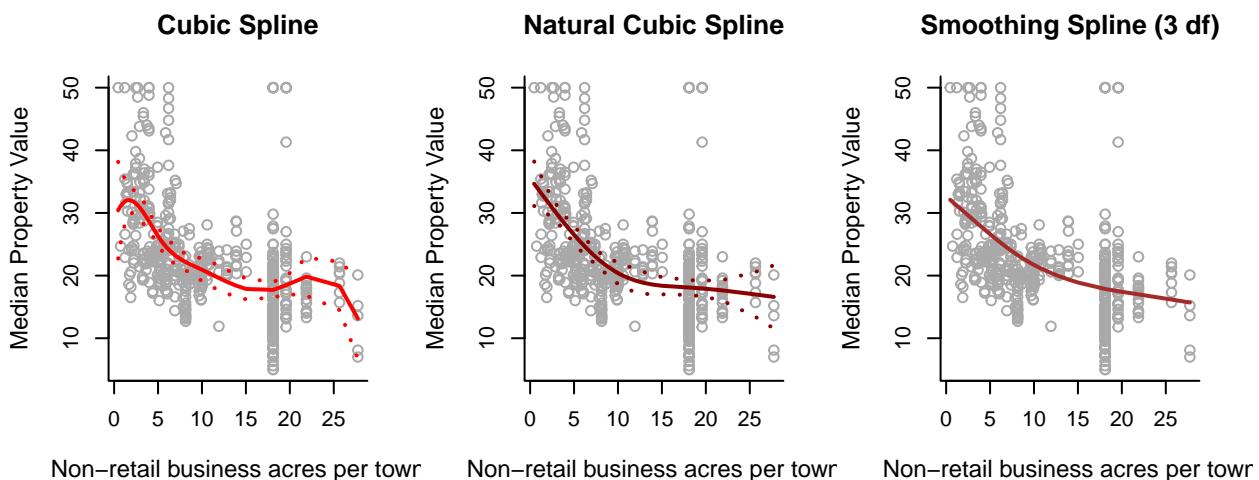
# Split the plotting device into 3.
par(mfrow = c(1,3))

# Plot the cubic spline.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Cubic Spline", bty = 'l')
lines(sort.x, pred.ns$fit, lwd = 2, col = "red")
matlines(sort.x, se.bands.ns, lwd = 2, col = "red", lty = 3)

# Plot the natural cubic spline.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Natural Cubic Spline", bty = 'l')
lines(sort.x, pred.ns$fit, lwd = 2, col = "darkred")
matlines(sort.x, se.bands.ns, lwd = 2, col = "darkred", lty = 3)

# Plot the smoothing spline.
plot(x, y, cex.lab = 1.1, col="darkgrey", xlab = x.lab, ylab = y.lab,
      main = "Smoothing Spline (3 df)", bty = 'l')
lines(spline.smooth, lwd = 2, col = "brown")

```



#(b)

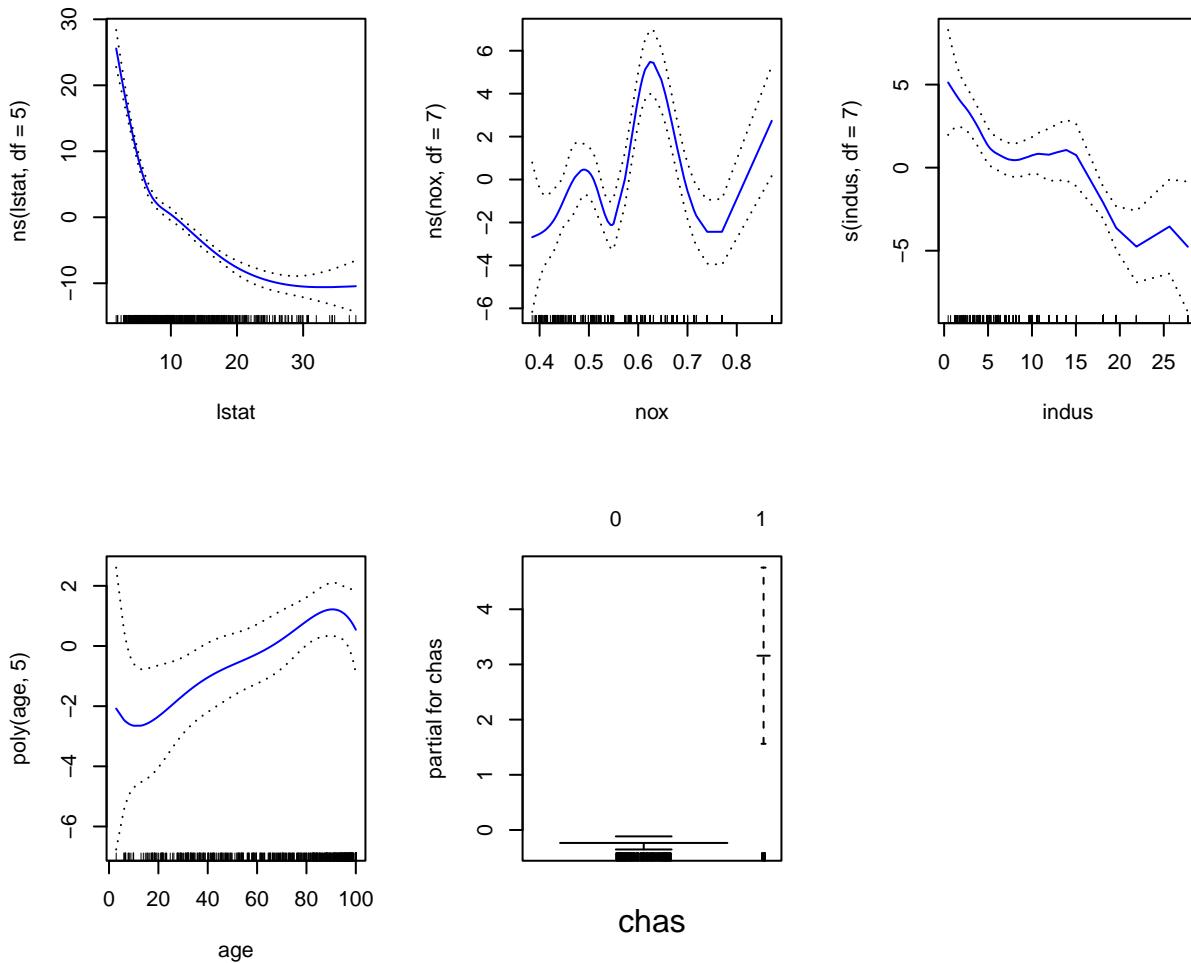
```

# Of course we discussed one possible GAM in lecture practical
# demonstration. We try another one here.

```

```
# Let's set chas to be a factor (you may want to make rad a factor as well,
# if used).
Boston1 = Boston
Boston1$chas = factor(Boston1$chas)

# Let's fit a GAM - can you see how each predictor is contributing to
# modelling the response?
gam1 = gam( medv ~ ns( lstat, df = 5 ) + ns( nox, df = 7 ) +
            s( indus, df = 7 ) + poly( age, 5 ) + chas, data = Boston1 )
par(mfrow = c(2,3))
plot(gam1, se = TRUE, col = "blue")
```



## Logistic Regression

In this exercise we will apply logistic regression to a binary classification problem.

Suppose we are interested in how variables, such as GRE (Graduate Record Exam scores), GPA (grade point average) and prestige of the undergraduate institution, effect admission into graduate school. The response variable, admit/don't admit, is a binary variable.

This `admit` dataset has a binary response (outcome, dependent) variable called `admit`. There

are three predictor variables: gre, gpa and rank. We will treat the variables gre and gpa as continuous. The variable rank takes on the values 1 through 4. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest.

```
admit <- read.csv("https://www.maths.dur.ac.uk/users/hailiang.du/data/admit.csv")
head(admit)
```

```
##   admit gre  gpa rank
## 1     0 380 3.61    3
## 2     1 660 3.67    3
## 3     1 800 4.00    1
## 4     1 640 3.19    4
## 5     0 520 2.93    4
## 6     1 760 3.00    2
```

### Exercise 4.5

- Get basic descriptives for the entire data set by using `summary`. Explore the data graphically by producing a pairs plot of the predictors and, additionally, colouring the points according to whether admit/don't admit.
- Convert `rank` to a factor to indicate that rank should be treated as a categorical variable.
- Fit a logistic regression model in order to predict `admit` using `gre`, `gpa` and `rank`. The `glm()` function can be used to fit many types of generalized linear models , including logistic regression. The syntax of the `glm()` function is similar to that of `lm()`, except that we must pass in the argument `family = binomial` in order to tell R to run a logistic regression rather than some other type of generalized linear model.
- Use the `predict()` function to predict the probability of `admit`, given values of the predictors for the training data. Hint: [The `type = "response"` option tells R to output probabilities of the form  $P(Y=1|X)$ , as opposed to other information such as the logit. If no data set is supplied to the `predict()` function, then the probabilities are computed for the training data that was used to fit the logistic regression model. ]
- Make a prediction as to whether admit or not by converting these predicted probabilities into binary values, 0 or 1, based on whether the predicted probability of admit is greater than or less than 0.5.
- Given these predictions, use the `table()` function to produce a confusion matrix in order to determine how many observations were correctly or incorrectly classified. The confusion matrix is a two by two table with counts of the number of times each combination occurred e.g. predicted admit and the student is admitted, predicted admit and the student is not admitted etc.

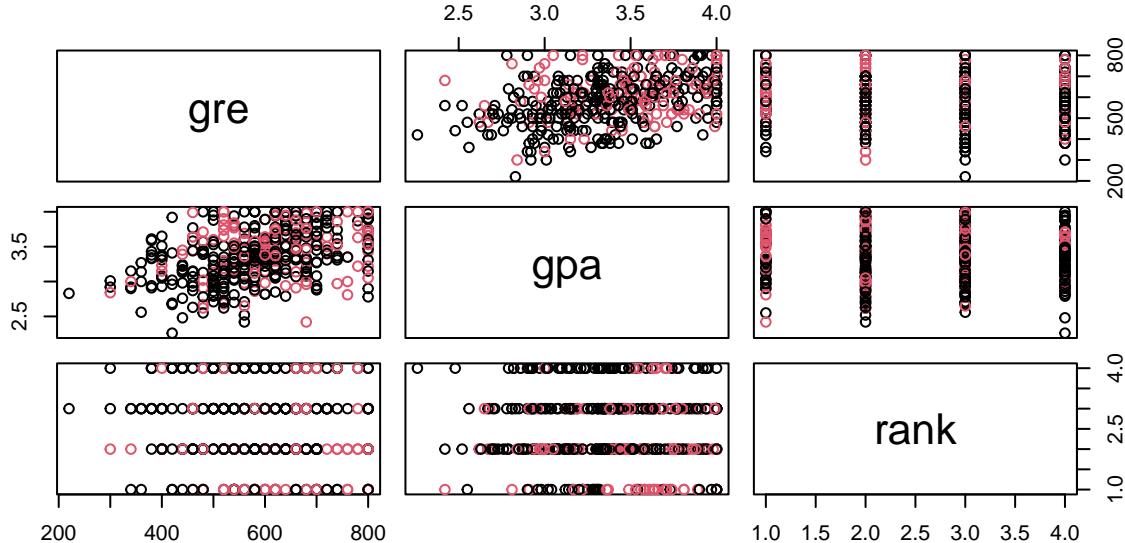
Click for solution

```
## SOLUTION
```

```
#(a)
summary(admit)
```

```
##      admit          gre          gpa         rank
##  Min.   :0.0000   Min.   :220.0   Min.   :2.260   Min.   :1.000
##  1st Qu.:0.0000  1st Qu.:520.0   1st Qu.:3.130   1st Qu.:2.000
##  Median :0.0000  Median :580.0   Median :3.395   Median :2.000
##  Mean   :0.3175  Mean   :587.7   Mean   :3.390   Mean   :2.485
##  3rd Qu.:1.0000  3rd Qu.:660.0   3rd Qu.:3.670   3rd Qu.:3.000
##  Max.   :1.0000  Max.   :800.0   Max.   :4.000   Max.   :4.000

pairs(admit[,2:4], col=admit[,1]+1)
```



*#it appears that higher gre score and gpa score tend to lead to admit.*

```
#(b)
admit$rank <- factor(admit$rank)

#(c)
glm.fit = glm(admit ~ ., data=admit, family="binomial")
summary(glm.fit)
```

```
##
## Call:
## glm(formula = admit ~ ., family = "binomial", data = admit)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.6268  -0.8662  -0.6388   1.1490   2.0790
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.989979   1.139951  -3.500 0.000465 ***
## gre          0.002264   0.001094   2.070 0.038465 *
## gpa          0.804038   0.331819   2.423 0.015388 *
```

```

## rank2      -0.675443  0.316490 -2.134 0.032829 *
## rank3      -1.340204  0.345306 -3.881 0.000104 ***
## rank4      -1.551464  0.417832 -3.713 0.000205 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 499.98 on 399 degrees of freedom
## Residual deviance: 458.52 on 394 degrees of freedom
## AIC: 470.52
##
## Number of Fisher Scoring iterations: 4

#(d)
glm.probs <- predict(glm.fit, type = "response")
glm.probs[1:10]

##          1         2         3         4         5         6         7         8
## 0.1726265 0.2921750 0.7384082 0.1783846 0.1183539 0.3699699 0.4192462 0.2170033
##          9        10
## 0.2007352 0.5178682

#(e)
glm.pred=rep(0, 400)
glm.pred[glm.probs > .5] = 1

#(f)
table(glm.pred, admit$admit)

##
## glm.pred  0   1
##           0 254  97
##           1  19  30
## (254 + 30) / 400

## [1] 0.71
mean(glm.pred == admit$admit)

## [1] 0.71

#The diagonal elements of the confusion matrix indicate correct predictions,
#while the off-diagonals represent incorrect predictions.
#Hence our model correctly predicted 254+30 out of 400 cases.
#The `mean()` function can be used to compute the fraction of the prediction was corre

```



# References



# Bibliography

- S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 7 edition, 2009.
- J. J. Faraway. *Linear Models with R*. CRC press, London, 2009.
- J. Hurwitz and D. Kirsch. *Machine Learning for Dummies*. John Wiley & Sons, New Jersey, 2018.
- G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*. Springer, New York, 2013.
- W.J. Krzanowski. *Principles of Multivariate Analysis: A User's Perspective*. Oxford University Press, Oxford, 2000.
- K.V. Mardia, J.T. Kent, and J.M. Bibby. *Multivariate Analysis*. Academic Press, London, 1979.
- T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- Kevin P. Murphy. *Machine Learning : A Probabilistic Perspective*. MIT Press, Cambridge, 2012.
- Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, Cambridge, 2022.
- Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine, Series 6*, 11(2):559–572, 1901.
- N.A. Weiss. *Introductory Statistics*. Addison-Wesley Pearson Inc., Boston, 2012.