MAS8381: Statistics for Big data

Part 2: Multivariate Data Analysis using R

Prof Darren J Wilkinson
@darrenjw

Component description:

In the 21st Century, statisticians and data analysts typically work with data sets containing a large number of observations and many variables. This part of the course will consider methods for making sense of data of this kind, with an emphasis on practical techniques. Consider, for example, a medical database containing records on a large number of people. Each person has a height, a weight, an age, a blood type, a blood pressure, and a host of other attributes, some quantitative, and others categorical. We will look at graphical and descriptive techniques which help us to visualise a multi-dimensional data set and at inferential methods which help us to answer more specific questions about the population from which the individuals were sampled.

Relevant texts:

T. Hastie, R. Tibshirani, J. Friedman: The Elements of Statistical Learning: Data mining, inference, and prediction, 2nd Edition (Springer-Verlag, 2009).

B. Everitt: An R and S-Plus Companion to Multivariate Analysis (Springer-Verlag, 2005).

I will refer frequently to these texts in the notes, especially the former, which I will cite as **[ESL]**. I will refer to the latter as **[Everitt]**, mainly for R-related information. Note that the PDF of the full text of **[ESL]** is available freely on-line, and that **[Everitt]** should be available electronically to Newcastle University students via the University Library reading list web site.

WWW page:

http://www.staff.ncl.ac.uk/d.j.wilkinson/teaching/mas8381/

Last update:

April 6, 2017

These notes have been re-written for delivery in the Autumn of 2014. I hope they are now finalised. I will only update if I spot errors.

Use the date above to check when this file was generated.

Contents

Chapter 1

Introduction to multivariate data and random quantities

1.1 Introduction

1.1.1 A few quotes...

Google's Chief Economist Hal Varian on Statistics and Data:

> I keep saying the sexy job in the next ten years will be statisticians. People think I'm joking, but who would've guessed that computer engineers would've been the sexy job of the 1990s?

Varian then goes on to say:

> The ability to take data - to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it's going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids. Because now we really do have essentially free and ubiquitous data. So the complimentary scarce factor is the ability to understand that data and extract value from it.

Source: FlowingData.com

The big data revolution's "lovely" and "lousy" jobs:

> The lovely jobs are why we should all enroll our children immediately in statistics courses. Big data can only be unlocked by shamans with tremendous mathematical aptitude and training. McKinsey estimates that by 2018 in the United States alone, there will be a shortfall of between 140,000 and 190,000 graduates with "deep analytical talent". If you are one of them, you will surely have a "lovely" well-paying job.

Source: The Globe and Mail

The Search For Analysts To Make Sense Of 'Big Data' (an article on an NPR programme) begins:

> Businesses keep vast troves of data about things like online shopping behavior, or millions of changes in weather patterns, or trillions of financial transactions — information that goes by the generic name of big data.
>
> Now, more companies are trying to make sense of what the data can tell them about how to do business better. That, in turn, is fueling demand for people who can make sense of the information — mathematicians — and creating something of a recruiting war.

Source: NPR.org

Also see this article in the NYT: For Today's Graduate, Just One Word: Statistics

1.1.2 Data in the internet age

It is clear from the above quotes that technology is currently having a dramatic impact on the way that data is being collected and analysed in the 21st Century. Until recently, the cost of collecting and measuring data for statistical analysis and interpretation meant that most analyses of statistical data concerned small data sets or carefully designed experiments that were relatively straightforward to model. Today, most organisations (commercial businesses and companies, as well as other institutions, such as research institutes and public sector bodies) record and conduct most aspects of their operation electronically, by default. Furthermore, technology for measuring, recording, analysing, and archiving data is rapidly evolving, and this is leading to a so-called "data explosion". Most organisations have vast "data warehouses" of information which contain vital information that could allow the organisation to work more effectively. The business of extracting value from such data goes by various names in different contexts, including "business intelligence", "business analytics", "predictive analytics", "predictive modelling", "informatics", "machine learning", "data science", "data mining", as well as the more conventional "data analysis and modelling". Occasionally, even as "statistics"...

Similarly, many areas of scientific research are currently being similarly transformed. Within physics, the CERN Large Hadron Collider is generating terabytes of data. In biology, new technologies such as high throughput sequencing are resulting in routine generation of massive data sets. Here a typical (processed) datafile will contain many millions of "reads", each of which will consist of around 100 base pairs (letters), and a typical experiment will generate several such data files. Such experiments are being conducted every day in research institutions all over the world, and "bioinformaticians" skilled in statistics and data analysis are in short supply.

Analysing and interpreting large and complex data sets is a significant challenge requiring many skills, including those of statistical data analysis and modelling. It would be unrealistic to attempt in a single module to provide all of the knowledge and skills necessary to become a real "data scientist". Here we will concentrate on some of the key statistical concepts and techniques necessary for modern data analysis. Typical characteristics of modern data analysis include working with data sets that are large, multivariate, and highly structured, but with a non-trivial structure inconsistent with classical experimental design ideas.

1.1.3 Module outline

Chapter 1: Introduction to multivariate data and random quantities
This first chapter will lay the foundation for the rest of the course, providing an introduction to multivariate data, mainly using **R** "data frames" as a canonical example, and introducing some basic methods for visualisation and summarisation, illustrated with plenty of worked examples.

Chapter 2: PCA and matrix factorisations
In this chapter we will examine how techniques from linear algebra can be used to transform data in a way that better reveals its underlying structure, and the role that matrix factorisations play in this. Principal components analysis (PCA) will be used to illustrate the ideas.

Chapter 3: Inference, the MVN distribution, and multivariate regression
In this chapter we will introduce the simplest and most useful multivariate probability distribution: the multivariate normal (MVN) distribution. We will construct it, examine some of its key properties, and look at some applications. We will also briefly examine linear regression for multivariate outputs.

Chapter 4: Cluster analysis and unsupervised learning
In this chapter we will look at how it is possible to uncover latent structure in multivariate data without any "training data". Clustering is the canonical example of what data miners refer to as "unsupervised learning".

Chapter 5: Discrimination and classification
In this chapter we will briefly examine the related problems of discrimination and classification.

Chapter 6: Graphical modelling
In this chapter we will look how notions of conditional independence can be used to understand multivariate relationships, and how these naturally lead to graphical models of variable dependencies.

Chapter 7: Variable selection and multiple testing
In this chapter we will look briefly at some of the issues that arise when working with data sets that are large or high dimensional.

Chapter 8: Bayesian inference
In this final chapter we will discuss Bayesian inference for linear Gaussian models, emphasising efficient and numerically stable computation, using techniques from numerical linear algebra discussed earlier in the course.

1.2 Multivariate data and basic visualisation

1.2.1 Tables

Many interesting data sets start life as a (view of a) table from a relational database. Not at all coincidentally, database tables have a structure very similar to an **R** "data frame" (**R**

data frames are modelled on relational database tables). Therefore for this course we will restrict our attention to the analysis and interpretation of data sets contained in **R** data frames. Conceptually a data frame consists as a collection of n "rows", each of which is a p-tuple, $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ from some set of the form $S_1 \times S_2 \times \cdots \times S_p$, where each S_j is typically a finite discrete set (representing a *factor*) or $S_j \subseteq \mathbb{R}$. In the important special case where we have $S_j \subseteq \mathbb{R}$, $j = 1, 2, \ldots, p$, we may embed each S_j in $\mathbb{R}$ and regard each p-tuple x_i as a vector in (some subset of) $\mathbb{R}^p$. If we now imagine our collection of p-tuples stacked one on top of the other, we have a table with n rows and p columns, where everything in the jth column is an element of S_j. In fact, **R** internally stores data frames as a list of columns of the same length, where everything in the column is a vector of a particular type.* It is important to make a clear distinction between the rows and the columns. The rows, $i = 1, \ldots, n$ typically represent *cases*, or *individuals*, or *items*, or *objects*, or *replicates*, or *observations*, or *realisations* (from a probability model). The columns $j = 1, \ldots, p$ represent *attributes*, or *variables*, or *types*, or *factors* associated with a particular case, and each contains data that are *necessarily* of the same type.

Example: Insect sprays

To make things concrete, let's look at a few simple data frames bundled as standard data frames in **R**. We can get a list of **R** data sets using the **R** command `data()`. The data set `InsectSprays` is a data frame. We can get information about this object with `?InsectSprays`, and we can look at the first few rows with

```
> head(InsectSprays)
  count spray
1    10     A
2     7     A
3    20     A
4    14     A
5    14     A
6    12     A
>
```

The data represents counts of numbers of insects in a given agricultural area for different kinds of insect sprays. We can get a simple plot of the data with `boxplot(count~spray,data=InsectSprays)` (which shows that sprays C, D and E seem to be most effective), and understand the content with

```
> class(InsectSprays)
[1] "data.frame"
> dim(InsectSprays)
[1] 72  2
> levels(InsectSprays$spray)
[1] "A" "B" "C" "D" "E" "F"
> table(InsectSprays$count)

 0  1  2  3  4  5  6  7  9 10 11 12 13 14 15 16 17 19 20 21 22 23 24 26
```

*This internal storage is different to the way that relational databases typically store tables — database tables are typically stored internally as collections of rows. However, since **R** provides mechanisms for accessing data by rows as well as columns, this difference will not trouble us.

```
 2  6  4  8  4  7  3  3  1  3  3  2  4  4  2  2  4  1  2  2  1  1  1  2
> str(InsectSprays)
'data.frame':   72 obs. of  2 variables:
 $ count: num  10 7 20 14 14 12 10 23 17 20 ...
 $ spray: Factor w/ 6 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1
    ...
>
```

Consequently the data frame has $n = 72$ and $p = 2$. Formally, we can regard each row as being a 2-tuple from $\mathbb{Z} \times \{A, B, C, D, E, F\}$. Note that although we can easily embed $\mathbb{Z}$ in $\mathbb{R}$, there is no natural embedding of the unordered factor $\{A, B, C, D, E, F\}$ in $\mathbb{R}$, and so it will probably not be sensible to attempt to regard the rows as vectors in $\mathbb{R}^2$.

1.2.2 Working with data frames

The previous example **R** session shows some useful methods for finding out some basic information about a data frame. However, we very often wish to extract information from data frames for subsequent analysis. We will look at methods for *transforming* data later. Here we will just look at methods for extracting and subsetting rows and columns from a data frame ready for further analysis. First, we can get the names of the variables using `names()`, and the names of the cases with `rownames()`. We can illustrate with the `InsectSprays` data frame that we have already introduced:

```
> names(InsectSprays)
[1] "count" "spray"
> rownames(InsectSprays)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13"
[14] "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26"
[27] "27" "28" "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39"
[40] "40" "41" "42" "43" "44" "45" "46" "47" "48" "49" "50" "51" "52"
[53] "53" "54" "55" "56" "57" "58" "59" "60" "61" "62" "63" "64" "65"
[66] "66" "67" "68" "69" "70" "71" "72"
```

Here the row names are not interesting, and just correspond to row numbers, but in some cases the row names are useful. We can access individual elements of the data frame by row and column number, so that `InsectSprays[3,1]` returns 20. Alternatively, we can access using names, so that `InsectSprays[3,"count"]` is equivalent.

We can *column slice* a data frame, to get a data frame with fewer columns, using single bracket notation, so that `InsectSprays[1]` and `InsectSprays["count"]` both return a data frame containing a single column.

A vector representing a single column of a data frame can be obtained in a variety of ways — the following are all equivalent:

```
InsectSprays[[1]]
InsectSprays[["count"]]
InsectSprays$count
InsectSprays[,1]
InsectSprays[,"count"]
```

Note the subtle but important distinction between a vector and a data frame containing a single column.

We can also access and subset data frames by rows. A row can be accessed by name or number, so that here `InsectSprays[3,]` and `InsectSprays["3",]` both return a data frame consisting of just one row, corresponding to the third row of the original `InsectSprays` data frame. Note that there is no *direct* way to return a vector corresponding to a row, since the elements of the row will typically not all be of the same type. We will look at some of the things that can be done with a data frame where all of the values are numeric in the next example. We can create a data frame containing a subset of rows from the data frame by using a vector of row numbers or names. eg.

```
> InsectSprays[3:5,]
  count spray
3    20     A
4    14     A
5    14     A
>
```

We can also subset the rows using a boolean vector of length n which has `TRUE` elements corresponding the the required rows. eg. `InsectSprays[InsectSprays$spray=="B",]` will extract the rows where the `spray` factor is `B`. If the vector of counts is required, this can be extracted from the resulting data frame in the usual way, but this could also be obtained more directly using

```
> InsectSprays$count[InsectSprays$spray=="B"]
 [1] 11 17 21 11 16 14 17 17 19 21  7 13
>
```

Example: Motor Trend Cars

We will illustrate a few more important **R** techniques for working with data frames using another simple built-in data set, `mtcars`, before moving on to look at some more interesting examples. Again, the `str()` command is very useful.

```
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
>
```

So we see that we have $n = 32$ and $p = 11$, and here also that all 11 variables are `numeric`, so we *could* regard the rows as points in $\mathbb{R}^{11}$ if we felt that was sensible. However, closer inspection reveals that some columns are counts, and some are numeric encodings of binary variables, so direct embedding into $\mathbb{R}^{11}$ may not be such a great idea. This data frame has sensible names for both the rows and the columns

```
> names(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "
    gear"
[11] "carb"
> rownames(mtcars)
 [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
 [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
 [7] "Duster 360"          "Merc 240D"           "Merc 230"
[10] "Merc 280"            "Merc 280C"           "Merc 450SE"
[13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
[16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
[22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
[28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"
>
```

Let us suppose that we are only interested in the first 7 variables. A new data frame containing just those columns can be created with

```
myCars=mtcars[c("mpg","cyl","disp","hp","drat","wt","qsec")]
```

or more consisely with

```
myCars=mtcars[1:7]
```

A particular row can be extracted with

```
> myCars["Fiat 128",]
          mpg cyl disp hp drat  wt  qsec
Fiat 128 32.4   4 78.7 66 4.08 2.2 19.47
>
```

And a subset of rows (say, economical cars) can be extracted with

```
> myCars[myCars$mpg>30,]
                mpg cyl disp  hp drat    wt  qsec
Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47
Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52
Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90
Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90
>
```

There are obviously many simple plots one can do to investigate the relationship between pairs of variables, so `boxplot(mpg~cyl,data=myCars)` shows how fuel economy varies with the number of cylinders, and `plot(myCars$disp,myCars$mpg)` shows fuel economy as a function of engine size. Plots such as these are useful, but multivariate analysis is concerned with developing more sophisticated methods for understanding the relationships between larger groups of variables.

Before moving on to look at more interesting examples, it is worth noting that in cases such as this, where all of the variables are numeric, we can use the function `as.matrix()` to coerce the data frame into an $n \times p$ real-valued matrix, which can be used as a regular numeric matrix in other R functions. *This is very useful!*

```
> class(myCars)
[1] "data.frame"
> carsMat=as.matrix(myCars)
> class(carsMat)
[1] "matrix"
> dim(carsMat)
[1] 32  7
>
```

Example: Galaxy data

Several of the example data sets that we will consider in this course are associated with the book **[ESL]**. There is a CRAN **R** package containing data sets from the book called `ElemStatLearn`. Typing `library(ElemStatLearn)` or `require(ElemStatLearn)` will load the package provided that it is installed. If these commands give an error, the most likely explanation is that the package is not installed. It should be possible to install on a machine with an internet connection using `install.packages("ElemStatLearn")`. A list of data sets in the package can be obtained with

```
require(ElemStatLearn)
data(package="ElemStatLearn")
help(package="ElemStatLearn")
```

We will look first at the `galaxy` data frame (use `?galaxy` for background information about the data).

```
> str(galaxy)
'data.frame':   323 obs. of  5 variables:
 $ east.west      : num  8.46 7.96 7.47 6.97 6.47 ...
 $ north.south    : num  -38.2 -35.9 -33.7 -31.4 -29.2 ...
 $ angle          : num  102 102 102 102 102 ...
 $ radial.position: num  39.1 36.8 34.5 32.2 29.9 ...
 $ velocity       : int  1769 1749 1749 1758 1750 1745 1750 1753 1734
    1710 ...
>
```

Clearly here we have $n = 323$ and $p = 5$, and it appears at first as though all of the variables are numerical, making it sensible to regard the rows as vectors in $\mathbb{R}^5$. For a numerical data frame such as this, we can produce a set of all pairs of scatterplots of one variable against another using the `pairs()` command. So `pairs(galaxy)` gives such a *scatterplot matrix*, as shown in Figure 1.1. In fact, the `plot()` command defaults to `pairs()` for a data frame, so `plot(galaxy)` works just as well, but it is probably better to be explicit about exactly what kind of plot is required.

Scatterplot matrices such as this are one of the simplest yet most useful methods for beginning to understand a multivariate dataset, provided that the number of variables is not too large. All pairwise plots of one variable against another are shown in the matrix. The row determines the variable plotted on the vertical (y) axis, and the column determines the variable plotted on the horizontal (x) axis. For example, the plot in the last column of the first row shows a plot of `east.west` against `velocity`. Close inspection of these plots reveals that `angle` in fact looks as though it takes on only a small number of discrete values. We can confirm this with

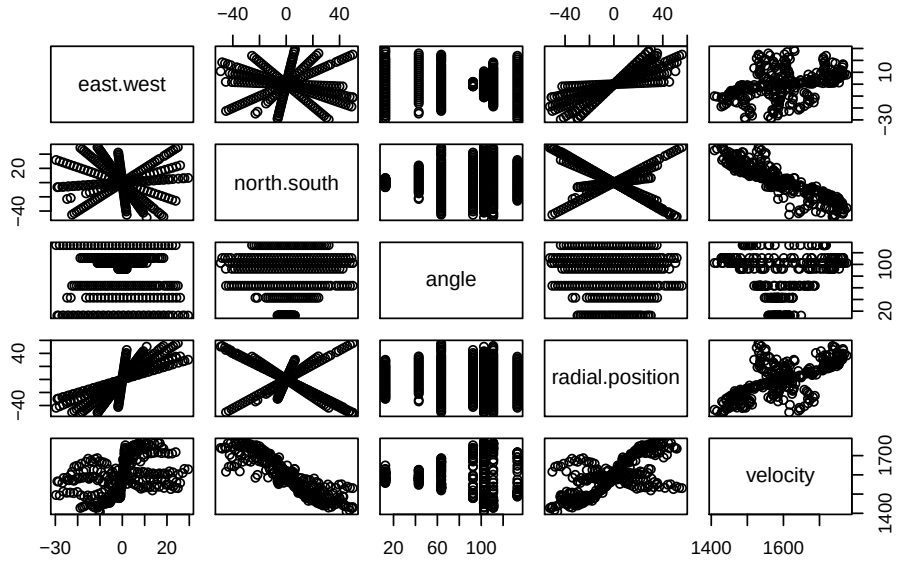


Figure 1.1: Scatterplot matrix for the Galaxy data

```
> table(galaxy$angle)

 12.5    43  63.5  92.5 102.5   111   133
   51    28    46    38    80    45    35
>
```

which shows that `angle` does indeed take on just 7 distinct values, and hence perhaps has more in common with an ordered categorical variable than a real-valued variable. We can use `angle` as a factor to colour the scatterplot as follows

```
pairs(galaxy,col=galaxy$angle)
```

to obtain the plot shown in Figure 1.2.

This new plot indicates that there appear to be rather simple relationships between the variables, but that those relationships change according to the level of the `angle` "factor". We can focus in on one particular value of angle in the following way

```
pairs(galaxy[galaxy$angle==102.5,-3])
```

Note the use of `-3` to drop the third (`angle`) column from the data frame. The new plot is shown in Figure 1.3.

This plot reveals that (for a given fixed value of `angle`) there is a simple deterministic linear relationship between the first three variables (and so these could be reduced to just one variable), and that there is a largely monotonic "S"-shaped relationship between `velocity` and (say) `radial.position`. So with just a few simple scatterplot matrices and some simple **R** commands we have reduced this multivariate data frame with apparently 5 quantitative variables to just 2 quantitative variables plus an ordered categorical factor. It is not quite as simple as this, as the relationships between the three positional variables varies with the level of `angle`, but we have nevertheless found a lot of simple structure within the data set with some very basic investigation.

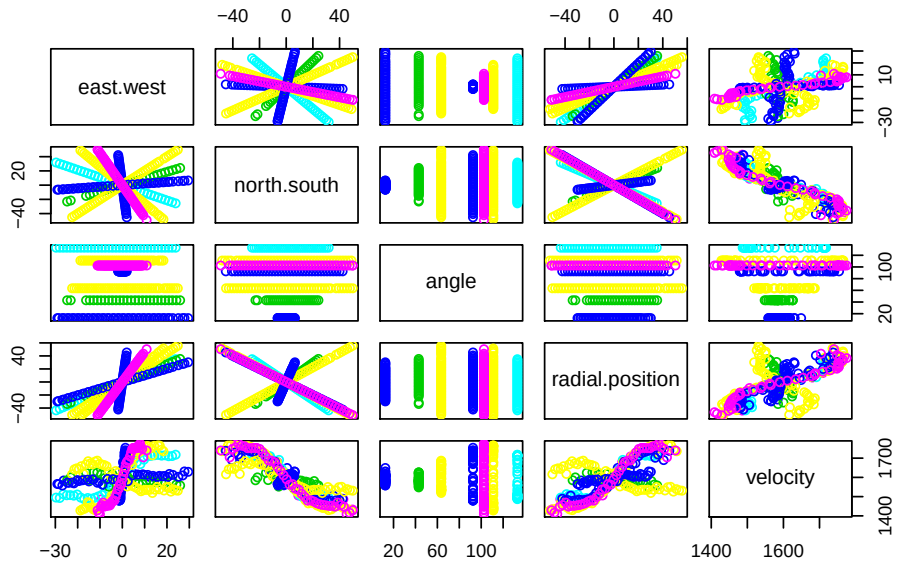


Figure 1.2: Scatterplot matrix for the Galaxy data, coloured according to the levels of the `angle` variable

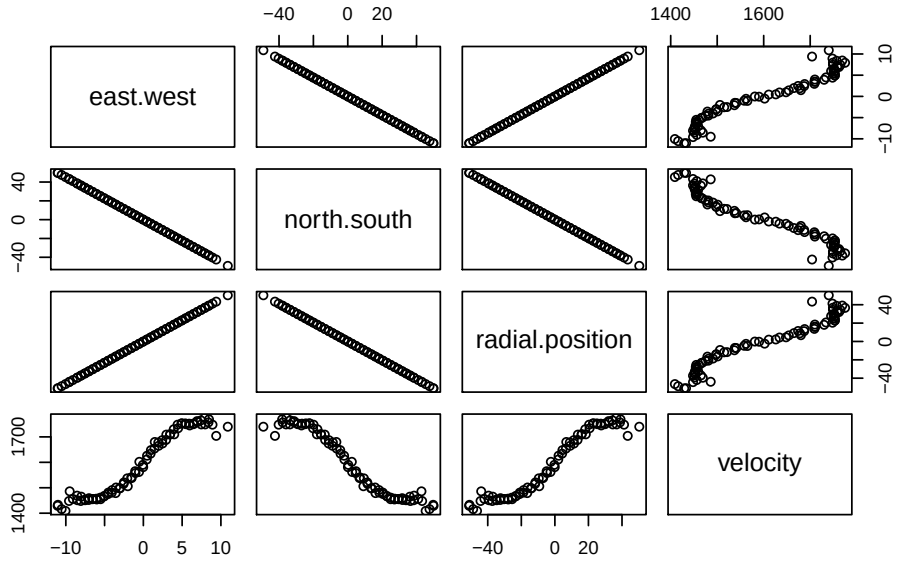


Figure 1.3: Scatterplot matrix for the subset of the Galaxy data corresponding to an `angle` variable of 102.5

Example: Microarray data

The dataset `nci` represents measurements of the expression levels of 6,830 genes in 64 different tissue samples taken from cancerous tumors. The first thing we need to think about here is what the rows and columns should be. Which is n and which is p? The **R** command `class(nci)` reveals that this data set is actually stored as a matrix and not a data frame, so that is no help. The command `dim(nci)` indicates that the matrix is stored with 6,830 rows and 64 columns, but matrices can be easily transposed using the `t()` command, so that doesn't really tell us anything other than how the data happens to be stored. This problem can arise whenever a multivariate data set consists of measurements of exactly the same type in all rows and columns. Here the measurements are expression levels (basically, a measure of the *amount* of a particular kind of mRNA in a sample). Since all measurements have essentially the same units, and can be stored in a matrix, it is technically arbitrary which should be considered rows and columns. However, from the viewpoint of multivariate data analysis, it matters a great deal.

Here, fundamentally, the observations are 64 different tissue samples, and the variables are the 6,830 genes, so $n = 64$ and $p = 6,830$, so in some sense the `nci` matrix should be transposed. However, it is common practice to store microarray data this way, and sometimes for certain analyses people really do take the genes to be the observations and the samples to be the variables. Generally with multivariate analysis we wish to use observations to learn about the relationship between the variables. This data set is no exception, in that we wish to use the samples in order to learn something about the relationship between the genes. Sometimes people work with the transposed data in order to use the genes in order to learn something about the relationship between the samples. However, at a fundamental level, the measurements are of the amount of a particular gene, in units specific to that gene, and in (essentially) identical units for each sample. So it is really the genes that represent the variables, as columns of the same type, in some sense...

There is actually something very special about this data in that $p > n$, and in fact, $p >> n$ (p is *much* bigger than n). Multivariate data with this property is often referred to as "wide data" (as opposed to the more normal "tall data"), for obvious reasons. This is quite different to the other examples we have looked at, and rather atypical of classical multivariate data. Indeed, many of the classical statistical techniques we shall examine in this course actually *require* $n > p$, and ideally need $n >> p$ in order to work well. Data with $p > n$ will cause many classical algorithms to fail (and this is one reason why people sometimes work with the transposed data). We will revisit this issue at appropriate points in the course, and look at some simple techniques for analysing wide data near the end of the course.

It is obviously impractical to produce a $6,380 \times 6,380$ scatterplot matrix for this data! In fact, even if we work with the transposed data, a 64×64 scatterplot matrix is also unlikely to be helpful. So we must use other strategies for visualisation here. The standard way to visualise high-dimensional real-valued matrices is as an *image*, or *heat-map*. We can start with a very simple image plot, using

```
image(nci,axes=FALSE,xlab="genes",ylab="samples")
```

giving rise to the plot in Figure 1.4.

Each individual rectangular area, or *pixel*, of the image represents a single value from the input matrix. By default the origin is at the bottom left, so the matrix is transposed

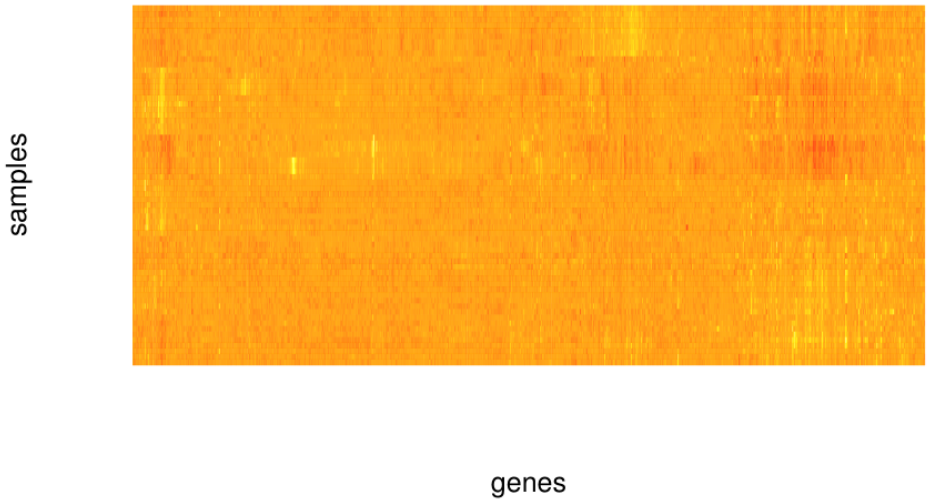


Figure 1.4: Image plot of the `nci` cancer tumor microarray data, using the default colour scheme

(or rather, rotated anti-clockwise). Using the default "heat" colour scheme, low values are represented as "cold" red colours, and "hot" bright yellow/white colours represent the high values. Although many people find this colour scheme intuitive, it is not without problems, and in particular, can be especially problematic for colour-blind individuals. We can produce a simple greyscale image using

```
image(nci,axes=FALSE,xlab="genes",ylab="samples",col=grey((0:31)/31))
```

which has the lightest colours for the highest values, or using

```
image(nci,axes=FALSE,xlab="genes",ylab="samples",col=grey((31:0)/31))
```

to make the highest values coloured dark. Another popular colouring scheme is provided by `cm.colors()`, which uses cyan for low values, magenta for high values, and middle values white. It is good for picking out extreme values, and can be used as

```
image(nci,axes=FALSE,xlab="genes",ylab="samples",col=cm.colors(32))
```

The `image()` function is a fairly low-level function for creating images, which makes it very flexible, but relatively difficult to produce attractive plots. For imaging multivariate data there is a higher-level function called `heatmap()` which produces attractive plots very simply. A basic plot can be obtained with

```
heatmap(nci,Rowv=NA,Colv=NA,labRow=NA,col=grey((31:0)/31))
```

leading to the plot shown in Figure 1.5.

Note that this function keeps the natural orientation of the supplied matrix, and by default will label rows and columns with the row and column names associated with the matrix. Note that the options `Rowv=NA,Colv=NA` are critical, as they disable the default behaviour of the function to use *clustering* to reorder the rows and columns to reveal interesting structure present in the data. This is a computationally intensive operation, so we don't want to attempt this on a full microarray matrix at this stage.

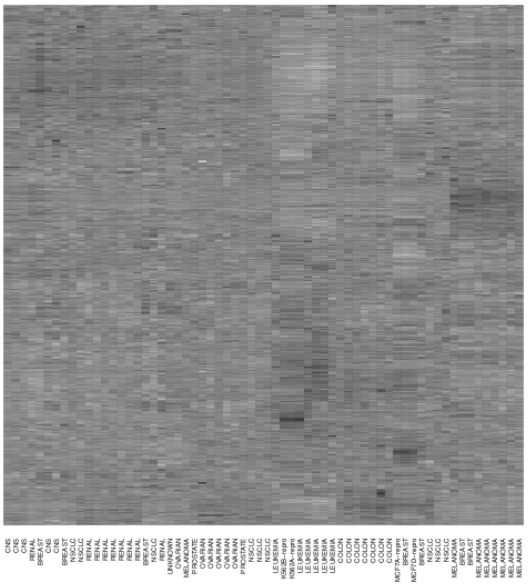


Figure 1.5: Heatmap of the `nci` cancer tumor microarray data, using a greyscale colour scheme with darker shades for higher values

Example: Handwritten digit images

The next example we consider is a dataset representing images of handwritten digits. The underlying motivation is automated recognition of US ZIP codes written on envelopes, for automated routing of US mail. A large number of manually classified examples are in the data set `zip.train`. A sample of images is shown in Figure 1.6.

The dataset is stored as a $7,291 \times 257$ matrix. Each row represents a single classified image, so that $n = 7,291$ and $p = 257$. Each image is actually a greyscale image of 16 by 16 pixels. These 256 pixels can be "unwrapped" into a 256-dimensional vector, which can be regarded as a vector in $\mathbb{R}^{256}$. The first number in each row is the manually classified digit for the image. Therefore each row is a 257-tuple from $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \times \mathbb{R}^{256}$. Although it is tempting to regard the digit as an integer or ordered categorical factor, it should really be considered an unordered factor, as we do not expect the images to change gradually as the digit value increases. The `ElemStatLearn` package includes a convenience function, `zip2image()` for converting a row back into a 16×16 matrix oriented correctly for passing into the **`image()`** function. For example, the image corresponding to the fifth row can be displayed as simply as

```
image(zip2image(zip.train,5))
```

This can be tidied up and imaged in greyscale using

```
image(zip2image(zip.train,5), col=gray(15:0/15), zlim=c(0,1), xlab="",
   ylab="" )
```

giving the image shown in Figure 1.7.

For exploratory analysis of the image data, we can easily strip off the first column corresponding to the classification using

```
myDigits=zip.train[,-1]
```

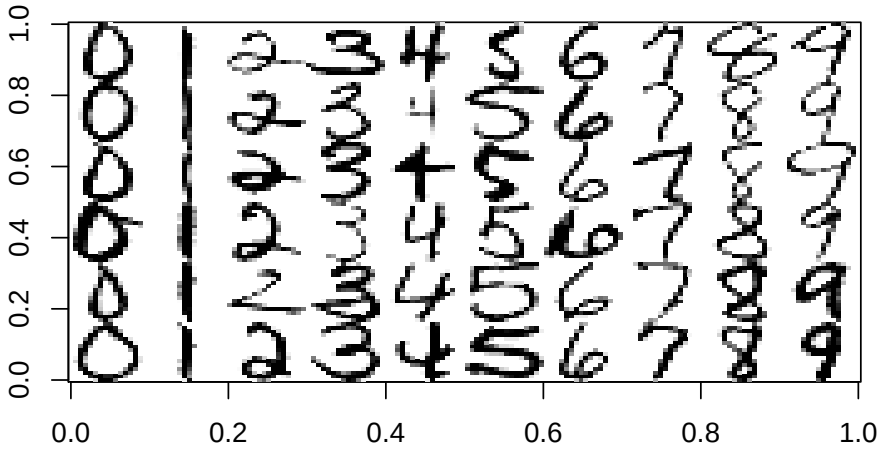


Figure 1.6: A sample of images from the `zip.train` dataset, generated with the command `example(zip.train)`

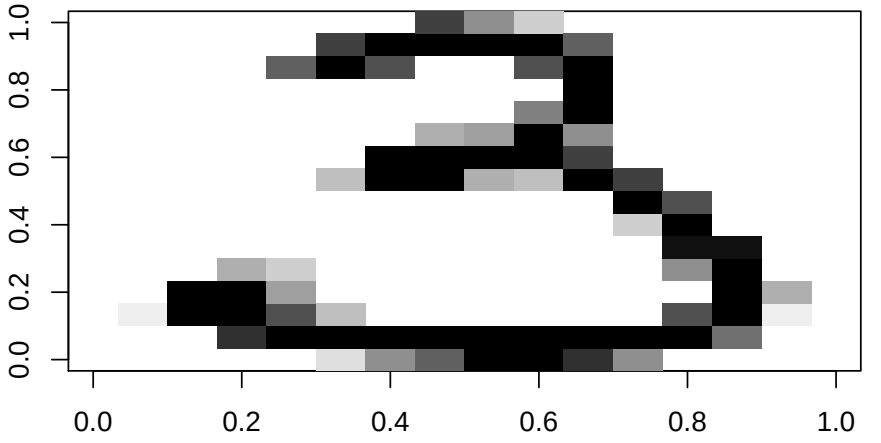


Figure 1.7: Image corresponding to the fifth row of the `zip.train` dataset — the digit shown is a "3"

to give a multivariate dataset with $n = 7,291$ and $p = 256$, where each row represents a vector in $\mathbb{R}^{256}$. It is perhaps concerning that representing the images as a vector in $\mathbb{R}^{256}$ loses some important information about the structure of the data — namely that it is actually a 16×16 matrix. So, for example, we know that the 2nd and 18th elements of the 256-dimensional vector actually correspond to adjacent pixels, and hence are likely to be highly correlated. This is clearly valuable information that we aren't explicitly including into our framework. The idea behind "data mining" is that given a sufficiently large number of example images, it should be possible to "learn" that the 2nd and 18th elements are highly correlated without explicitly building this in to our modelling framework in the first place. We will look at how we can use data on observations to learn about the relationship between variables once we have the appropriate mathematical framework set up.

1.3 Representing and summarising multivariate data

Clearly an important special case of (multivariate) data frames arises when the p variables are all real-valued and so the data can be regarded as an $n \times p$ real-valued matrix. We saw in the previous example that even when the original data is not in exactly this format, it is often possible to extract a subset of variables (or otherwise transform the original data) to give a new dataset which does have this form. This is the classical form of a multivariate dataset, and many classical modelling and analysis techniques of multivariate statistics are tailored specifically to data of this simple form. Classically, as we have seen, the rows of the matrix correspond to observations, and the columns correspond to variables. However, it is often convenient to regard the observations as column vectors, and so care must be taken when working with multivariate data mathematically to transpose rows from a data matrix in order to get a column vector representing an observation.

Definition 1 *The measurement of the jth variable on the ith observation is denoted x_{ij}, and is stored in the ith row and jth column of an $n \times p$ matrix denoted by* X*, known as the* data matrix*.*

However, we use $\boldsymbol{x}_i$ to denote a *column* vector corresponding to the ith observation, and hence the ith *row* of X. That is

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\mathsf{T} = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}.$$

We denote the column vector representing the jth variable by $\boldsymbol{x}^{(j)}$, which we can obviously define directly as

$$\boldsymbol{x}^{(j)} = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix}.$$

We can begin to think about summarising multivariate data by applying univariate summaries that we already know about to the individual variables.

1.3.1 The sample mean

As a concrete example, it is natural to be interested in the sample mean of each variable. The sample mean of $\boldsymbol{x}^{(j)}$ is denoted $\bar{x}_j$, and is clearly given by

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}.$$

We can compute $\bar{x}_j$ for $j = 1, 2, \ldots, p$, and then collect these sample means together into a p-vector that we denote $\bar{\boldsymbol{x}}$, given by

$$\bar{\boldsymbol{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_p)^\mathsf{T}.$$

Note, however, that we could have defined $\bar{\boldsymbol{x}}$ more directly as follows.

Definition 2 *The sample mean of the $n \times p$ data matrix,* X*, is the p-vector given by the sample mean of the* observation vectors*,*

$$\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i.$$

A moments thought reveals that this is equivalent to our previous construction. The vector version is more useful however, as we can use it to build an even more convenient expression directly in terms of the data matrix, X. For this we need notation for a vector of ones. For an n-vector of ones, we use the notation $\mathbf{1}_n$, so that

$$\mathbf{1}_n \equiv (1, 1, \ldots, 1)^\mathsf{T}.$$

We will sometimes drop the subscript if the dimension is clear from the context. Note that $\mathbf{1}_n{}^\mathsf{T}\mathbf{1}_n = n$ (an inner product), and that $\mathbf{1}_n\mathbf{1}_p{}^\mathsf{T}$ is an $n \times p$ matrix of ones (an outer product), sometimes denoted $\mathsf{J}_{n\times p}$. Pre- or post-multiplying a matrix by a row or column vector of ones has the effect of summing over rows or columns. In particular, we can now write the sample mean of observation vectors as follows.

Proposition 1 *The sample mean of a data matrix can be computed as*

$$\bar{\boldsymbol{x}} = \frac{1}{n}\mathsf{X}^\mathsf{T}\mathbf{1}_n.$$

Proof

$$\begin{aligned}
\frac{1}{n}\mathsf{X}^\mathsf{T}\mathbf{1}_n &= \frac{1}{n}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)\begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix} \\
&= \frac{1}{n}\left[\boldsymbol{x}_1 \times 1 + \boldsymbol{x}_2 \times 1 + \cdots + \boldsymbol{x}_n \times 1\right] \\
&= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = \bar{\boldsymbol{x}}
\end{aligned}$$

□

Example: Galaxy data

The R command `summary()` applies some univariate summary statistics to each variable in a data frame separately, and this can provide some useful information about the individual variables that make up the data frame

```
> summary(galaxy)
   east.west          north.south          angle         radial.position
 Min.   :-29.66693   Min.   :-49.108   Min.   : 12.50   Min.   :-52.4000
 1st Qu.: -7.91687   1st Qu.:-13.554   1st Qu.: 63.50   1st Qu.:-21.3500
 Median : -0.06493   Median :  0.671   Median : 92.50   Median : -0.8000
 Mean   : -0.33237   Mean   :  1.521   Mean   : 80.89   Mean   : -0.8427
 3rd Qu.:  6.95053   3rd Qu.: 18.014   3rd Qu.:102.50   3rd Qu.: 19.6500
 Max.   : 29.48414   Max.   : 49.889   Max.   :133.00   Max.   : 55.7000
    velocity
 Min.   :1409
 1st Qu.:1523
 Median :1586
 Mean   :1594
 3rd Qu.:1669
 Max.   :1775
>
```

The `apply()` command can also be used to apply arbitrary functions to the rows or columns of a matrix. Here we can obtain the mean vector using

```
> apply(galaxy,2,mean)
      east.west      north.south            angle radial.position         velocity
     -0.3323685        1.5210889       80.8900929       -0.8427245     1593.6253870
>
```

The `2` is used to indicate that we wish to apply the function to each column in turn. If we had instead used `1`, the mean of each *row* of the matrix would have been computed (which would have been much less interesting). We can also use our matrix expression to directly compute the mean from the data matrix

```
> as.vector(t(galaxy) %*% rep(1,323)/323)
[1]   -0.3323685    1.5210889   80.8900929   -0.8427245 1593.6253870
>
```

where `%*%` is the matrix multiplication operator in R. Since R helpfully transposes vectors as required according to the context, we can actually compute this more neatly using

```
> rep(1,nrow(galaxy))%*%as.matrix(galaxy)/nrow(galaxy)
      east.west north.south    angle radial.position velocity
[1,] -0.3323685    1.521089 80.89009      -0.8427245 1593.625
>
```

It is typically much faster to use matrix operations than the `apply()` command. Note however, that R includes a convenience function, `colMeans()` for computing the sample mean of a data frame:

```
> colMeans(galaxy)
      east.west     north.south            angle radial.position
     -0.3323685        1.5210889       80.8900929       -0.8427245
       velocity
   1593.6253870
```

and so it will usually be most convenient to use that.

Example

Before moving on, it is worth working through a very small, simple problem by hand, in order to make sure that the ideas are all clear. Suppose that we have measured the height (in metres) and weight (in kilograms) of 5 students. We have $n = 5$ and $p = 2$, and the data matrix is as follows

$$\mathsf{X} = \begin{pmatrix} 1.67 & 65.0 \\ 1.78 & 85.0 \\ 1.60 & 54.5 \\ 1.83 & 72.0 \\ 1.80 & 94.5 \end{pmatrix}$$

We can calculate the mean vector, $\bar{\boldsymbol{x}}$ for this data matrix in three different ways. First start by calculating the column means. For $\bar{x}_1$ we have

$$\begin{aligned} \bar{x}_1 &= \frac{1}{n}\sum_{i=1}^{n} x_{i1} = \frac{1}{5}(1.67 + 1.78 + 1.60 + 1.83 + 1.80) \\ &= \frac{8.68}{5} = 1.736, \end{aligned}$$

and similarly

$$\bar{x}_2 = \frac{1}{5}(65.0 + 85.0 + 54.5 + 72.0 + 94.5) = 74.2.$$

So our mean vector is

$$\bar{\boldsymbol{x}} = (1.736, 74.2)^{\mathsf{T}}.$$

Next we can calculate it as the sample mean of the observation vectors as

$$\begin{aligned} \bar{\boldsymbol{x}} &= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = \frac{1}{5}\left[\begin{pmatrix} 1.67 \\ 65.0 \end{pmatrix} + \begin{pmatrix} 1.78 \\ 85.0 \end{pmatrix} + \begin{pmatrix} 1.60 \\ 54.5 \end{pmatrix} + \begin{pmatrix} 1.83 \\ 72.0 \end{pmatrix} + \begin{pmatrix} 1.80 \\ 94.5 \end{pmatrix}\right] \\ &= \begin{pmatrix} 1.736 \\ 74.2 \end{pmatrix}. \end{aligned}$$

Finally, we can use our matrix expression for the sample mean

$$\bar{\boldsymbol{x}} = \frac{1}{n}\mathsf{X}^{\mathsf{T}}\mathbf{1}_n = \frac{1}{5}\begin{pmatrix} 1.67 & 1.78 & 1.60 & 1.83 & 1.80 \\ 65.0 & 85.0 & 54.5 & 72.0 & 94.5 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{5}\begin{pmatrix} 8.68 \\ 371.00 \end{pmatrix} = \begin{pmatrix} 1.736 \\ 74.2 \end{pmatrix}.$$

This example hopefully makes clear that our three different ways of thinking about computing the sample mean are all equivalent. However, the final method based on a matrix multiplication operation is the neatest both mathematically and computationally, and so we will make use of this expression, as well as other similar expressions, throughout the course.

1.3.2 Sample variance and covariance

Just as for the mean, we can think about other summaries starting from univariate summaries applied to the individual variables of the data matrix. We write the sample variance of $\boldsymbol{x}^{(j)}$ as s_j^2 or s_{jj}, and calculate as

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

Similarly, we can calculate the sample covariance between variables j and k as

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

which clearly reduces to the sample variance when $k = j$. If we compute the sample covariances for all $j, k = 1, 2, \ldots, p$, then we can use them to form a $p \times p$ matrix,

$$\mathsf{S} = \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{pmatrix},$$

known as the *sample covariance matrix*, or *sample variance matrix*, or sometimes as the *sample variance-covariance matrix*. Again, a with a little thought, one can see that we can construct this matrix directly from the observation vectors.

Definition 3 *The* sample variance matrix *is defined by*

$$\mathsf{S} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^{\mathsf{T}}.$$

This expression is much simpler than computing each component individually, but can be simplified further if we consider the problem of *centering* the data.

We can obviously write the sample covariance matrix as

$$\mathsf{S} = \frac{1}{n-1}\sum_{i=1}^{n}\boldsymbol{w}_i\boldsymbol{w}_i^{\mathsf{T}},$$

where $\boldsymbol{w}_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}},\ i = 1, 2, \ldots, n$, and we can regard the w_i as observations from a new

$n \times p$ data matrix W. We can construct W as

$$\begin{aligned}
\mathsf{W} &= \mathsf{X} - \begin{pmatrix} \bar{\boldsymbol{x}}^\mathsf{T} \\ \bar{\boldsymbol{x}}^\mathsf{T} \\ \vdots \\ \bar{\boldsymbol{x}}^\mathsf{T} \end{pmatrix} \\
&= \mathsf{X} - \mathbf{1}_n \bar{\boldsymbol{x}}^\mathsf{T} \\
&= \mathsf{X} - \mathbf{1}_n \left[\frac{1}{n} \mathsf{X}^\mathsf{T} \mathbf{1}_n \right]^\mathsf{T} \\
&= \mathsf{X} - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n{}^\mathsf{T} \mathsf{X} \\
&= \left(\mathrm{I}_{n \times n} - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n{}^\mathsf{T} \right) \mathsf{X} \\
&= \mathsf{H}_n \mathsf{X},
\end{aligned}$$

where

Definition 4

$$\mathsf{H}_n \equiv \mathrm{I}_{n \times n} - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n{}^\mathsf{T}$$

is known as the centering matrix.

So we can subtract the mean from a data matrix by pre-multiplying by the centering matrix. This isn't a numerically efficient way to strip out the mean, but is mathematically elegant and convenient. The centering matrix has several useful properties that we can exploit.

Proposition 2 *The centering matrix* H_n *has the following properties:*

1. H_n *is* symmetric, $\mathsf{H}_n{}^\mathsf{T} = \mathsf{H}_n$,
2. H_n *is* idempotent, $\mathsf{H}_n^2 = \mathsf{H}_n$,
3. *If* X *is an* $n \times p$ *data matrix, then the* $n \times p$ *matrix* $\mathsf{W} = \mathsf{H}_n \mathsf{X}$ *has sample mean equal to the zero* p*-vector.*

Proof

These are trivial exercises in matrix algebra. 1. and 2. are left as exercises. We will use symmetry to show 3.

$$\begin{aligned}
\bar{\boldsymbol{w}} &= \frac{1}{n} \mathsf{W}^\mathsf{T} \mathbf{1}_n \\
&= \frac{1}{n} (\mathsf{H}_n \mathsf{X})^\mathsf{T} \mathbf{1}_n \\
&= \frac{1}{n} \mathsf{X}^\mathsf{T} \mathsf{H}_n \mathbf{1}_n \\
&= \frac{1}{n} \mathsf{X}^\mathsf{T} (\mathrm{I}_{n \times n} - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n{}^\mathsf{T}) \mathbf{1}_n \\
&= \frac{1}{n} \mathsf{X}^\mathsf{T} (\mathbf{1}_n - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n{}^\mathsf{T} \mathbf{1}_n) \\
&= \frac{1}{n} \mathsf{X}^\mathsf{T} (\mathbf{1}_n - \mathbf{1}_n) \\
&= \mathbf{0}.
\end{aligned}$$

□

Re-writing the sample variance matrix, S, in terms of the centered data matrix W, is useful, since we can now simplify the expression further. We first write

$$\sum_{i=1}^{n} \boldsymbol{w}_i \boldsymbol{w}_i^{\mathsf{T}} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n) \begin{pmatrix} \boldsymbol{w}_1^{\mathsf{T}} \\ \boldsymbol{w}_2^{\mathsf{T}} \\ \vdots \\ \boldsymbol{w}_n^{\mathsf{T}} \end{pmatrix} = \mathsf{W}^{\mathsf{T}}\mathsf{W},$$

using properties of block matrix multiplication. From this we conclude that

$$\mathsf{S} = \frac{1}{n-1} \sum_{i=1}^{n} \boldsymbol{w}_i \boldsymbol{w}_i^{\mathsf{T}} = \frac{1}{n-1} \mathsf{W}^{\mathsf{T}}\mathsf{W}$$

We can then substitute back in our definition of W using the centering matrix to get

$$\mathsf{S} = \frac{1}{n-1}\mathsf{W}^{\mathsf{T}}\mathsf{W} = \frac{1}{n-1}(\mathsf{H}_n\mathsf{X})^{\mathsf{T}}\mathsf{H}_n\mathsf{X}$$
$$= \frac{1}{n-1}\mathsf{X}^{\mathsf{T}}\mathsf{H}_n{}^{\mathsf{T}}\mathsf{H}_n\mathsf{X} = \frac{1}{n-1}\mathsf{X}^{\mathsf{T}}\mathsf{H}_n\mathsf{X},$$

using symmetry and idempotency of H_n. This gives us the rather elegant result:

Proposition 3 *The sample variance matrix can be written*

$$\mathsf{S} = \frac{1}{n-1}\mathsf{X}^{\mathsf{T}}\mathsf{H}_n\mathsf{X}.$$

We shall make considerable use of this result.

Example: galaxy data

With the galaxy data, we can start by computing the column variances with

```
> apply(galaxy,2,var)
      east.west     north.south           angle radial.position
       144.6609        523.8497       1462.6269        670.2299
       velocity
      8886.4772
```

which gives diagonal elements consistent with the built-in `var()` function:

```
> var(galaxy)
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west         451.46551
north.south     -1929.95131
angle              37.21646
radial.position  1637.78301
velocity         8886.47724
```

as can be verified with

```
> diag(var(galaxy))
      east.west     north.south           angle radial.position
       144.6609        523.8497       1462.6269        670.2299
       velocity
      8886.4772
```

The built-in `var()` function is very efficient, and should generally be preferred as the way to compute a sample variance matrix in R. However, we can easily check that our mathematically elegant matrix result works using

```
> H=diag(323)-matrix(1/323,ncol=323,nrow=323)
> t(galaxy)%*%H%*%as.matrix(galaxy)/(323-1)
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west          451.46551
north.south      -1929.95131
angle               37.21646
radial.position   1637.78301
velocity          8886.47724
```

This method is clearly mathematically correct. However, the problem is that building the $n \times n$ centering matrix and carrying out a matrix multiplication using it is a very computationally inefficient way to simply strip the mean out of a data matrix. If we wanted to implement our own method more efficiently, we could do it along the following lines

```
> Wt=t(galaxy)-colMeans(galaxy)
> Wt %*% t(Wt)/(323-1)
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west          451.46551
north.south      -1929.95131
angle               37.21646
radial.position   1637.78301
velocity          8886.47724
```

This uses a couple of R tricks, relying on the fact that R stores matrices in column-major order and "recycles" short vectors. We can improve on this slightly by directly constructing the outer product $\mathbf{1}_n \bar{\boldsymbol{x}}^\mathsf{T}$.

```
> W=galaxy-outer(rep(1,323),colMeans(galaxy))
> W=as.matrix(W)
> t(W)%*%W/(323-1)
```

```
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west          451.46551
north.south      -1929.95131
angle              37.21646
radial.position   1637.78301
velocity          8886.47724
```

In fact, we can do even better than this by exploiting the **R** function `sweep()`, which is intended to be used for exactly this sort of centering procedure, where some statistics are to be "swept" out of a data frame.

```
> W=as.matrix(sweep(galaxy,2,colMeans(galaxy)))
> t(W)%*%W/(323-1)
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west          451.46551
north.south      -1929.95131
angle              37.21646
radial.position   1637.78301
velocity          8886.47724
```

Example: ZIP code digits

For our handwritten image data, we can easily compute the sample variance matrix for the 256 columns corresponding to the image with

```
> v=var(zip.train[,-1])
> dim(v)
[1] 256 256
> prod(dim(v))
[1] 65536
```

There probably isn't much point in printing the resulting 256×256 matrix to the screen and reading through all 65,536 covariances. However, we have already seen that we can sometimes use images to visualise large matrices, and we can also do that here

```
image(v[,256:1],col=gray(15:0/15),axes=FALSE)
```

giving the plot shown in Figure 1.8.

The image shows a narrow dark band down the main diagonal corresponding to the pixel variances, but alternating between dark and light, due to the pixels near the edge

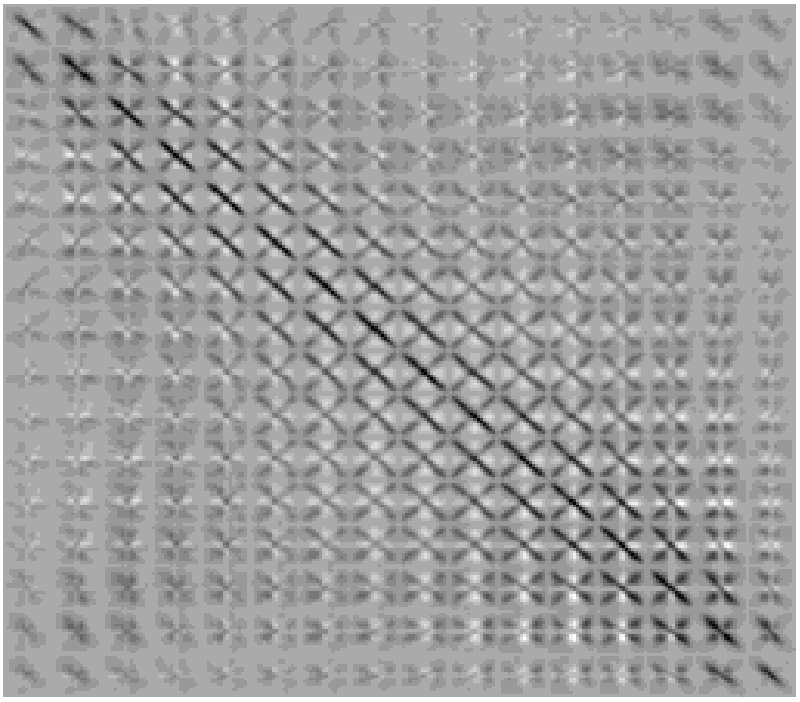


Figure 1.8: An image of the variance matrix for the `zip.train` dataset

of each row having relatively low variance (due to usually being blank). Then we see parallel bands every 16 pixels due to pixels adjacent on consecutive rows being highly correlated. The bands perpendicular to the leading diagonal are at first a little more mysterious, but interestingly, these arise from the fact that many digits have a degree of bilateral symmetry, with digits such as "1" and "8" being classic examples, but others such as "6", "5" and "2" also having some degree of bilateral symmetry.

Example

To work through a simple example by hand, consider a data matrix with $n = 3$ and $p = 2$:

$$\mathsf{X} = \begin{pmatrix} 2 & 3 \\ 4 & 1 \\ 6 & 2 \end{pmatrix}$$

First, let's calculate the sample covariance matrix using the original definition: For this we first need to compute

$$\bar{\boldsymbol{x}} = (4, 2)^{\mathsf{T}}$$

Then we have

$$\begin{aligned}
\mathsf{S} &= \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^\mathsf{T} \\
&= \frac{1}{2}\left\{\left[\begin{pmatrix}2\\3\end{pmatrix}-\begin{pmatrix}4\\2\end{pmatrix}\right]\left[\begin{pmatrix}2\\3\end{pmatrix}-\begin{pmatrix}4\\2\end{pmatrix}\right]^\mathsf{T}\right. \\
&\quad + \left[\begin{pmatrix}4\\1\end{pmatrix}-\begin{pmatrix}4\\2\end{pmatrix}\right]\left[\begin{pmatrix}4\\1\end{pmatrix}-\begin{pmatrix}4\\2\end{pmatrix}\right]^\mathsf{T} \\
&\quad \left. + \left[\begin{pmatrix}6\\2\end{pmatrix}-\begin{pmatrix}4\\2\end{pmatrix}\right]\left[\begin{pmatrix}6\\2\end{pmatrix}-\begin{pmatrix}4\\2\end{pmatrix}\right]^\mathsf{T}\right\} \\
&= \frac{1}{2}\left\{\begin{pmatrix}-2\\1\end{pmatrix}(-2,1)+\begin{pmatrix}0\\-1\end{pmatrix}(0,-1)+\begin{pmatrix}2\\0\end{pmatrix}(2,0)\right\} \\
&= \left\{\begin{pmatrix}4&-2\\-2&1\end{pmatrix}+\begin{pmatrix}0&0\\0&1\end{pmatrix}+\begin{pmatrix}4&0\\0&0\end{pmatrix}\right\} \\
&= \frac{1}{2}\begin{pmatrix}8&-2\\-2&2\end{pmatrix} = \begin{pmatrix}4&-1\\-1&1\end{pmatrix}
\end{aligned}$$

Next, let's calculate using our formula based around the centering matrix First calculate

$$\mathsf{H}_3 = \mathrm{I}_3 - \frac{1}{3}\mathbf{1}_3\mathbf{1}_3^\mathsf{T} = \begin{pmatrix}\frac{2}{3}&\frac{-1}{3}&\frac{-1}{3}\\\frac{-1}{3}&\frac{2}{3}&\frac{-1}{3}\\\frac{-1}{3}&\frac{-1}{3}&\frac{2}{3}\end{pmatrix},$$

and then use this to compute

$$\begin{aligned}
\mathsf{S} &= \frac{1}{n-1}\mathsf{X}^\mathsf{T}\mathsf{H}_n\mathsf{X} \\
&= \frac{1}{2}\begin{pmatrix}2&4&6\\3&1&2\end{pmatrix}\begin{pmatrix}\frac{2}{3}&\frac{-1}{3}&\frac{-1}{3}\\\frac{-1}{3}&\frac{2}{3}&\frac{-1}{3}\\\frac{-1}{3}&\frac{-1}{3}&\frac{2}{3}\end{pmatrix}\begin{pmatrix}2&3\\4&1\\6&2\end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix}2&4&6\\3&1&2\end{pmatrix}\begin{pmatrix}-2&1\\0&-1\\2&0\end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix}8&-2\\-2&2\end{pmatrix} = \begin{pmatrix}4&-1\\-1&1\end{pmatrix}
\end{aligned}$$

Finally, let's think about how to do this computation a little more efficiently. Let's start by constructing

$$\mathsf{W} = \mathsf{X} - \mathbf{1}_3\bar{\boldsymbol{x}}^\mathsf{T} = \begin{pmatrix}-2&1\\0&-1\\2&0\end{pmatrix}$$

Now

$$\begin{aligned}
\mathsf{S} &= \frac{1}{2}\mathsf{W}^\mathsf{T}\mathsf{W} = \frac{1}{2}\begin{pmatrix}-2&0&2\\1&-1&0\end{pmatrix}\begin{pmatrix}-2&1\\0&-1\\2&0\end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix}8&-2\\-2&2\end{pmatrix} = \begin{pmatrix}4&-1\\-1&1\end{pmatrix}
\end{aligned}$$

So, the variances of the variables are 4 and 1, and the covariance between them is -1, which indicates that the variables are negatively correlated.

Before moving on, it is worth dwelling a little more on the intermediate result

$$\mathsf{S} = \frac{1}{n-1}\mathsf{W}^\mathsf{T}\mathsf{W},$$

which shows us that the matrix S *factorises* in a very nice way. We will return to matrix factorisation later, but for now it is worth noting that we can re-write the above as

$$\mathsf{S} = \mathsf{A}^\mathsf{T}\mathsf{A}$$

where

$$\mathsf{A} = \frac{1}{\sqrt{n-1}}\mathsf{W}.$$

Matrices which factorise in this way have two very important properties

Proposition 4 *Let* S *be a* $p \times p$ *matrix of the form*

$$\mathsf{S} = \mathsf{A}^\mathsf{T}\mathsf{A},$$

where A *is an* $n \times p$ *matrix. Then* S *has the following properties:*

1. S *is symmetric,* $\mathsf{S}^\mathsf{T} = \mathsf{S}$*,*
2. S *is* positive semi-definite *(or* non-negative definite*), written* $\mathsf{S} \geq 0$*. This means that* $\boldsymbol{\alpha}^\mathsf{T}\mathsf{S}\boldsymbol{\alpha} \geq 0$ *for all choices of* p*-vector* $\boldsymbol{\alpha}$*.*

Proof
For 1.,

$$\mathsf{S}^\mathsf{T} = (\mathsf{A}^\mathsf{T}\mathsf{A})^\mathsf{T} = \mathsf{A}^\mathsf{T}(\mathsf{A}^\mathsf{T})^\mathsf{T} = \mathsf{A}^\mathsf{T}\mathsf{A} = \mathsf{S}.$$

For 2., note that

$$\begin{aligned}\boldsymbol{\alpha}^\mathsf{T}\mathsf{S}\boldsymbol{\alpha} &= \boldsymbol{\alpha}^\mathsf{T}\mathsf{A}^\mathsf{T}\mathsf{A}\boldsymbol{\alpha} \\ &= (\mathsf{A}\boldsymbol{\alpha})^\mathsf{T}(\mathsf{A}\boldsymbol{\alpha}) \\ &= \|\mathsf{A}\boldsymbol{\alpha}\|_2^2 \\ &\geq 0.\end{aligned}$$

□

So the sample variance matrix is symmetric and non-negative definite.

1.3.3 Sample correlation

The *sample correlation* between variables i and j is defined by

$$r_{ij} = \frac{s_{ij}}{s_i s_j}.$$

Clearly, from the definition of sample covariance we have that $r_{ii} = 1,\ i = 1, 2, \ldots, p$, and putting $\boldsymbol{w}^{(i)} = \boldsymbol{x}^{(i)} - \bar{x}_i \mathbf{1}_n$,

$$r_{ij} = \frac{\boldsymbol{w}^{(i)\mathsf{T}} \boldsymbol{w}^{(j)}}{\sqrt{\boldsymbol{w}^{(i)\mathsf{T}} \boldsymbol{w}^{(i)}} \sqrt{\boldsymbol{w}^{(j)\mathsf{T}} \boldsymbol{w}^{(j)}}} = \frac{\boldsymbol{w}^{(i)} \cdot \boldsymbol{w}^{(j)}}{\|\boldsymbol{w}^{(i)}\| \|\boldsymbol{w}^{(j)}\|}$$

and so by the Cauchy-Schwarz inequality we have $|r_{ij}| \leq 1,\ i, j = 1, 2, \ldots, p$. As with the sample covariance matrix, we can create a $p \times p$ matrix of sample correlations, known as the *sample correlation matrix*, R. If $\mathsf{R} = \mathrm{I}_p$, so that all off-diagonal correlations are zero, then we say that the variables are *uncorrelated*.

If we now define D to be the diagonal matrix of sample standard deviations, that is

$$\mathsf{D} = \mathrm{diag}\{s_1, s_2, \ldots, s_p\} = \begin{pmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_p \end{pmatrix},$$

then we get the following result.

Proposition 5 *The sample correlation matrix may be computed from the sample variance matrix as*

$$\mathsf{R} = \mathsf{D}^{-1}\mathsf{S}\mathsf{D}^{-1},$$

and conversely, the sample covariance matrix may be computed from the sample correlation matrix as

$$\mathsf{S} = \mathsf{D}\mathsf{R}\mathsf{D}.$$

Proof

This follows since pre-multiplying by a diagonal matrix re-scales the rows and post-multiplying by a diagonal matrix re-scales the columns. □

Proposition 6 *The sample correlation matrix is symmetric and positive semi-definite.*

Proof

We have already shown that the sample covariance matrix can be written as $\mathsf{S} = \mathsf{A}^{\mathsf{T}}\mathsf{A}$. Consequently, R can be written as

$$\mathsf{R} = \mathsf{B}^{\mathsf{T}}\mathsf{B},$$

where $\mathsf{B} = \mathsf{A}\mathsf{D}^{-1}$. We showed earlier that matrices that can be written in this form must be symmetric and positive semi-definite. □

Example: Galaxy data

Returning to the galaxy data example, we can compute the sample correlation matrix directly using the built-in **R** command `cor()` as follows

```
> cor(galaxy)
                 east.west north.south       angle radial.position
east.west        1.00000000 -0.11871407 -0.04674954       0.8476414
north.south     -0.11871407  1.00000000  0.03011136      -0.4418112
angle           -0.04674954  0.03011136  1.00000000      -0.0495015
radial.position  0.84764144 -0.44181124 -0.04950150       1.0000000
velocity         0.39818438 -0.89449554  0.01032294       0.6710883
                  velocity
east.west        0.39818438
north.south     -0.89449554
angle            0.01032294
radial.position  0.67108826
velocity         1.00000000
```

So, for example, the correlation between radial position and velocity is 0.671, as we can verify with

```
> cor(galaxy$radial.position,galaxy$velocity)
[1] 0.6710883
```

Using the built-in `cor()` function is the preferred way to compute a sample correlation matrix, as it is computationally efficient. Nevertheless, we can calculate using Proposition 5 as

```
> diag(1/apply(galaxy,2,sd)) %*% var(galaxy) %*% diag(1/apply(galaxy,2,
   sd))
             [,1]        [,2]        [,3]       [,4]        [,5]
[1,]  1.00000000 -0.11871407 -0.04674954  0.8476414  0.39818438
[2,] -0.11871407  1.00000000  0.03011136 -0.4418112 -0.89449554
[3,] -0.04674954  0.03011136  1.00000000 -0.0495015  0.01032294
[4,]  0.84764144 -0.44181124 -0.04950150  1.0000000  0.67108826
[5,]  0.39818438 -0.89449554  0.01032294  0.6710883  1.00000000
```

Example: ZIP digits

Again, we can easily compute the correlation matrix using

```
R=cor(zip.train[,-1])
```

but we cannot easily comprehend all of the numbers in this huge matrix. So we can image the correlation matrix in exactly the same way that we imaged the sample variance matrix earlier, viz

```
image(R[,256:1],col=gray(15:0/15),axes=FALSE)
```

giving the image shown in Figure 1.9.

We see that the overall pattern looks quite similar to the image of the variance matrix we created earlier. However, whilst this image obviously loses the information about variance (the diagonal is now solid black), it better highlights the correlation structure, bringing out a somewhat stronger off-diagonal pattern.

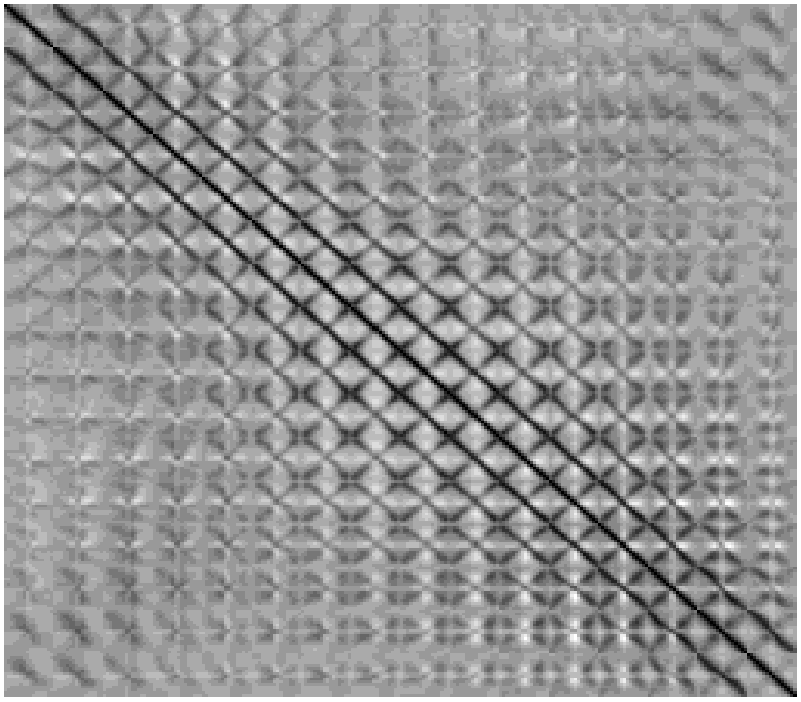


Figure 1.9: An image of the sample correlation matrix for the `zip.train` dataset

Example

Earlier we calculated by hand a 2×2 sample variance matrix

$$\mathsf{S} = \begin{pmatrix} 4 & -1 \\ -1 & 1 \end{pmatrix}$$

We can now use Proposition 5 to calculate the associated correlation matrix as

$$\begin{aligned} R &= \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 \\ -1/2 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & -1/2 \\ -1/2 & 1 \end{pmatrix}, \end{aligned}$$

giving a sample correlation coefficient between the two variables of -0.5.

1.4 Multivariate random quantities

So far everything we have been doing has been purely descriptive, focused on understanding the particular dataset we have available. Usually, however, in statistics we want at some point to go beyond this, and use the data at hand to say something predictive about the larger population from which the data was sampled. To do this, we need to make some modelling assumptions. In the context of an $n \times p$ data matrix, X, the typical assumption made is that the observations, $\boldsymbol{x_i} \in \mathbb{R}^p$ have been drawn *independently* from some unknown probability distribution on $\mathbb{R}^p$. So the really key distinction between the rows and columns of a data matrix is that we typically assume that the rows are independent but that the columns are not (necessarily). A *random vector* $\boldsymbol{X}$ takes the form

$$\boldsymbol{X} = (X_1, X_2, \ldots, X_p)^{\mathsf{T}}$$

where each $X_j,\ j = 1, 2, \ldots, p$ is a real-valued random variable. We *do not* assume that the components of the random vector are independent of one another.

Definition 5 *The* expectation *of a random p-vector $\boldsymbol{X}$ is defined by*

$$\mathrm{E}(\boldsymbol{X}) = [\,\mathrm{E}(X_1)\,,\, \mathrm{E}(X_2)\,, \ldots,\, \mathrm{E}(X_p)]^{\mathsf{T}}.$$

So, the expectation of a random vector is just defined to be the vector of expected values. Similarly, *the expected value of a random matrix is the matrix of expected values*. Also, if we recall that the covariance between two scalar random variables X and Y is defined by

$$\mathrm{Cov}(X, Y) = \mathrm{E}([X - \mathrm{E}(X)][Y - \mathrm{E}(Y)])$$

(so for independent X, Y we will have $\mathrm{Cov}(X, Y) = 0$), we see that the variance matrix for a random matrix is defined to be the matrix of covariances between pairs of elements:

Definition 6 *The* variance matrix *of a random p-vector $\boldsymbol{X}$ is defined by*

$$\mathrm{Var}(\boldsymbol{X}) = \mathrm{E}\big([\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\big)\,.$$

Example

Suppose that $\boldsymbol{X} = (X_1, X_2)^{\mathsf{T}}$, where $X_1 \sim Exp(1)$ and $X_2 \sim Exp(2)$ are independent random quantities. What is $\mathrm{E}(\boldsymbol{X})$ and $\mathrm{Var}(\boldsymbol{X})$? Calculating everything required one component at a time, we have $\mathrm{E}(X_1) = 1$, $\mathrm{Var}(X_1) = 1$, $\mathrm{E}(X_2) = 1/2$, $\mathrm{Var}(X_2) = 1/4$, and since the variables are independent, we must have $\mathrm{Cov}(X_1, X_2) = 0$, giving

$$\mathrm{E}(\boldsymbol{X}) = \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}, \quad \mathrm{Var}(\boldsymbol{X}) = \begin{pmatrix} 1 & 0 \\ 0 & 1/4 \end{pmatrix}.$$

We will show later that in some appropriate sense, if we regard a multivariate dataset X as a random sample from a multivariate distribution, then it turns out that the sample mean of X is a "good" estimator of $\mathrm{E}(\boldsymbol{X})$, and similarly, the sample variance matrix, S, is a "good" estimator of $\mathrm{Var}(\boldsymbol{X})$.

1.5 Transformation and manipulation of multivariate random quantities and data

When working with multivariate random quantities and data, we often want to transform them in some way in order to highlight some aspect of interest, or reduce dimension, etc. We have already seen some examples of transformations applied to multivariate data. Subsetting columns from a data matrix is a particularly simple example, and formally can be regarded as a projection into a lower dimensional space. In particular, picking out two columns in order to produce a scatter-plot is an example of a projection of the data into a 2-dimensional space. The centering matrix was another example of a transformation of the data matrix, corresponding to subtraction of the mean. These simple examples are all examples of affine transformations, and so it will be helpful to remind ourselves of the basic idea behind affine transformations before proceeding to see how we can use them to transform random vectors and multivariate data.

1.5.1 Linear and affine transformations

In Euclidean space a *linear transformation* (or linear map) is represented by a real-valued matrix. For example, the map

$$f : \mathbb{R}^p \longrightarrow \mathbb{R}^q$$

will have matrix representation

$$f(\boldsymbol{x}) = \mathsf{A}\boldsymbol{x}$$

for some $q \times p$ matrix A. In the context of multivariate statistics we would expect that $q \leq p$, so that the map will either preserve the dimension of the original space $q = p$, or project the data into a lower dimensional space $q < p$. Since it is clear that

$$f(\boldsymbol{e}_j) = \mathsf{A}\boldsymbol{e}_j = \boldsymbol{a}^{(j)},$$

the jth column of A, we can form A by considering the destination of $\boldsymbol{e}_j,\ j = 1, 2, \ldots, p$ in the new space, $\mathbb{R}^q$.

An important special case arises in the case $q = 1$, so that the map is real valued, and represents a linear functional. Linear functionals can be represented with a p-vector, $\boldsymbol{a}$, as

$$f(\boldsymbol{x}) = \boldsymbol{a}^{\mathsf{T}}\boldsymbol{x}.$$

It turns out that in many applications in multivariate statistics, we want to combine a linear transformation with a shift in location. Such maps are referred to as affine transformations, and the affine map

$$f : \mathbb{R}^p \longrightarrow \mathbb{R}^q$$

can be represented as

$$f(\boldsymbol{x}) = \mathsf{A}\boldsymbol{x} + \boldsymbol{b},$$

for some $q \times p$ matrix A and q-vector $\boldsymbol{b}$. Note that $f(\boldsymbol{0}) = \boldsymbol{b}$, so that $\boldsymbol{b}$ can be identified as the image of $\boldsymbol{0}$, and then

$$f(\boldsymbol{e}_j) = \boldsymbol{a}^{(j)} + \boldsymbol{b},$$

and hence the columns of A are determined by

$$\boldsymbol{a}^{(j)} = f(\boldsymbol{e_j}) - \boldsymbol{b}.$$

Note that affine transformations can be represented with a single $(q+1) \times (p+1)$ matrix of the form

$$\mathcal{A} = \begin{pmatrix} \mathsf{A} & \boldsymbol{b} \\ \boldsymbol{0}^\mathsf{T} & 1 \end{pmatrix}$$

by augmenting the vectors in both spaces with a "1", since then

$$\begin{aligned} \begin{pmatrix} \boldsymbol{y} \\ 1 \end{pmatrix} &= \mathcal{A} \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \mathsf{A} & \boldsymbol{b} \\ \boldsymbol{0}^\mathsf{T} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \mathsf{A}\boldsymbol{x} + \boldsymbol{b} \\ 1 \end{pmatrix}, \end{aligned}$$

and is therefore equivalent to

$$\boldsymbol{y} = \mathsf{A}\boldsymbol{x} + \boldsymbol{b}.$$

This technique of representing an affine transformation as a linear transformation in an augmented space makes it convenient to combine multiple affine transformations (via matrix multiplication), and is used extensively in computer graphics.

Example: linear combination

The linear combination of variables

$$y = a_1 x_1 + a_2 x_2 + \cdots + a_p x_p + b$$

can be represented as the affine transformation

$$y = \boldsymbol{a}^\mathsf{T} \boldsymbol{x} + b.$$

Example: 2d projection

We may view the selection of variables to plot in one window of a scatterplot matrix as a 2d projection of the multivariate data. Selection of components x_k and x_j corresponds to the projection

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{e}_k{}^\mathsf{T} \\ \boldsymbol{e}_j{}^\mathsf{T} \end{pmatrix} \boldsymbol{x},$$

and hence the linear transformation with $\mathsf{A} = (\boldsymbol{e}_k, \boldsymbol{e}_j)^\mathsf{T}$.

More generally, we can project onto the linear subspace spanned by any two orthogonal unit p-vectors, $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ using

$$\mathsf{A} = (\boldsymbol{v}_1, \boldsymbol{v}_2)^\mathsf{T} = \begin{pmatrix} \boldsymbol{v}_1{}^\mathsf{T} \\ \boldsymbol{v}_2{}^\mathsf{T} \end{pmatrix}.$$

This follows since we can write $\boldsymbol{x}$ as

$$\boldsymbol{x} = y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \boldsymbol{w},$$

where $\boldsymbol{w}$ is orthogonal to $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$. From this it follows that

$$\begin{aligned}\boldsymbol{v}_i \cdot \boldsymbol{x} &= \boldsymbol{v}_i \cdot (y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \boldsymbol{w}) \\ &= y_i.\end{aligned}$$

In other words, $\boldsymbol{v}_i{}^{\mathsf{T}}\boldsymbol{x} = y_i$, and the result follows. Note that, as above, A has the property $\mathsf{A}\mathsf{A}^{\mathsf{T}} = \mathrm{I}_2$. 2d projections are obviously very useful for visualisation of high-dimensional data on a computer screen.

1.5.2 Transforming multivariate random quantities

We can apply an affine transformation to a random p-vector $\boldsymbol{X}$ to get a random q-vector $\boldsymbol{Y}$. It is natural to wonder how the expectation and variance of $\boldsymbol{Y}$ relate to that of $\boldsymbol{X}$.

Proposition 7 *Let $\boldsymbol{X}$ be a random p-vector with expectation vector $\mathrm{E}(\boldsymbol{X})$ and variance matrix $\mathrm{Var}(\boldsymbol{X})$. Consider an affine transformation of $\boldsymbol{X}$ from $\mathbb{R}^p$ to $\mathbb{R}^q$ given by*

$$\boldsymbol{Y} = \mathsf{A}\boldsymbol{X} + \boldsymbol{b},$$

where A is a $q \times p$ matrix and $\boldsymbol{b}$ is a q-vector. Then the expectation and variance of $\boldsymbol{Y}$ are given by

$$\mathrm{E}(\boldsymbol{Y}) = \mathsf{A}\ \mathrm{E}(\boldsymbol{X}) + \boldsymbol{b}, \quad \mathrm{Var}(\boldsymbol{Y}) = \mathsf{A}\,\mathrm{Var}(\boldsymbol{X})\,\mathsf{A}^{\mathsf{T}}.$$

Proof

First consider $\mathrm{E}(\boldsymbol{Y})$,

$$\begin{aligned}\mathrm{E}(\boldsymbol{Y}) &= \mathrm{E}(\mathsf{A}\boldsymbol{X} + \boldsymbol{b}) \\ &= \mathrm{E}\left(\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qp} \end{pmatrix}\begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_q \end{pmatrix}\right) \\ &= \mathrm{E}\left(\begin{pmatrix} a_{11}X_1 + a_{12}X_2 + \cdots + a_{1p}X_p + b_1 \\ \vdots \\ a_{q1}X_1 + a_{q2}X_2 + \cdots + a_{qp}X_p + b_q \end{pmatrix}\right) \\ &= \begin{pmatrix} \mathrm{E}(a_{11}X_1 + a_{12}X_2 + \cdots + a_{1p}X_p + b_1) \\ \vdots \\ \mathrm{E}(a_{q1}X_1 + a_{q2}X_2 + \cdots + a_{qp}X_p + b_q) \end{pmatrix} \\ &= \begin{pmatrix} a_{11}\ \mathrm{E}(X_1) + a_{12}\ \mathrm{E}(X_2) + \cdots + a_{1p}\ \mathrm{E}(X_p) + b_1 \\ \vdots \\ a_{q1}\ \mathrm{E}(X_1) + a_{q2}\ \mathrm{E}(X_2) + \cdots + a_{qp}\ \mathrm{E}(X_p) + b_q \end{pmatrix} \\ &= \mathsf{A}\ \mathrm{E}(\boldsymbol{X}) + \boldsymbol{b}.\end{aligned}$$

Once we have established linearity of expectation for vector random quantities, the variance result is relatively straightforward.

$$\begin{aligned}
\mathrm{Var}(\boldsymbol{Y}) &= \mathrm{E}\big([\boldsymbol{Y} - \mathrm{E}(\boldsymbol{Y})][\boldsymbol{Y} - \mathrm{E}(\boldsymbol{Y})]^{\mathsf{T}}\big) \\
&= \mathrm{E}\big([\mathsf{A}\boldsymbol{X} + \boldsymbol{b} - \mathsf{A}\,\mathrm{E}(\boldsymbol{X}) - \boldsymbol{b}][\mathsf{A}\boldsymbol{X} + \boldsymbol{b} - \mathsf{A}\,\mathrm{E}(\boldsymbol{X}) - \boldsymbol{b}]^{\mathsf{T}}\big) \\
&= \mathrm{E}\big([\mathsf{A}\boldsymbol{X} - \mathsf{A}\,\mathrm{E}(\boldsymbol{X})][\mathsf{A}\boldsymbol{X} - \mathsf{A}\,\mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\big) \\
&= \mathrm{E}\big(\mathsf{A}[\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\mathsf{A}^{\mathsf{T}}\big) \\
&= \mathsf{A}\,\mathrm{E}\big([\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\big)\,\mathsf{A}^{\mathsf{T}} \\
&= \mathsf{A}\,\mathrm{Var}(\boldsymbol{X})\,\mathsf{A}^{\mathsf{T}}
\end{aligned}$$

□

We are now in a position to establish that the variance matrix of a random vector shares two important properties with sample covariance matrices.

Proposition 8 *The variance matrix,* $\mathrm{Var}(\boldsymbol{X})$ *is* symmetric *and* positive semi-definite*.*

Proof

$$\begin{aligned}
\mathrm{Var}(\boldsymbol{X})^{\mathsf{T}} &= \{\mathrm{E}\big([\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\big)\}^{\mathsf{T}} \\
&= \mathrm{E}\big(\{[\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\}^{\mathsf{T}}\big) \\
&= \mathrm{E}\big([\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\big) \\
&= \mathrm{Var}(\boldsymbol{X})\,.
\end{aligned}$$

Similarly, for any p-vector $\boldsymbol{\alpha}$ we have

$$\begin{aligned}
\boldsymbol{\alpha}^{\mathsf{T}}\,\mathrm{Var}(\boldsymbol{X})\,\boldsymbol{\alpha} &= \boldsymbol{\alpha}^{\mathsf{T}}\,\mathrm{E}\big([\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\big)\,\boldsymbol{\alpha} \\
&= \mathrm{E}\big(\boldsymbol{\alpha}^{\mathsf{T}}[\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})][\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\boldsymbol{\alpha}\big) \\
&= \mathrm{E}\big(\{[\boldsymbol{X} - \mathrm{E}(\boldsymbol{X})]^{\mathsf{T}}\boldsymbol{\alpha}\}^2\big) \\
&\geq 0,
\end{aligned}$$

since non-negative random scalars cannot have negative expectation. □

Example: mean and variance of a linear combination

The special case of linear functionals of random quantities turns out to be very important. So consider the random (scalar) variable Y defined by

$$Y = \boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{X} + b$$

where $\boldsymbol{\alpha}$ is a given fixed p-vector and b is a fixed scalar. From our above results it is clear that

$$\mathrm{E}(Y) = \boldsymbol{\alpha}^{\mathsf{T}}\,\mathrm{E}(\boldsymbol{X}) + b$$

and

$$\mathrm{Var}(Y) = \boldsymbol{\alpha}^{\mathsf{T}}\,\mathrm{Var}(\boldsymbol{X})\,\boldsymbol{\alpha}.$$

Importantly, we note that the positive semi-definiteness of $\mathrm{Var}(\boldsymbol{X})$ corresponds to the fact that there are no linear combinations of $\boldsymbol{X}$ with negative variance.

Example

Suppose that $\boldsymbol{X} = (X_1, X_2, X_3)^\mathsf{T}$, where the components $X_i \sim Po(i),\ i = 1, 2, 3$ are mutually independent. Consider

$$\boldsymbol{Y} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \boldsymbol{X} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

What are $\mathrm{E}(\boldsymbol{Y})$ and $\mathrm{Var}(\boldsymbol{Y})$? We first need to know that

$$\mathrm{E}(\boldsymbol{X}) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \mathrm{Var}(\boldsymbol{X}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}.$$

The we can calculate

$$\mathrm{E}(\boldsymbol{Y}) = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 14 \\ 32 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 14 \\ 33 \end{pmatrix},$$

and

$$\begin{aligned} \mathrm{Var}(\boldsymbol{Y}) &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 4 & 10 \\ 9 & 18 \end{pmatrix} \\ &= \begin{pmatrix} 36 & 78 \\ 78 & 174 \end{pmatrix}. \end{aligned}$$

Example: transforming standard random vectors

Suppose that we start with a random p-vector $\boldsymbol{Z}$ where the components are mutually independent with zero expectation and variance 1. Now consider a p-vector $\boldsymbol{Y}$ where

$$\boldsymbol{Y} = \mathsf{A}\boldsymbol{Z} + \boldsymbol{\mu},$$

where A is a $p \times p$ matrix and $\boldsymbol{\mu}$ is a p-vector. What is $\mathrm{E}(\boldsymbol{Y})$ and $\mathrm{Var}(\boldsymbol{Y})$?

We first note that $\mathrm{E}(\boldsymbol{Z}) = \boldsymbol{0}$ and $\mathrm{Var}(\boldsymbol{Z}) = \mathrm{I}_p$, and then compute

$$\mathrm{E}(\boldsymbol{Y}) = \mathsf{A}\ \mathrm{E}(\boldsymbol{Z}) + \boldsymbol{\mu} = \boldsymbol{\mu}$$

and

$$\mathrm{Var}(\boldsymbol{Y}) = \mathsf{A}\,\mathrm{Var}(\boldsymbol{Z})\,\mathsf{A}^\mathsf{T} = \mathsf{A}\,\mathrm{I}_p\,\mathsf{A}^\mathsf{T} = \mathsf{A}\mathsf{A}^\mathsf{T}.$$

Notes:

1. Suppose we would like to be able to simulate random quantities with a given expectation $\boldsymbol{\mu}$ and variance Σ. The above transformation provides a recipe to do so, provided that we can simulate iid random quantities with mean zero and variance one, and we have a method to find a matrix A such that $\mathsf{A}\mathsf{A}^\mathsf{T} = \Sigma$. We will examine techniques for factorising Σ in the next chapter.

2. The special case where the elements of $\boldsymbol{Z}$ are iid $N(0,1)$ is of particular importance, as then $\boldsymbol{Y}$ is said to have a multivariate normal distribution. We will investigate this case in more detail later.

1.5.3 Transforming multivariate data

We can apply an affine transformation to each observation of a data matrix, X. Conceptually, we do this by transposing the data matrix to get column vectors of observations, then pre-multiply by A, then add $\boldsymbol{b}$ to each column in order to get the transpose of the new data matrix, Y. In other words

$$\mathsf{Y} = (\mathsf{A}\mathsf{X}^\mathsf{T} + \boldsymbol{b}\mathbf{1}_n{}^\mathsf{T})^\mathsf{T},$$

leading to

Proposition 9 *The affine transformation from $\mathbb{R}^p$ to $\mathbb{R}^q$ given by*

$$\boldsymbol{y} = \mathsf{A}\boldsymbol{x} + \boldsymbol{b}$$

can be applied to an $n \times p$ data matrix X to get a new $n \times q$ data matrix Y given by

$$\mathsf{Y} = \mathsf{X}\mathsf{A}^\mathsf{T} + \mathbf{1}_n\boldsymbol{b}^\mathsf{T}.$$

Again, we would like to know how the sample mean, $\bar{\boldsymbol{y}}$ and variance S_Y of the new data matrix Y relate to the sample mean $\bar{\boldsymbol{x}}$ and variance matrix S_X of X.

Proposition 10 *The sample mean and variance of Y are related to the sample mean and variance of X by*

$$\bar{\boldsymbol{y}} = \mathsf{A}\bar{\boldsymbol{x}} + \boldsymbol{b}, \quad \mathsf{S}_\mathsf{Y} = \mathsf{A}\mathsf{S}_\mathsf{X}\mathsf{A}^\mathsf{T}.$$

Proof

Let's start with the sample mean,

$$\begin{aligned}
\bar{\boldsymbol{y}} &= \frac{1}{n}\mathsf{Y}^\mathsf{T}\mathbf{1}_n \\
&= \frac{1}{n}(\mathsf{X}\mathsf{A}^\mathsf{T} + \mathbf{1}_n\boldsymbol{b}^\mathsf{T})^\mathsf{T}\mathbf{1}_n \\
&= \frac{1}{n}(\mathsf{A}\mathsf{X}^\mathsf{T}\mathbf{1}_n + \boldsymbol{b}\mathbf{1}_n{}^\mathsf{T}\mathbf{1}_n) \\
&= \frac{1}{n}\mathsf{A}\mathsf{X}^\mathsf{T}\mathbf{1} + \boldsymbol{b} \\
&= \mathsf{A}\bar{\boldsymbol{x}} + \boldsymbol{b}.
\end{aligned}$$

The sample variance matrix result is left as an exercise. □

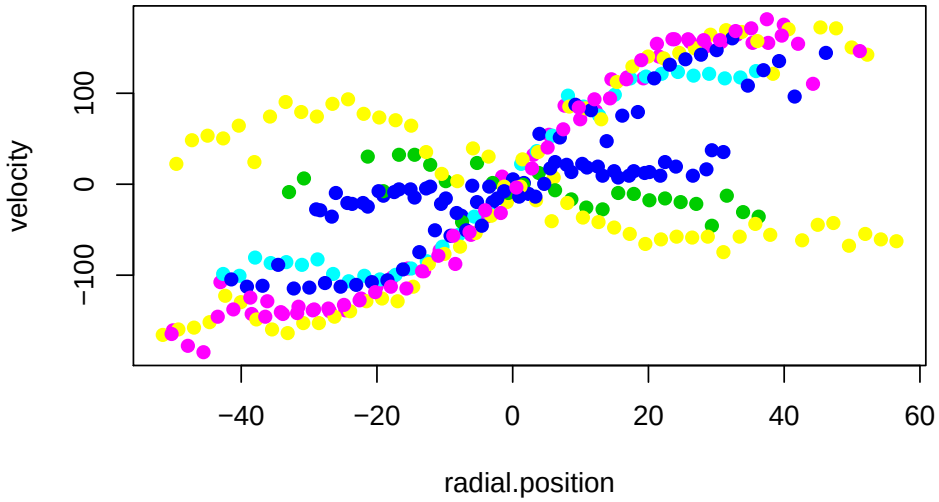


Figure 1.10: A plot of velocity against radial position for the centered galaxy data

Example: Galaxy data

First let's return to the galaxy data, and just extract two columns:

```
sub=galaxy[c("radial.position","velocity")]
```

This is clearly a projection into a 2d space, but there is no point in explicitly constructing the linear transformation. We can then center the data. This is also an affine transformation, which we can construct and then plot with

```
subCentered=sweep(sub,2,colMeans(sub))
plot(subCentered,col=galaxy$angle,pch=19)
```

giving the figure shown in Figure 1.10.

Now suppose that we would like to apply a further transformation to this data, namely rotation anti-clockwise through 10 degrees ($\pi/18$ radians). We know from elementary geometry that this can be accomplished as a linear transformation with matrix

$$\mathsf{A} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix},$$

where $\theta = \pi/18$. We can then apply this transformation to the data matrix and re-plot with

```
A=matrix(c(cos(pi/18),-sin(pi/18),sin(pi/18),cos(pi/18)),ncol=2,byrow=
    TRUE)
subRotated=as.matrix(subCentered) %*% t(A)
plot(subRotated,col=galaxy$angle,pch=19)
```

giving the plot shown in Figure 1.11.

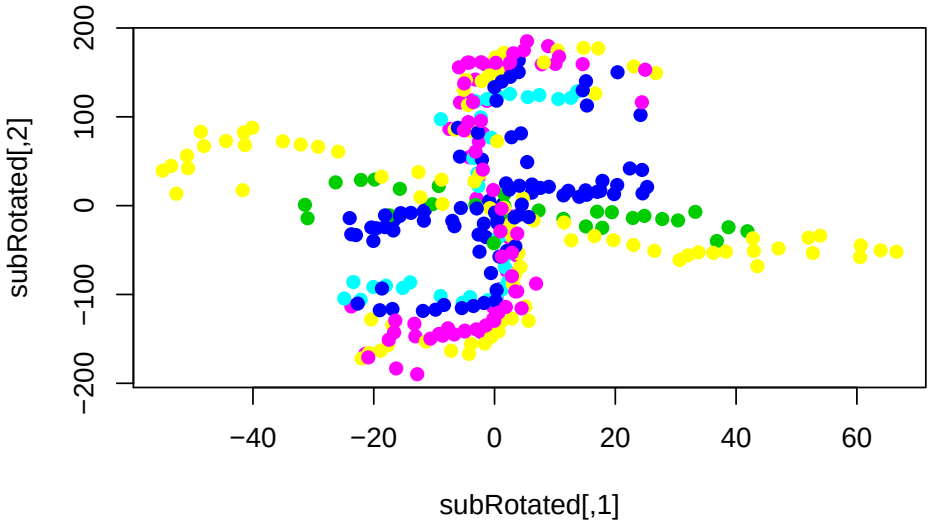


Figure 1.11: A plot of the rotated galaxy data

Example: ZIP digits

Suppose that we are interested in summarising each image by its mean pixel intensity. This is a linear combination of the image vectors in $\mathbb{R}^{256}$ given by

$$y = \frac{1}{256}\mathbf{1}_{256}{}^{\mathsf{T}}\boldsymbol{x}.$$

We could apply this transformation to our data in a variety of ways. First, directly from the transformation as

```
y=zip.train[,-1] %*% rep(1/256,256)
```

We can histogram the resulting vector of mean image intensities with

```
hist(y,30,col=2)
```

giving the image shown in Figure 1.12.

Alternatively, we could have used the `apply()` function to create the vector using `y=apply(zip.train[,-1],1,mean)`. If we are interested in the variance of this vector, we can obviously compute it using

```
> var(y)
[1] 0.0345547
```

On the other hand, if we still have the 256×256 variance matrix for the image data in a variable `v` from a previous example, we could have computed this variance directly without ever explicitly constructing the transformation as

```
> rep(1/256,256) %*% v %*% rep(1/256,256)
             [,1]
[1,] 0.0345547
```

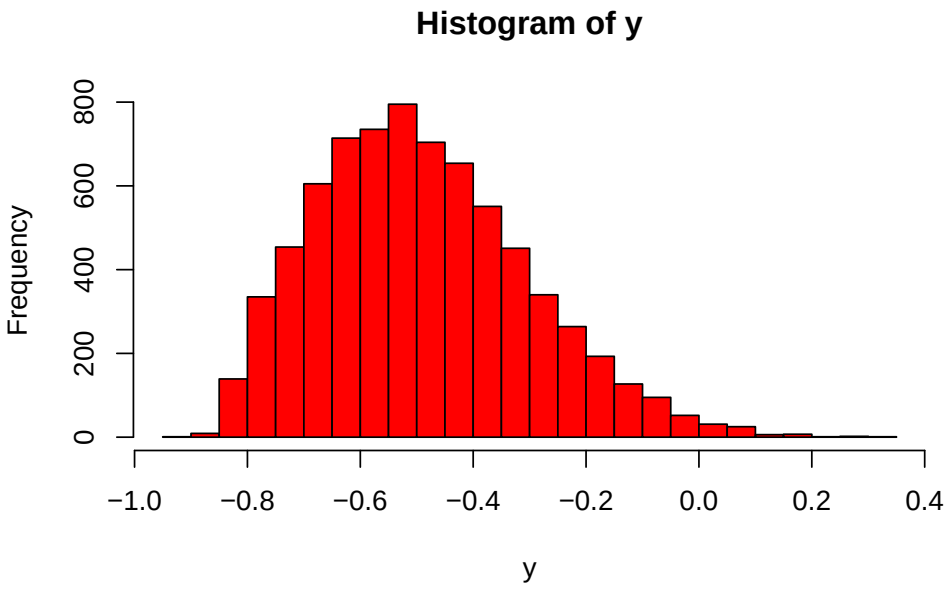


Figure 1.12: Histogram of the mean pixel intensity for each image in the `zip.train` data

Example: normal transformation

If we have $\boldsymbol{Z} = (Z_1, Z_2)^\mathsf{T}$, where Z_1 and Z_2 are iid $N(0,1)$, then the random variable

$$\boldsymbol{Y} = \mathsf{A}\boldsymbol{Z}$$

where A is a 2×2 matrix will have variance matrix $\mathsf{A}\mathsf{A}^\mathsf{T}$. We can investigate this using **R** with the matrix

$$\mathsf{A} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

as follows

```
> Z=matrix(rnorm(2000),ncol=2)
> A=matrix(c(1,0,1,1),ncol=2,byrow=TRUE)
> Y=Z %*% t(A)
> plot(Y,col=2,pch=19)
> A %*% t(A)
     [,1] [,2]
[1,]    1    1
[2,]    1    2
> var(Y)
          [,1]      [,2]
[1,] 1.0194604 0.9926367
[2,] 0.9926367 1.9257760
```

So we see that the sample variance matrix for 1,000 simulated observations of $\boldsymbol{Y}$ is very close to the true variance matrix. The sample correlation is also close to the true correlation of $1/\sqrt{2}$. The plot of the samples is shown in Figure 1.13.

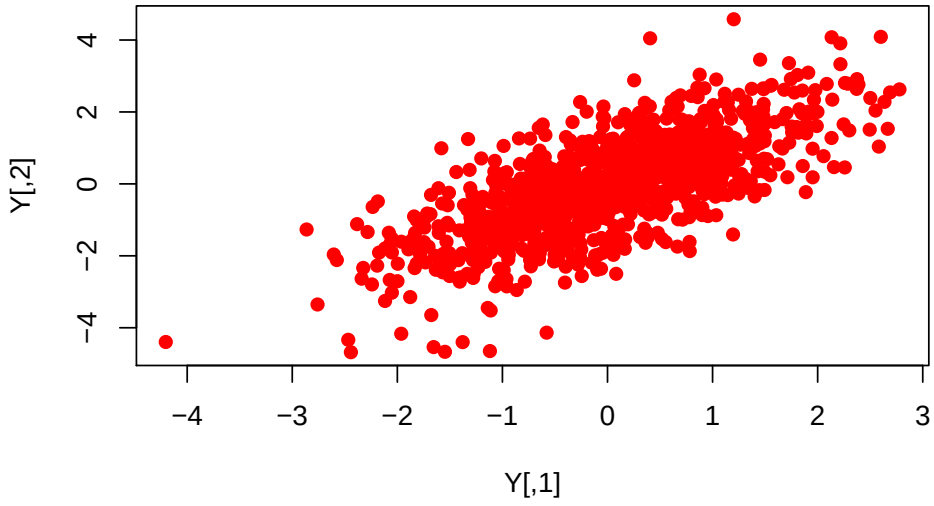


Figure 1.13: Scatterplot of some simulated correlated samples with correlation $1/\sqrt{2}$

Example

Consider the following data matrix with $n = 4$ and $p = 3$:

$$\mathsf{X} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{pmatrix}.$$

Suppose that we wish to construct a new data matrix Y based on the affine transformation

$$\boldsymbol{y} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Construct the 4×2 data matrix Y.

$$\begin{aligned} \mathsf{Y} &= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 3 & 3 \\ 5 & 4 \\ 5 & 1 \\ 4 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 3 & 4 \\ 5 & 5 \\ 5 & 2 \\ 4 & 3 \end{pmatrix}. \end{aligned}$$

When we introduced affine transformations, we discussed the augmented form of the transformation, where the vector to be transformed was augmented with a "1" so that the affine map could be accomplished as a linear transformation. If we wish, we can use this method on a data matrix, X, by augmenting it with a column of "1"s. Then the key transformation equation is

$$(\mathsf{Y}, \mathbf{1}_n) = (\mathsf{X}, \mathbf{1}_n) \begin{pmatrix} \mathsf{A}^\mathsf{T} & \mathbf{0} \\ \boldsymbol{b}^\mathsf{T} & 1 \end{pmatrix}.$$

Note that this is very much analogous to prepending the design matrix with a column of "1"s in multiple linear regression, in order to avoid explicit consideration of an intercept term.

We now have all of the basic skills we need to be able to begin to think about more sophisticated methods of multivariate data analysis. See Chapter 1 of **[ESL]** and Chapters 1 and 2 of **[Everitt]** for further background and details relating to the material in this chapter.

Chapter 2

PCA and matrix factorisations

2.1 Introduction

2.1.1 Factorisation, inversion and linear systems

It should be clear by now that linear algebra and matrix computations are central to multivariate statistics. For the analysis of large and complex datasets it is essential to have a basic knowledge of computational linear algebra, and this starts with an understanding of the efficient solution of linear systems, and the role of various different matrix factorisations, often known as matrix decompositions, to transform general linear systems into very special linear systems that are much easier to solve.

We have already seen a couple of instances of matrix factorisation. First, we showed that the sample covariance matrix, S, could be decomposed as

$$\mathsf{S} = \mathsf{A}^\mathsf{T}\mathsf{A}$$

for some matrix A. We were able to use this fact to deduce important properties of S. Similarly, we showed that applying a matrix A to a vector of independent random quantities with unit variance resulted in a vector with covariance matrix $\mathsf{A}\mathsf{A}^\mathsf{T}$. We noted that if we were interested in simulating random quantities with a given covariance matrix Σ, we could do this if we were able to find a matrix A such that

$$\Sigma = \mathsf{A}\mathsf{A}^\mathsf{T}.$$

This would be an example of a matrix factorisation, or matrix decomposition. Unsurprisingly, there is not a unique solution to this factorisation problem, but in this chapter we will see how to construct two different factorisations of Σ which both provide an effective solution to this problem.

The solution of linear systems will also turn out to be a recurring theme. That is, solving equations of the form

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$$

for $\boldsymbol{x}$, given a specific matrix A and vector $\boldsymbol{b}$. We know that in principle, if A is invertible, the unique solution is given by

$$\boldsymbol{x} = \mathsf{A}^{-1}\boldsymbol{b}.$$

However, it turns out that solving the problem directly in this way, by first computing A^{-1} and then pre-multiplying $\boldsymbol{b}$ by this matrix is a very inefficient and numerically unstable way

to approach the problem. It is hardly ever necessary to directly compute the inverse of a matrix. There are usually far more efficient ways to solve a linear system, sometimes by solving the linear system directly (say, using Gaussian elimination), or more usually by solving in multiple steps using matrix factorisations. By way of motivation, suppose that we have an efficient way to factorise A as

$$\mathsf{A} = \mathsf{L}\mathsf{U},$$

where L and U are lower and upper *triangular* matrices, respectively.* We can then solve $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$ by writing the problem as

$$\mathsf{L}\mathsf{U}\boldsymbol{x} = \boldsymbol{b}.$$

If we define $\boldsymbol{v} = \mathsf{U}\boldsymbol{x}$, we can first solve

$$\mathsf{L}\boldsymbol{v} = \boldsymbol{b}$$

for $\boldsymbol{v}$ and then solve

$$\mathsf{U}\boldsymbol{x} = \boldsymbol{v}$$

for $\boldsymbol{x}$. It turns out that triangular linear systems are very easy to solve, so solving two triangular linear systems is much faster than solving one general linear system. To progress further we need to know more about triangular matrices, linear systems, and factorisations.

2.2 Triangular matrices

2.2.1 Upper and lower triangular matrices

It is possible to develop a theory of triangular matrices and triangular matrix factorisation for non-square matrices, but for the purposes of this course this is unnecessary, so we will restrict ourselves to thinking about square matrices.

Definition 7 *An $n \times n$ matrix* A *is* lower triangular *if all elements above the leading diagonal are zero, that is $a_{ij} = 0,\ \forall j > i$. Similarly, a matrix is* upper triangular *if all elements below the leading diagonal are zero, that is $a_{ij} = 0,\ \forall i > j$.*

Clearly only *diagonal* matrices are *both* upper and lower triangular. Upper and lower triangular matrices have very similar properties (since one is just a transpose of the other). We will focus mainly on lower triangular matrices. It should be assumed that there are exactly analogous properties for upper triangular matrices unless otherwise stated.

Proposition 11

1. *The sum of two lower triangular matrices is also lower triangular*
2. *The product of two lower triangular matrices is also lower triangular*
3. *If a lower triangular matrix is invertible, the inverse is also lower triangular*
4. *The determinant of a lower triangular matrix is the product of the diagonal elements*

*We will explain what this means in the next section. For now it suffices to know that they are just special matrices.

5. *A lower triangular matrix is invertible iff all diagonal elements are non-zero*
6. *The eigenvalues of a lower triangular matrix are the diagonal elements of the matrix*

Proof

1. Clear from definition
2. Left as an exercise
3. Thinking about using the Gaussian elimination method to construct this inverse, it is clear that it will only be necessary to use row operations which are lower triangular, leading to a lower triangular inverse
4. Direct expansion of the determinant along the top row in terms of minor determinants, applied recursively, makes this clear
5. The determinant will be non-zero iff all diagonal elements are non-zero, by previous result
6. The eigenvalues are given by the roots of $|\mathrm{A} - \lambda \mathrm{I}|$. This matrix is lower triangular, so the determinant is the product of diagonal elements.

□

2.2.2 Unit triangular matrices

Definition 8 *A matrix* L *is* unit lower triangular *if it is lower triangular, and has the additional property that all elements along the leading diagonal are 1, that is,* $l_{ii} = 1, \ \forall i$.

In addition to all of the properties of lower triangular matrices, unit lower triangular matrices have some additional useful properties.

Proposition 12
1. *The product of two unit lower triangular matrices is unit lower triangular*
2. *The inverse of a unit lower triangular matrix is unit lower triangular*
3. *The determinant of a unit lower triangular matrix is 1, and hence all unit lower triangular matrices are invertible*
4. *The eigenvalues of a unit lower triangular matrix are all equal to 1*

Proof

1. Left as an exercise
2. Again, clear by thinking about Gaussian elimination and row operations
3. The determinant is the product of the diagonal elements
4. $|\mathrm{L} - \lambda \mathrm{I}| = (1 - \lambda)^n$.

□

In summary, triangular matrices form a group under *addition*, and unit triangular matrices form a group under *multiplication*.

It is clear that a non-singular lower triangular matrix A can be factorised in the form

$$\mathsf{A} = \mathsf{DL}$$

where D is diagonal and L is unit lower triangular, by choosing $\mathsf{D} = \operatorname{diag}\{a_{11}, a_{22}, \ldots, a_{nn}\}$ and $\mathsf{L} = \mathsf{D}^{-1}\mathsf{A}$.

Example

Write the 2×2 lower triangular matrix

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

in the form $\mathsf{A} = \mathsf{DL}$, where D is diagonal and L is unit lower triangular.

If we put

$$\mathsf{D} = \operatorname{diag}\{2, 4\}, \quad \mathsf{D}^{-1} = \operatorname{diag}\{1/2, 1/4\}$$

we get

$$\mathsf{L} = \mathsf{D}^{-1}\mathsf{A} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/4 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1/4 & 1 \end{pmatrix}$$

and we are done. The resulting factorisation is

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

2.2.3 Forward and backward substitution

One of the main reasons for thinking specifically about triangular matrices is that triangular linear systems turn out to be rather easy to solve. Consider the lower triangular equation

$$\mathsf{L}\boldsymbol{x} = \boldsymbol{b},$$

where $n \times n$ L and n-dimensional $\boldsymbol{b}$ are given, and a solution for n-dimensional $\boldsymbol{x}$ is required. We will assume that L is invertible (no zeros on the diagonal), so that the solution is unique. If we re-write the equation in component form, a simple solution strategy becomes apparent:

$$\begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

We can see that the first equation is just

$$l_{11}x_1 = b_1 \Rightarrow x_1 = b_1/l_{11}.$$

But now we can use x_1 to solve the second equation as

$$l_{21}x_1 + l_{22}x_2 = b_2 \Rightarrow x_2 = (b_2 - l_{21}x_1)/l_{22}.$$

In general we can solve for x_i in terms of $x_1, \ldots, x_{i-1}$ as

$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j \right) \Big/ l_{ii}.$$

Iterative construction of the solution in this way is known as *forward substitution*. Clearly the algorithm would fail if any $l_{ii} = 0$, but this corresponds to the rank degenerate case. For the invertible case we are considering, the algorithm provides a simple and efficient method for computing a solution.

Example: forward substitution

Solve the triangular system

$$\begin{pmatrix} 2 & 0 \\ 1 & 4 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

for $\boldsymbol{x}$.

$$\begin{pmatrix} 2 & 0 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix},$$

so, starting with the first equation we have

$$2x_1 = 4 \Rightarrow x_1 = 2.$$

The second equation can then be solved as

$$x_1 + 4x_2 = 6 \Rightarrow x_2 = \frac{6 - x_1}{4} = \frac{6 - 2}{4} = 1,$$

so our solution is $\boldsymbol{x} = (2, 1)^\mathsf{T}$.

Example: forward solve using R

Use R to solve the system

$$\begin{pmatrix} 2 & 0 & 0 \\ 1 & 3 & 0 \\ 2 & 3 & 4 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 6 \\ 9 \\ 28 \end{pmatrix}$$

for $\boldsymbol{x}$.

This is easily accomplished as shown in the following R session

```
> L=matrix(c(2,0,0,1,3,0,2,3,4),ncol=3,byrow=TRUE)
> L
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    1    3    0
[3,]    2    3    4
> forwardsolve(L,c(6,9,28))
[1] 3 2 4
>
```

The solution is therefore given by $\boldsymbol{x} = (3, 2, 4)^\mathsf{T}$.

Backward substitution

The solution of upper triangular systems is analogous, but looks a little different, so we will consider it explicitly. Suppose that U is an upper triangular matrix, and we want to solve the system

$$\mathsf{U}\boldsymbol{x} = \boldsymbol{b}$$

for $\boldsymbol{x}$. Again, we write out the system in component form:

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

We now see that system is easiest to solve starting with the last equation

$$u_{nn}x_n = b_n \Rightarrow x_n = b_n/u_{nn},$$

then

$$u_{n-1,n-1}x_{n-1} + u_{(n-1),n}x_n = b_{n-1} \Rightarrow x_{n-1} = (b_{n-1} - u_{(n-1),n}x_n)/u_{n-1,n-1},$$

and so on, with x_i given by

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \Big/ u_{ii}.$$

This procedure is known as *backward substitution*, for obvious reasons.

Example: backward substitution

Solve the upper triangular system

$$\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

for $\boldsymbol{x}$.

$$\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix},$$

so, starting with the second (last) equation, we have

$$3x_2 = 6 \Rightarrow x_2 = 2.$$

The first equation can then be solved as

$$x_1 + 2x_2 = 3 \Rightarrow x_1 = 3 - 2x_2 = 3 - 4 = -1,$$

so our solution is $\boldsymbol{x} = (-1, 2)^{\mathsf{T}}$.

Example: backward solve using R

Solve the upper triangular system

$$\begin{pmatrix} 2 & 2 & 3 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 8 \\ 6 \\ -6 \end{pmatrix}$$

for $\boldsymbol{x}$ using **R**.

This is easily accomplished as the following **R** session shows

```
> R=matrix(c(2,2,3,0,2,-1,0,0,3),ncol=3,byrow=TRUE)
> R
     [,1] [,2] [,3]
[1,]    2    2    3
[2,]    0    2   -1
[3,]    0    0    3
> backsolve(R,c(8,6,-6))
[1]  5  2 -2
```

So the solution is $\boldsymbol{x} = (5, 2, -2)^{\mathsf{T}}$.

2.3 Triangular matrix decompositions

2.3.1 LU decomposition

We now know how to solve triangular systems of equations, but most linear systems of interest do not start out in triangular form. However, as already outlined, if we can factorise or decompose a system into triangular components, we can solve a sequence of triangular systems in order to get a solution to our original problem. The problem we will consider now is the following.

Proposition 13 (LU decomposition) *Given a non-singular $n \times n$ matrix A, we can find unit lower triangular L and upper triangular U such that*

$$\mathsf{A} = \mathsf{LU}.$$

Clearly then, since $|\mathsf{A}| = |\mathsf{LU}| = |\mathsf{L}||\mathsf{U}| = |\mathsf{U}|$, we have $|\mathsf{A}| = \prod_{i=1}^{n} u_{ii}$, and this is an efficient way to compute the determinant in the rare cases where it is really needed.

The factorisation is just a matrix representation of the procedure of using Gaussian elimination to zero out the lower triangle of A to create U. L represents the (inverse of the) row operations required to achieve this. We will not give a formal derivation of the construction, but instead give a rough outline. First we note that the row operations used in Gaussian elimination can be represented by matrices. So the row operation which adds λ times row i to row j can be represented by the matrix

$$\mathsf{M}_{ij}(\lambda) = \mathrm{I} + \lambda \boldsymbol{e}_j \boldsymbol{e}_i^{\mathsf{T}},$$

and λ is known as the *multiplier* of the operation. The row operations needed to zero out the lower triangle will all have $i < j$, so $\mathsf{M}_{ij}(\lambda)$ will be unit lower triangular, and therefore

the product of all of the row operations will also be unit lower triangular. Row operations are easily inverted, as

$$\mathsf{M}_{ij}(\lambda)^{-1} = \mathrm{I} - \lambda \boldsymbol{e}_j \boldsymbol{e}_i^{\mathsf{T}}.$$

This can be verified by multiplication, noting particularly how the final product term vanishes. It will require $n(n-1)/2$ such row operations to zero out the lower triangle. If we number them sequentially, so that M_1 is the row operation to zero out a_{21}, M_2 is the row operation to zero out a_{31}, etc., then the matrix representing the complete set of row operations is

$$\mathsf{M} = \mathsf{M}_{n(n-1)/2} \cdots \mathsf{M}_2 \mathsf{M}_1.$$

Then if we denote the upper triangle that we are left with by U we have

$$\mathsf{MA} = \mathsf{U} \Rightarrow \mathsf{A} = \mathsf{M}^{-1}\mathsf{U} = \mathsf{LU}$$

where $\mathsf{L} = \mathsf{M}^{-1}$. Now

$$\mathsf{L} = {\mathsf{M}_1}^{-1} {\mathsf{M}_2}^{-1} \cdots {\mathsf{M}_{n(n-1)/2}}^{-1},$$

and the individual terms of this product are all trivial to write down. Remarkably, the unit lower triangular product is also trivial to compute. It turns out to be simply the unit lower triangular matrix whose off-diagonal elements are (minus) the row operation multipliers used to zero out that entry. Note that this is due to all of the product terms cancelling, and this in turn depends crucially on the order in which the operations are applied. The product terms cancel in the inverse L and not in M because in the inverse, no new row operation is introduced which uses a row that has already been modified as the "source" for a new row operation. The factorisation procedure is best illustrated by example.

Example

Let A be the 3×3 matrix

$$\mathsf{A} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix}.$$

Find the LU decomposition of A. Start with

$$\mathsf{A} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix}$$

and zero out the first column. The required multipliers are -2 and -3 respectively, so we record their "inverses", 2 and 3, in the locations we have zeroed out:

$$\begin{pmatrix} 1 & 4 & 7 \\ \mathbf{2} & -3 & -6 \\ \mathbf{3} & -6 & -11 \end{pmatrix}.$$

Now we zero out the second column with the multiplier -2, and so we record the number 2 in the position we zero out:

$$\begin{pmatrix} 1 & 4 & 7 \\ \mathbf{2} & -3 & -6 \\ \mathbf{3} & \mathbf{2} & 1 \end{pmatrix}.$$

That's it! The LU factorisation is

$$\mathsf{A} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{pmatrix}.$$

2.3.2 LDM^T decomposition

The LU factorisation has U upper triangular and L *unit* lower triangular. We have already seen that we can factorise non-singular U as $\mathsf{U} = \mathsf{DR}$ where D is diagonal and R is unit upper triangular, and this leads directly to the LDM^T decomposition of a non-singular matrix A as

$$\mathsf{A} = \mathsf{LDM}^\mathsf{T},$$

where L and $\mathsf{M} = \mathsf{R}^\mathsf{T}$ are unit lower triangular, and D is diagonal. It is clear from this decomposition that $|\mathsf{A}| = |\mathsf{LDM}^\mathsf{T}| = |\mathsf{L}||\mathsf{D}||\mathsf{M}^\mathsf{T}| = |\mathsf{D}| = \prod_{i=1}^{n} d_{ii}$. We can obviously construct this decomposition from the LU factorisation of A. It turns out that there is also a direct construction which is slightly more efficient, but this is not especially relevant in the context of this course.

2.3.3 LDL^T decomposition

In many statistical applications, the matrix A we want to factorise is *symmetric* (for example, it is often a covariance or correlation matrix). In this case it turns out that the LDM^T decomposition takes the form LDL^T, that is $\mathsf{L} = \mathsf{M}$. In some sense this is obvious, and is certainly intuitive, as it has the same form as the LDM^T decomposition, and is clearly symmetric. However, it is not *completely* obvious that the LDM^T decomposition *has* to be of this form, but it has.

Proposition 14 *The* LDM^T *decomposition of a non-singular symmetric matrix* A *has* $\mathsf{L} = \mathsf{M}$*, giving the decomposition*

$$A = \mathsf{LDL}^\mathsf{T},$$

where L *is unit lower triangular and* D *is diagonal.*

Proof

Start with $\mathsf{A} = \mathsf{LDM}^\mathsf{T}$, and first post-multiply by $\mathsf{M}^{-\mathsf{T}}$ and then pre-multiply by M^{-1} to get

$$\mathsf{M}^{-1}\mathsf{A}\mathsf{M}^{-\mathsf{T}} = \mathsf{M}^{-1}\mathsf{LD}.$$

The LHS of this equation is clearly symmetric, and the RHS is clearly lower triangular.

Therefore both sides must be diagonal.

But if $\mathsf{M}^{-1}\mathsf{LD}$ is diagonal, $\mathsf{M}^{-1}\mathsf{L}$ must also be.

But this is also clearly unit lower triangular, so must be the identity.

That is, $\mathsf{M}^{-1}\mathsf{L} = \mathrm{I}$, and hence $\mathsf{M} = \mathsf{L}$. □

The symmetry in this factorisation opens up the possibility of developing more efficient algorithms for constructing the decomposition, rather than starting with a basic LU decomposition. Rather than examine such strategies for this particular factorisation, we will do so in the context of a very closely related decomposition, usually known as the *Cholesky decomposition*.

2.3.4 The Cholesky decomposition

If the symmetric matrix A is not only non-singular, but also *positive definite*, then we can construct another factorisation that is very widely used in statistical applications. We will first construct it from the LDL^T decomposition, but then derive a more direct and efficient algorithm for its construction. We first need a couple of simple results whose proofs are left as exercises.

Proposition 15 *If*

$$\mathsf{A} = \mathsf{LDL}^\mathsf{T}$$

is the LDL^T *decomposition of symmetric positive definite* A*, then*

1. D *is positive definite, and*
2. *the diagonal elements of* D *are all strictly positive.*

Now since the elements of D are all strictly positive, we can define $\mathsf{D}^{1/2}$ to be the matrix whose diagonal elements are the square root of the corresponding elements of D, and then we have

Proposition 16 (Cholesky decomposition) *A symmetric positive definite matrix* A *can be decomposed as*

$$\mathsf{A} = \mathsf{GG}^\mathsf{T},$$

where G *is a lower triangular matrix.*

Proof

Start with the decomposition $\mathsf{A} = \mathsf{LDL}^\mathsf{T}$ and put $\mathsf{G} = \mathsf{LD}^{1/2}$. □

The Cholesky decomposition has many important applications in multivariate statistics and data analysis, and we will examine a few of these shortly, and more later in the course. Note that if we are interested in the determinant of A, we have $|\mathsf{A}| = |\mathsf{G}||\mathsf{G}^\mathsf{T}| = |\mathsf{G}|^2$, and $|\mathsf{G}| = \prod_{i=1}^n g_{ii}$, so this provides an efficient way to compute the determinant of a symmetric positive definite matrix. Obviously we can construct the decomposition starting from a basic LU factorisation, but that is a very inefficient method. It turns out to be quite straightforward to derive a fairly efficient algorithm for its construction from first principles.

Direct construction

We can write out the Cholesky decomposition in component form as

$$\begin{pmatrix} g_{11} & 0 & \cdots & 0 \\ g_{21} & g_{22} & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ g_{n1} & g_{n2} & \cdots & g_{nn} \end{pmatrix} \begin{pmatrix} g_{11} & g_{21} & \cdots & g_{n1} \\ 0 & g_{22} & \ddots & g_{n2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & g_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}.$$

Somewhat analogous to the technique of forward substitution, we can work through the equations one by one, starting from the top left, working row by row through the non-zero

elements of G. The first few equations can be solved as follows:

$$\begin{aligned} g_{11}^2 = a_{11} &\Rightarrow g_{11} = \sqrt{a_{11}} \\ g_{21}g_{11} = a_{21} &\Rightarrow g_{21} = \frac{a_{21}}{g_{11}} \\ g_{21}^2 + g_{22}^2 = a_{22} &\Rightarrow g_{22} = \sqrt{a_{22} - g_{21}^2} \\ g_{31}g_{11} = a_{31} &\Rightarrow g_{31} = \frac{a_{31}}{g_{11}} \\ g_{31}g_{21} + g_{32}g_{22} = a_{32} &\Rightarrow g_{32} = \frac{a_{32} - g_{31}g_{21}}{g_{22}}, \end{aligned}$$

and so on. We see that if we work through the equations in order, we know enough at each stage to solve explicitly for the next unknown. In general we have

$$g_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} g_{ik}^2}, \qquad g_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} g_{ik}g_{jk}}{g_{ii}}, \ i > j.$$

This algorithm is a bit faster than an LU decomposition as it exploits the symmetry in A. It is also a bit more stable for the same reason. However, it only works reliably for matrices that are strictly positive definite. It can be extended to the positive semi-definite case using *pivoting* (beyond the scope of this course), but there are numerical stability issues there, and another decomposition (such as the SVD, to be covered later) may be preferable in that case.

Example

Compute the Cholesky factorisation of the 3×3 matrix

$$\mathsf{A} = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 0 \\ 2 & 0 & 14 \end{pmatrix}.$$

For a small problem like this, it is simplest to write out the matrix relations in full as

$$\begin{pmatrix} g_{11} & 0 & 0 \\ g_{21} & g_{22} & 0 \\ g_{31} & g_{32} & g_{33} \end{pmatrix} \begin{pmatrix} g_{11} & g_{21} & g_{31} \\ 0 & g_{22} & g_{32} \\ 0 & 0 & g_{33} \end{pmatrix} = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 0 \\ 2 & 0 & 14 \end{pmatrix}.$$

We now solve the equations one at a time as

$$\begin{aligned}
g_{11}^2 = 1 &\Rightarrow g_{11} = 1 \\
g_{21}g_{11} = -1 &\Rightarrow g_{21} = \frac{-1}{g_{11}} = \frac{-1}{1} = -1 \\
g_{21}^2 + g_{22}^2 = 5 &\Rightarrow g_{22} = \sqrt{5 - g_{21}^2} = \sqrt{5-1} = 2 \\
g_{31}g_{11} = 2 &\Rightarrow g_{31} = \frac{2}{g_{11}} = \frac{2}{1} = 2 \\
g_{31}g_{21} + g_{32}g_{22} = 0 &\Rightarrow g_{32} = \frac{-g_{31}g_{21}}{g_{22}} = \frac{2\times 1}{2} = 1 \\
g_{31}^2 + g_{32}^2 + g_{33}^2 = 14 &\Rightarrow g_{33} = \sqrt{14 - g_{31}^2 - g_{32}^2} = \sqrt{14-4-1} = 3.
\end{aligned}$$

We therefore have $\mathsf{A} = \mathsf{G}\mathsf{G}^\mathsf{T}$, where

$$\mathsf{G} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 1 & 3 \end{pmatrix}.$$

Having computed the decomposition, it is always a good idea to check the solution as

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 1 & 3 \end{pmatrix} \begin{pmatrix} 1 & -1 & 2 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 0 \\ 2 & 0 & 14 \end{pmatrix}.$$

Having computed the Cholesky factorisation by hand using a relatively efficient method, it is instructive to, just once, construct the factorisation starting from the LU decomposition. Let's begin by directly constructing the LU decomposition of A as

$$\begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 0 \\ 2 & 0 & 14 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -1 & 2 \\ \mathbf{-1} & 4 & 2 \\ \mathbf{2} & 2 & 10 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -1 & 2 \\ \mathbf{-1} & 4 & 2 \\ \mathbf{2} & \mathbf{1/2} & 9 \end{pmatrix},$$

giving the LU decomposition

$$\begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 0 \\ 2 & 0 & 14 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1/2 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 2 \\ 0 & 4 & 2 \\ 0 & 0 & 9 \end{pmatrix}.$$

We now construct the LDM^T decomposition by extracting the diagonal of U and transposing the result to get

$$\begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 0 \\ 2 & 0 & 14 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1/2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2^2 & 0 \\ 0 & 0 & 3^2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1/2 & 1 \end{pmatrix}^\mathsf{T}.$$

At this point we note that, due to the symmetry of A, we have $\mathsf{L} = \mathsf{M}$, and so this is in fact the LDL^T decomposition. Since the elements of D are all positive, A must be strictly positive definite, and we can construct the Cholesky factor as $\mathsf{L}\mathsf{D}^{1/2}$ to get

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1/2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 1 & 3 \end{pmatrix},$$

as before.

Example: Galaxy data

We can use R to construct the Cholesky decomposition of the sample variance matrix for the Galaxy data, and confirm the solution, as illustrated in the following session

```
> v=var(galaxy)
> v
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west          451.46551
north.south      -1929.95131
angle               37.21646
radial.position   1637.78301
velocity          8886.47724
> c=chol(v)
> c
                east.west north.south      angle radial.position
east.west         12.0275    -2.71710 -1.7879038     21.94441872
north.south        0.0000    22.72591  0.9460287     -8.89575759
angle              0.0000     0.00000 38.1907749     -0.03564306
radial.position    0.0000     0.00000  0.0000000     10.46597448
velocity           0.0000     0.00000  0.0000000      0.00000000
                  velocity
east.west         37.536090
north.south      -80.435144
angle              4.724212
radial.position    9.431724
velocity          29.940461
> g=t(c)
> g
                east.west north.south       angle radial.position
east.west       12.027505   0.0000000  0.00000000        0.000000
north.south     -2.717100  22.7259122  0.00000000        0.000000
angle           -1.787904   0.9460287 38.19077490        0.000000
radial.position 21.944419  -8.8957576 -0.03564306       10.465974
velocity        37.536090 -80.4351441  4.72421244        9.431724
                velocity
east.west        0.00000
north.south      0.00000
angle            0.00000
radial.position  0.00000
velocity        29.94046
> g%*%c
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
```

```
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west         451.46551
north.south     -1929.95131
angle              37.21646
radial.position  1637.78301
velocity         8886.47724
>
```

*Note that **R** returns the upper triangle of the Cholesky factorisation*, which can easily be transposed to give the lower triangle if required. We next consider why it might be useful to be able to compute the Cholesky factorisation of a variance matrix.

Transformations from and to standardised variables

Recall that one of the motivations for thinking about matrix factorisations was to be able to find a matrix G such that

$$\Sigma = \mathsf{G}\mathsf{G}^{\mathsf{T}}. \tag{2.1}$$

Clearly the Cholesky decomposition provides an efficient solution to this problem for the case of a symmetric positive definite Σ. Before looking at applications of this decomposition, it is worth thinking briefly about the existence and uniqueness of solutions to this problem. Since Σ is symmetric, there are only $n(n+1)/2$ degrees of freedom. If we allowed an arbitrary choice of G, there would be n^2 degrees of freedom, and so it seems likely that there would be many different G which satisfy the (2.1). This is indeed the case. By restricting ourselves to looking for a lower triangular matrix, we have ensured that we have only $n(n+1)/2$ degrees of freedom, and this leads to a unique solution (in the positive definite case). Clearly another obvious strategy would be to look for a *symmetric* matrix satisfying (2.1). This is also possible, and also leads to an essentially unique solution. This matrix is known as the *symmetric square root*, as it is a symmetric matrix satisfying the equation

$$\Sigma = \mathsf{G}^2,$$

in addition to (2.1). Note that many mathematicians would not regard the Cholesky factor as a "true" square root, since it does not have this property. However, since (2.1) is the most commonly encountered requirement for a "square root" in statistical applications, most statisticians use (2.1) as the definition of a square root matrix. This could potentially lead to confusion, so care must be taken when reading literature relating to matrix square roots.

We now consider the problem stated in Chapter 1 of how to transform standard random quantities to give a vector of random quantities with a pre-specified variance matrix. Suppose that we wish to simulate random quantities with p-dimensional mean vector $\boldsymbol{\mu}$ and $p \times p$ variance matrix Σ. Let $\boldsymbol{Z}$ be a vector of independent random quantities each with mean zero and variance one. Then the random quantity

$$\boldsymbol{X} = \boldsymbol{\mu} + \mathsf{G}\boldsymbol{Z},$$

where G is a matrix satisfying $\Sigma = \mathsf{G}\mathsf{G}^{\mathsf{T}}$, has the required mean and variance.

We can also invert this affine transformation, to get

$$\boldsymbol{Z} = \mathsf{G}^{-1}(\boldsymbol{X} - \boldsymbol{\mu}),$$

a method for transforming quantities with a given mean and variance to standard uncorrelated form. This kind of *standardisation* transformation also has many applications, and we shall look at the symmetric version of this kind of transformation later. Note also that the above equation does not require us to compute the inverse of G. We are merely using the equation as shorthand for solving the linear system

$$\mathsf{G}\boldsymbol{Z} = \boldsymbol{X} - \boldsymbol{\mu}$$

for $\boldsymbol{Z}$ using forward substitution.

Example: standardising the Galaxy data

The following R session shows how to standardise the Galaxy data

```
> W=sweep(galaxy,2,colMeans(galaxy))
> g=chol(var(galaxy))
> Z=as.matrix(W)%*%solve(g)
> var(Z)
                   east.west   north.south         angle
east.west        1.000000e+00  9.870280e-18  4.921924e-18
north.south      9.870280e-18  1.000000e+00 -7.548842e-18
angle            4.921924e-18 -7.548842e-18  1.000000e+00
radial.position -6.091896e-16 -3.758545e-17 -3.116896e-17
velocity         2.160761e-17 -1.472481e-16 -1.313282e-17
                radial.position      velocity
east.west         -6.091896e-16  2.160761e-17
north.south       -3.758545e-17 -1.472481e-16
angle             -3.116896e-17 -1.313282e-17
radial.position    1.000000e+00  1.231192e-16
velocity           1.231192e-16  1.000000e+00
> colMeans(Z)
      east.west     north.south           angle radial.position
  -1.353205e-17   -1.512512e-17   -6.784303e-17   -7.906418e-17
       velocity
  -3.376896e-15
>
```

Note that here, to keep things simple, I did actually compute the inverse of the Cholesky factor (using `solve()`), and then used this to transform the data matrix. As there are many more rows than columns in the data matrix, this isn't actually terrible way to do the transformation, in this particular case, but we will see a much better method in the next section. We will examine later why it might be useful to standardise data in this way.

2.4 Other matrix factorisations

2.4.1 QR factorisation

Not all useful matrix factorisations use triangular matrices. Orthogonal matrices also turn out to be very useful (they are even easier to invert than triangular matrices). We will start by looking briefly at a factorisation that is part orthogonal and part triangular. The QR factorisation can be applied to an $n \times p$ matrix A, where (typically) $n \geq p$. There are several variants of this factorisation, but the variant most commonly used in practice, and the version implemented in **R**, is the so-called "thin" QR factorisation where A is factorised as

$$\mathrm{A} = \mathrm{QR},$$

where Q is an $n \times p$ matrix with orthogonal columns ($\mathrm{Q}^{\mathrm{T}}\mathrm{Q} = \mathrm{I}$, but typically $\mathrm{QQ}^{\mathrm{T}} \neq \mathrm{I}$), and R is $p \times p$ and upper triangular. We will not dwell extensively on the construction of this decomposition, but it is essentially a matrix representation of the Gram-Schmidt orthonormalisation of the columns of A. Column operations are represented by post-multiplication by a matrix. The Gram-Schmidt operations are represented by upper triangular matrices, so the combined set of operations is represented by an upper triangular matrix, C. We then have

$$\mathrm{AC} = \mathrm{Q}$$

but then

$$\mathrm{A} = \mathrm{QR},$$

where $\mathrm{R} = \mathrm{C}^{-1}$. There are various variations on the precise way that this decomposition is computed, but these will not concern us. This decomposition has many potential applications in statistical problems.

We can compute and manipulate QR factorisations using **R**.

```
> A=matrix(c(1,2,3,4,5,6),ncol=2)
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> myQR=qr(A)
> qr.Q(myQR)
           [,1]       [,2]
[1,] -0.2672612  0.8728716
[2,] -0.5345225  0.2182179
[3,] -0.8017837 -0.4364358
> qr.R(myQR)
          [,1]      [,2]
[1,] -3.741657 -8.552360
[2,]  0.000000  1.963961
> qr.Q(myQR)%*%qr.R(myQR)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
> t(qr.Q(myQR))%*%qr.Q(myQR)
              [,1]          [,2]
[1,]  1.000000e+00 -5.144539e-17
[2,] -5.144539e-17  1.000000e+00
```

In the case where A is square ($n = p$), we have

$$|\mathsf{A}| = |\mathsf{QR}| = |\mathsf{Q}||\mathsf{R}| = |\mathsf{R}| = \prod_{i=1}^{n} r_{ii},$$

and this represents an efficient way to compute the determinant.

Uniqueness and relation to Cholesky factorisation

It is clear that we can construct a "thin" QR factorisation via Gram-Schmidt orthonormalisation of the columns of A, but it is natural to wonder whether such a factorisation is unique? In the full rank case it is, and understanding why sheds light on its relationship with the Cholesky factorisation. Suppose that

$$\mathsf{A} = \mathsf{QR}$$

is the QR factorisation of A, and consider

$$\begin{aligned} \mathsf{A}^\mathsf{T}\mathsf{A} &= (\mathsf{QR})^\mathsf{T}\mathsf{QR} \\ &= \mathsf{R}^\mathsf{T}\mathsf{Q}^\mathsf{T}\mathsf{QR} \\ &= \mathsf{R}^\mathsf{T}\mathsf{R}, \end{aligned}$$

where R is upper triangular. It is therefore clear that $\mathsf{R} = \mathsf{G}^\mathsf{T}$, where G is the Cholesky factor for $\mathsf{A}^\mathsf{T}\mathsf{A}$. We know that the Cholesky factor is unique (in the positive definite case), and therefore so is R, and we can construct Q via

$$\mathsf{Q} = \mathsf{AR}^{-1}$$

and is also unique. Note that this construction is almost identical to the way we standardised a data matrix in the previous Galaxy data example. Note that this construction gives us insight into the nature of the QR factorisation, but does not represent an efficient method for its construction. Variations on Gram-Schmidt are much faster and more numerically stable.

Using the QR decomposition for standardisation

The QR factorisation of a centered data matrix therefore gives us a much more efficient way of standardising a data matrix, and also a very efficient way of computing the Cholesky factor of the sample variance matrix. Let's just spell out the details to be absolutely clear. Starting from an $n \times p$ data matrix X, we form the centered data matrix

$$\mathsf{W} = \mathsf{H}_n\mathsf{X},$$

where, again, we are just using $\mathsf{H}_n\mathsf{X}$ as shorthand for $\mathsf{X} - \mathbf{1}_n\bar{\boldsymbol{x}}^\mathsf{T}$ — we are not suggesting actually constructing the centering matrix H_n.[†] Next construct the QR decomposition of W as

$$\mathsf{W} = \mathsf{QR}.$$

Putting

$$\mathsf{G} = \frac{1}{\sqrt{n-1}}\mathsf{R}^\mathsf{T}$$

gives lower triangular G as the Cholesky factor of the sample variance matrix for X. We can verify this as follows:

$$\begin{aligned}\mathsf{S} &= \frac{1}{n-1}\mathsf{W}^\mathsf{T}\mathsf{W} \\ &= \frac{1}{n-1}(\mathsf{QR})^\mathsf{T}\mathsf{QR} \\ &= \frac{1}{n-1}\mathsf{R}^\mathsf{T}\mathsf{R} \\ &= \mathsf{G}\mathsf{G}^\mathsf{T}.\end{aligned}$$

Similarly, if we set

$$\mathsf{Z} = \sqrt{n-1}\mathsf{Q},$$

then Z has mean zero and variance the identity, since

$$\begin{aligned}\mathsf{S}_\mathsf{Z} &= \frac{1}{n-1}\mathsf{Z}^\mathsf{T}\mathsf{Z} \\ &= \mathsf{Q}^\mathsf{T}\mathsf{Q} \\ &= \mathrm{I}_p\,.\end{aligned}$$

Example: Galaxy data

We can use the QR factorisation to standardise the Galaxy data analysed previously as illustrated in the following **R** session:

```
> myQR=qr(W)
> qr.R(myQR)/sqrt(322)
  east.west north.south       angle radial.position   velocity
3  -12.0275     2.71710   1.7879038    -21.94441872 -37.536090
4    0.0000    22.72591   0.9460287     -8.89575759 -80.435144
5    0.0000     0.00000 -38.1907749      0.03564306  -4.724212
6    0.0000     0.00000   0.0000000    -10.46597448  -9.431724
7    0.0000     0.00000   0.0000000      0.00000000 -29.940461
> g
               east.west north.south      angle radial.position
east.west        12.0275    -2.71710 -1.7879038     21.94441872
north.south       0.0000    22.72591  0.9460287     -8.89575759
```

[†]This is analogous to using $\mathsf{A}^{-1}\boldsymbol{b}$ as shorthand for solving the linear system $\mathsf{A}\boldsymbol{x} = \boldsymbol{b}$, and not actually suggesting constructing the inverse of A.

```
angle              0.0000     0.00000 38.1907749     -0.03564306
radial.position    0.0000     0.00000  0.0000000     10.46597448
velocity           0.0000     0.00000  0.0000000      0.00000000
                  velocity
east.west         37.536090
north.south      -80.435144
angle              4.724212
radial.position    9.431724
velocity          29.940461
> head(qr.Q(myQR)*sqrt(322))
           [,1]       [,2]       [,3]       [,4]        [,5]
[1,] -0.7312537 -1.659224 -0.6411756 -0.8750822 -0.10633537
[2,] -0.6898643 -1.565365 -0.6369129 -0.8218676  0.24018028
[3,] -0.6484749 -1.471507 -0.6326503 -0.7686530 -0.08129645
[4,] -0.6070855 -1.377648 -0.6283877 -0.7154384 -0.70337004
[5,] -0.5656961 -1.283789 -0.6241250 -0.6622238 -0.75765010
[6,] -0.5243067 -1.189931 -0.6198624 -0.6090092 -0.91212873
> head(Z)
  east.west north.south     angle radial.position    velocity
3 0.7312537   -1.659224 0.6411756       0.8750822  0.10633537
4 0.6898643   -1.565365 0.6369129       0.8218676 -0.24018028
5 0.6484749   -1.471507 0.6326503       0.7686530  0.08129645
6 0.6070855   -1.377648 0.6283877       0.7154384  0.70337004
7 0.5656961   -1.283789 0.6241250       0.6622238  0.75765010
8 0.5243067   -1.189931 0.6198624       0.6090092  0.91212873
>
```

From this we see that the results of using the QR factorisation are identical to the results obtained earlier, except for arbitrary sign issues, which are usually unimportant, and can be easily corrected if necessary.

2.4.2 Sign issues and uniqueness

As we have just seen, when working with matrix factorisations results are often unique only up to the sign of rows or columns of certain matrices. It is important to understand what is going on here, and how it is often possible to choose a convention for the signs in order to make a factorisation really unique. We begin by thinking about how to flip the sign of a particular row or column of a matrix. Consider the $p \times p$ matrix F_i which is diagonal, and has a 1 in every diagonal position except the ith, where it has a -1. We can write this as

$$\mathsf{F}_i = \mathrm{I}_p - 2\boldsymbol{e}_i\boldsymbol{e}_i^{\mathsf{T}}.$$

It is clear that pre-multiplying a $p \times m$ matrix by F_i has the effect of flipping the sign of the ith row. Similarly, post-multiplying a $n \times p$ matrix by F_i flips the sign of the ith column (if this isn't clear, just try it!). So, this matrix can be used to flip the sign of arbitrary rows and columns of matrices.

Proposition 17

$$\mathsf{F}_i^2 = \mathrm{I}_p\,.$$

In other words, F_i *is its own inverse (and* F_i *is known as an involution).*

Proof

$$\begin{aligned}
\mathsf{F}_i^2 &= (\mathrm{I}_p - 2\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T})^2 \\
&= \mathrm{I}_p - 4\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T} + 4\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T}\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T} \\
&= \mathrm{I}_p - 4\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T} + 4\boldsymbol{e}_i(\boldsymbol{e}_i^\mathsf{T}\boldsymbol{e}_i)\boldsymbol{e}_i^\mathsf{T} \\
&= \mathrm{I}_p - 4\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T} + 4\boldsymbol{e}_i\boldsymbol{e}_i^\mathsf{T} \\
&= \mathrm{I}_p \,.
\end{aligned}$$

□

We can see the implications of this for the uniqueness of factorisations like the QR factorisation by considering

$$\begin{aligned}
\mathsf{A} &= \mathsf{QR} \\
&= \mathsf{Q}\,\mathrm{I}_p\,\mathsf{R} \\
&= \mathsf{QF}_i^2\mathsf{R} \\
&= (\mathsf{QF}_i)(\mathsf{F}_i\mathsf{R}) \\
&= \mathsf{Q}^\star\mathsf{R}^\star,
\end{aligned}$$

where $\mathsf{Q}^\star = \mathsf{QF}_i$ and $\mathsf{R}^\star = \mathsf{F}_i\mathsf{R}$. Clearly $\mathsf{Q}^\star$ is just Q with the ith column flipped, and $\mathsf{R}^\star$ is just R with the ith row flipped. However, the point is that $\mathsf{Q}^\star\mathsf{R}^\star$ is also a valid QR factorisation of A. It is clear that $\mathsf{R}^\star$ is upper triangular. It should be equally clear that $\mathsf{Q}^\star$ has orthonormal columns, since flipping the sign of a column does not affect orthogonality. If it isn't clear, just consider

$$\begin{aligned}
\mathsf{Q}^{\star\mathsf{T}}\mathsf{Q}^\star &= (\mathsf{QF}_i)^\mathsf{T}\mathsf{QF}_i \\
&= \mathsf{F}_i\mathsf{Q}^\mathsf{T}\mathsf{QF}_i \\
&= \mathsf{F}_i\,\mathrm{I}_p\,\mathsf{F}_i \\
&= \mathsf{F}_i\mathsf{F}_i \\
&= \mathsf{F}_i^2 \\
&= \mathrm{I}_p \,.
\end{aligned}$$

Consequently, flipping the sign of a column of Q and the corresponding row of R leads to another valid QR factorisation. Clearly this can be repeated for multiple columns of Q and rows of R. To ensure that the R matrix of a QR factorisation corresponds precisely to a Cholesky factor as typically computed (with non-negative elements on the diagonal), it is necessary to flip the sign of any rows with negative elements on the diagonal, being careful to also flip the sign of the corresponding columns of Q.

Note that there are similar sign issues for other matrix factorisations we will consider, such as the spectral and singular value decompositions.

2.4.3 Least squares problems

As the previous section suggests, QR factorisations have many potential applications in statistics, but so called "least squares" problems are the classic example. First note how

we can use the QR factorisation to solve a linear system. Suppose that we want to solve

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$$

for $\boldsymbol{x}$. Begin by assuming we have a square ($p \times p$) full rank system (with the same number of equations as unknowns). If A is factorised as $\mathrm{A} = \mathrm{QR}$, we have

$$\begin{aligned} \mathrm{QR}\boldsymbol{x} &= \boldsymbol{b} \\ \Rightarrow \mathrm{R}\boldsymbol{x} &= \mathrm{Q}^{\mathsf{T}}\boldsymbol{b}, \end{aligned}$$

since $\mathrm{Q}^{\mathsf{T}}\mathrm{Q} = \mathrm{I}$, and then we can solve for $\boldsymbol{x}$ using backward substitution.

In the case of a full rank $n \times p$ matrix A (with $n > p$), we can follow the same procedure and obtain a solution for $\boldsymbol{x}$. However, in this case we have more equations than unknowns and the system is said to be overdetermined. Clearly in this case there may not be a p-dimensional $\boldsymbol{x}$ which satisfies all n equations. The connection with least squares problems is revealed by first realising that when a solution does exist, it is the $\boldsymbol{x}$ which makes

$$\|\mathrm{A}\boldsymbol{x} - \boldsymbol{b}\|_2 = 0.$$

When such a $\boldsymbol{x}$ does not exist, in some sense the best $\boldsymbol{x}$ we can find is the $\boldsymbol{x}$ which makes $\|\mathrm{A}\boldsymbol{x} - \boldsymbol{b}\|_2$ as small as possible. This is a least squares problem. In the context of statistics, we usually describe such problems in the language of regression analysis.

So let us now consider the classical multiple linear regression problem

$$\boldsymbol{y} = \mathrm{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

for $n \times p$ covariate matrix X. Given data $\boldsymbol{y}$ and X we want to find the $\boldsymbol{\beta}$ which minimises $\|\boldsymbol{y} - \mathrm{X}\boldsymbol{\beta}\|_2^2 = \|\boldsymbol{\varepsilon}\|_2^2 = \boldsymbol{\varepsilon}^{\mathsf{T}}\boldsymbol{\varepsilon}$. We can minimise wrt $\boldsymbol{\beta}$ by differentiating with respect to the vector $\boldsymbol{\beta}$ and equating to zero. We will examine this in detail in the next chapter, but doing this leads to the so-called "normal equations"

$$\mathrm{X}^{\mathsf{T}}\mathrm{X}\boldsymbol{\beta} = \mathrm{X}^{\mathsf{T}}\boldsymbol{y}.$$

Note that we cannot "cancel" the X^{T} on each side, as X is not square, and hence not invertible. Also note that this is a $p \times p$ system of equations for $\boldsymbol{\beta}$, so that as long as X is of full rank, there exists a unique solution. The mathematical solution is given by

$$\boldsymbol{\beta} = (\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}\boldsymbol{y},$$

but this represents a very expensive and numerically unstable method to actually compute the solution on a computer.

The classical method for solving the normal equations was via a Cholesky decomposition for $\mathrm{X}^{\mathsf{T}}\mathrm{X}$ and then backward and forward substitution. However, it turns out to be more efficient and numerically stable to use the QR factorisation of X, as

$$\begin{aligned} \mathrm{X}^{\mathsf{T}}\mathrm{X}\boldsymbol{\beta} &= \mathrm{X}^{\mathsf{T}}\boldsymbol{y} \\ \Rightarrow (\mathrm{QR})^{\mathsf{T}}\mathrm{QR}\boldsymbol{\beta} &= (\mathrm{QR})^{\mathsf{T}}\boldsymbol{y} \\ \Rightarrow \mathrm{R}^{\mathsf{T}}\mathrm{Q}^{\mathsf{T}}\mathrm{QR}\boldsymbol{\beta} &= \mathrm{R}^{\mathsf{T}}\mathrm{Q}^{\mathsf{T}}\boldsymbol{y} \\ \Rightarrow \mathrm{R}^{\mathsf{T}}\mathrm{R}\boldsymbol{\beta} &= \mathrm{R}^{\mathsf{T}}\mathrm{Q}^{\mathsf{T}}\boldsymbol{y}. \end{aligned}$$

That is, $\boldsymbol{\beta}$ is the solution of

$$\mathrm{R}\boldsymbol{\beta} = \mathrm{Q}^{\mathsf{T}}\boldsymbol{y},$$

obtained by backward substitution. This is the method of solution used by **R**'s `lm()` function. Note that here it was fine to "cancel" the R^{T} on each side, as R^{T} is invertible, so we could pre-multiply both sides of the equation by its inverse.

2.4.4 Spectral decomposition

Construction for real symmetric matrices

Recall that $\boldsymbol{v}$ is an eigenvector of an $n \times n$ square matrix A, and λ is its corresponding eigenvalue if

$$\mathsf{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

Clearly $(\mathsf{A} - \lambda\,\mathrm{I})\boldsymbol{v} = \mathbf{0}$, so $\boldsymbol{v}$ is in the null-space of $\mathsf{A} - \lambda\,\mathrm{I}$, which means that this must be singular, and so eigenvalues can be identified as roots of the characteristic equation

$$|\mathsf{A} - \lambda\,\mathrm{I}\,| = 0.$$

Since this is a polynomial in λ of degree n, there are n (not necessarily distinct) eigenvalues of A. In general, some of these eigenvalues will be complex, but not if A is symmetric.

Proposition 18 *The eigenvalues of a real symmetric matrix* A *are real.*

Proof

Suppose that λ is a complex eigenvalue of A. In this case the corresponding eigenvector $\boldsymbol{v}$ is also complex. By definition we have $\mathsf{A}\boldsymbol{v} = \lambda\boldsymbol{v}$, and taking complex conjugates we have

$$\mathsf{A}\bar{\boldsymbol{v}} = \bar{\lambda}\bar{\boldsymbol{v}}$$

so $\bar{\lambda}$ is also an eigenvalue with corresponding eigenvector $\bar{\boldsymbol{v}}$.[‡] But then

$$\begin{aligned} \bar{\boldsymbol{v}}^{\mathsf{T}}\mathsf{A}\boldsymbol{v} = \bar{\lambda}\boldsymbol{v}^{\dagger}\boldsymbol{v} &= \lambda\boldsymbol{v}^{\dagger}\boldsymbol{v} \\ \Rightarrow \bar{\lambda}\|\boldsymbol{v}\|^2 &= \lambda\|\boldsymbol{v}\|^2 \\ \Rightarrow \bar{\lambda} &= \lambda, \end{aligned}$$

and so $\lambda \in \mathbb{R}$. □

Since we can choose the magnitude of an eigenvector arbitrarily, we will assume that we always choose eigenvectors to have norm 1. That is $\boldsymbol{v}^{\mathsf{T}}\boldsymbol{v} = 1$.

Proposition 19 *The eigenvectors corresponding to distinct eigenvalues of a real symmetric matrix* A *are orthogonal.*

Proof

Suppose that $(\lambda_1, \boldsymbol{v}_1)$ and $(\lambda_2, \boldsymbol{v}_2)$ are both eigen-pairs of A. Then

$$\begin{aligned} \boldsymbol{v}_1{}^{\mathsf{T}}\mathsf{A}\boldsymbol{v}_2 = \lambda_2\boldsymbol{v}_1{}^{\mathsf{T}}\boldsymbol{v}_1 &= \lambda_1\boldsymbol{v}_1{}^{\mathsf{T}}\boldsymbol{v}_2 \\ \Rightarrow (\lambda_2 - \lambda_2)\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 &= 0, \end{aligned}$$

so either $\lambda_1 = \lambda_2$ or $\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 = 0$. □

[‡]Note that here we are using a bar ($^{-}$) to denote complex conjugation, rather than a sample mean. This proof is the only place in the whole course where complex numbers occur, and hence the only place where a bar does not represent a sample mean. Here we also use a dagger for conjugate transpose. That is, $\boldsymbol{v}^{\dagger} = \bar{\boldsymbol{v}}^{\mathsf{T}}$.

In fact, it turns out that repeated eigenvalues do not complicate the picture, since the eigen-space corresponding to repeated eigenvalues has the same dimension as the multiplicity of the root, and hence orthogonal eigenvectors may also be constructed for the repeated eigenvalues. Hence we can order the eigenvalues in decreasing order as

$$\mathsf{A}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \quad i = 1, 2, \ldots, n, \quad \text{where} \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n,$$

and by stacking the eigenvectors as columns of an $n \times n$ orthogonal matrix V we can re-write the above equations as

$$\mathsf{AV} = \mathsf{VD}$$

where $\mathsf{D} = \operatorname{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. This then leads to one of the most important results in linear algebra.

Proposition 20 (Spectral decomposition) *If* A *is a real symmetric* $n \times n$ *matrix, it can be factorised in the form*

$$\mathsf{A} = \mathsf{VDV}^\mathsf{T}$$

where V *is an orthogonal matrix whose columns are the eigenvectors of* A *and* $\mathsf{D} = \operatorname{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$*, where the* λ_i *are the ordered eigenvalues of* A *corresponding to the eigenvectors in the columns of* V*.*

We will often be interested in symmetric matrices which are positive semi-definite. These matrices clearly have non-negative eigenvalues, and similarly, strictly positive definite matrices have strictly positive eigenvalues.

Symmetric square root

The spectral decomposition can be used to create a symmetric square root matrix, since if

$$\mathsf{A} = \mathsf{VDV}^\mathsf{T}$$

then

$$\mathsf{G} = \mathsf{VD}^{1/2}\mathsf{V}^\mathsf{T}$$

satisfies $\mathsf{G}^\mathsf{T} = \mathsf{G}$ and $\mathsf{GG} = \mathsf{A}$, and hence also $\mathsf{GG}^\mathsf{T} = \mathsf{A}$. This symmetric square root is a "true" square root matrix, unlike the Cholesky triangle. It is also well defined for singular positive semi-definite matrices, where use of the Cholesky decomposition can be problematic. In the positive definite case, we can also directly construct its inverse as

$$\mathsf{G}^{-1} = \mathsf{VD}^{-1/2}\mathsf{V}^\mathsf{T},$$

which can be very useful in several contexts. The symmetry of this square root matrix is also desirable in some applications. However, the Cholesky factorisation is much faster than an eigen-solver, so in the many applications where the cheaper Cholesky factor will be adequate, it is usually to be preferred.

Example

Construct the spectral decomposition of the matrix

$$\mathsf{A} = \begin{pmatrix} 9 & 7 \\ 7 & 9 \end{pmatrix},$$

and use it to construct a symmetric square root, G, of A. Verify that $\mathsf{G}^2 = \mathsf{A}$. We first find roots of the characteristic equation via

$$\begin{aligned} |\mathsf{A} - \lambda \mathrm{I}| &= \begin{vmatrix} 9-\lambda & 7 \\ 7 & 9-\lambda \end{vmatrix} \\ &= (9-\lambda)^2 - 49 \\ &= (\lambda - 2)(\lambda - 16). \end{aligned}$$

So the ordered eigenvalues are $\lambda_1 = 16$, $\lambda_2 = 2$. Now

$$\mathsf{A} - \lambda_1 \mathrm{I} = \begin{pmatrix} -7 & 7 \\ 7 & -7 \end{pmatrix},$$

and so

$$\boldsymbol{v}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Similarly,

$$\mathsf{A} - \lambda_2 \mathrm{I} = \begin{pmatrix} 7 & 7 \\ 7 & 7 \end{pmatrix},$$

so

$$\boldsymbol{v}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Consequently, our spectral decomposition is $\mathsf{A} = \mathsf{V}\mathsf{D}\mathsf{V}^\mathsf{T}$, where

$$\mathsf{V} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \mathsf{D} = \begin{pmatrix} 16 & 0 \\ 0 & 2 \end{pmatrix}.$$

Now we have the spectral decomposition, it is straightforward to construct the symmetric square root as

$$\begin{aligned} \mathsf{G} &= \mathsf{V}\mathsf{D}^{1/2}\mathsf{V}^\mathsf{T} \\ &= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & \sqrt{2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 4 & 4 \\ \sqrt{2} & -\sqrt{2} \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 4+\sqrt{2} & 4-\sqrt{2} \\ 4-\sqrt{2} & 4+\sqrt{2} \end{pmatrix}. \end{aligned}$$

Our proposed square root is clearly symmetric, and we can verify that it is indeed a square root with

$$\mathsf{G}^2 = \frac{1}{4} \begin{pmatrix} 4+\sqrt{2} & 4-\sqrt{2} \\ 4-\sqrt{2} & 4+\sqrt{2} \end{pmatrix} \begin{pmatrix} 4+\sqrt{2} & 4-\sqrt{2} \\ 4-\sqrt{2} & 4+\sqrt{2} \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 36 & 28 \\ 28 & 36 \end{pmatrix} = \mathsf{A}.$$

2.4.5 Mahalanobis transformation and distance

Mahalanobis transformation

The Mahalanobis transformation is just a standardisation transformation using the (inverse) symmetric square root of the variance matrix. So if $\boldsymbol{X}$ is a random vector with mean $\boldsymbol{\mu}$ and variance matrix Σ, the Mahalanobis transformation is

$$\boldsymbol{Z} = \Sigma^{-1/2}(\boldsymbol{X} - \boldsymbol{\mu}),$$

where $\Sigma^{-1/2}$ is the inverse of the symmetric square root of Σ. $\boldsymbol{Z}$ will have mean $\mathbf{0}$ and variance I.

Similarly, if X is an $n \times p$ data matrix, the Mahalanobis transform is

$$\mathsf{Z} = \mathsf{H}_n \mathsf{X} \mathsf{S}^{-1/2},$$

though again, it must be emphasised that this is a mathematical description of the transformation, and not a recipe for how to construct it numerically. Z will have sample mean zero and sample variance I_p.

Example: Galaxy data

We will look now at how to compute the spectral decomposition of the `galaxy` data set using **R**, and how to use it to construct a symmetric square root, and its inverse.

```
> v=var(galaxy)
> v
                east.west north.south      angle radial.position
east.west       144.66088   -32.67993  -21.50402       263.93661
north.south     -32.67993   523.84971   26.35728      -261.78938
angle           -21.50402    26.35728 1462.62686       -49.01139
radial.position 263.93661  -261.78938  -49.01139       670.22991
velocity        451.46551 -1929.95131   37.21646      1637.78301
                   velocity
east.west          451.46551
north.south      -1929.95131
angle               37.21646
radial.position   1637.78301
velocity          8886.47724
> e=eigen(v,symmetric=TRUE)
> e
$values
[1] 9642.07343 1466.48964  487.88910   68.87222   22.52020

$vectors
             [,1]        [,2]        [,3]       [,4]        [,5]
[1,]  0.051396600  0.02254908  0.45279536  0.1122205  0.88274167
[2,] -0.208493672 -0.01789604  0.31495251  0.8879502 -0.26183862
[3,]  0.002463535 -0.99826125  0.04727651 -0.0346832  0.00551559
[4,]  0.182722226  0.05000384  0.82508240 -0.3634307 -0.38893366
[5,]  0.959424462 -0.01205694 -0.11307185  0.2562542 -0.03013096
```

```
> e$vectors%*%t(e$vectors)
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00 -7.224852e-17 -4.978182e-17  1.115915e-16
[2,] -7.224852e-17  1.000000e+00  1.760090e-16 -8.700722e-18
[3,] -4.978182e-17  1.760090e-16  1.000000e+00  2.170533e-16
[4,]  1.115915e-16 -8.700722e-18  2.170533e-16  1.000000e+00
[5,]  2.232271e-17 -1.676939e-16 -1.400287e-16  8.567399e-17
              [,5]
[1,]  2.232271e-17
[2,] -1.676939e-16
[3,] -1.400287e-16
[4,]  8.567399e-17
[5,]  1.000000e+00
> e$vectors%*%diag(e$values)%*%t(e$vectors)
         [,1]        [,2]       [,3]       [,4]        [,5]
[1,] 144.66088   -32.67993  -21.50402  263.93661   451.46551
[2,] -32.67993   523.84971   26.35728 -261.78938 -1929.95131
[3,] -21.50402    26.35728 1462.62686  -49.01139    37.21646
[4,] 263.93661  -261.78938  -49.01139  670.22991  1637.78301
[5,] 451.46551 -1929.95131   37.21646 1637.78301  8886.47724
> G=e$vectors%*%diag(sqrt(e$values))%*%t(e$vectors)
> G
           [,1]        [,2]       [,3]       [,4]        [,5]
[1,]  8.6098647   1.8123819 -0.3859399  7.2496158   3.8132008
[2,]  1.8123819  13.3404531  0.7001528 -0.2300872 -18.4947058
[3,] -0.3859399   0.7001528 38.2218104 -0.9113324   0.5003811
[4,]  7.2496158  -0.2300872 -0.9113324 20.2249833  14.4131733
[5,]  3.8132008 -18.4947058  0.5003811 14.4131733  91.2244082
> G%*%G
         [,1]        [,2]       [,3]       [,4]        [,5]
[1,] 144.66088   -32.67993  -21.50402  263.93661   451.46551
[2,] -32.67993   523.84971   26.35728 -261.78938 -1929.95131
[3,] -21.50402    26.35728 1462.62686  -49.01139    37.21646
[4,] 263.93661  -261.78938  -49.01139  670.22991  1637.78301
[5,] 451.46551 -1929.95131   37.21646 1637.78301  8886.47724
> Ginv=e$vectors%*%diag(1/sqrt(e$values))%*%t(e$vectors)
> G%*%Ginv
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00 -2.478825e-16 -2.061869e-17  1.212003e-16
[2,] -2.428952e-16  1.000000e+00  3.009745e-16  1.170261e-16
[3,] -1.111113e-15  8.248890e-16  1.000000e+00  1.020284e-15
[4,]  6.617055e-16 -3.846478e-16  2.830987e-16  1.000000e+00
[5,]  8.075138e-16 -1.285430e-15 -4.671759e-16 -4.477755e-17
              [,5]
[1,]  1.171616e-17
[2,] -2.679606e-16
[3,] -1.481901e-16
[4,]  8.386304e-17
[5,]  1.000000e+00
```

Now we have `Ginv` we can use it to construct the Mahalanobis transform of the data

```
> W=sweep(galaxy,2,colMeans(galaxy))
> Z=as.matrix(W)%*%Ginv
> colMeans(Z)
[1]  4.419302e-16 -2.600506e-15  2.784151e-17  1.075502e-15
[5] -1.821798e-15
> var(Z)
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00  6.811770e-16 -8.034862e-16 1.586732e-15
[2,]  6.811770e-16  1.000000e+00  1.309490e-15 5.266862e-15
[3,] -8.034862e-16  1.309490e-15  1.000000e+00 4.550018e-16
[4,]  1.586732e-15  5.266862e-15  4.550018e-16 1.000000e+00
[5,] -3.265377e-16 -2.529045e-15 -2.198962e-16 1.568981e-15
              [,5]
[1,] -3.265377e-16
[2,] -2.529045e-15
[3,] -2.198962e-16
[4,]  1.568981e-15
[5,]  1.000000e+00
```

Mahalanobis distance

Given an observation $\boldsymbol{x}$ of a random vector $\boldsymbol{X}$ with $\mathrm{E}(\boldsymbol{X}) = \boldsymbol{\mu}$ and $\mathrm{Var}(\boldsymbol{X}) = \Sigma$, it is natural to wonder how "unusual" it is, in some appropriate sense. We cannot give a very formal solution to this problem until we develop some multivariate distribution theory, but even now we can give an intuitive explanation of the most commonly used summary. In some sense we want to know how "far" the observation is from the mean, but in general some components are more variable than others, so it doesn't make sense to look at "distance" on the scale of the original variables. However, it does make perfect sense for a standardised observation, $\boldsymbol{z}$. So the Mahalanobis distance of $\boldsymbol{x}$ from $\boldsymbol{\mu}$ is just

$$\begin{aligned}\|\boldsymbol{z}\| &= \sqrt{\boldsymbol{z}^{\mathsf{T}}\boldsymbol{z}} \\ &= \sqrt{[\Sigma^{-1/2}(\boldsymbol{x}-\boldsymbol{\mu})]^{\mathsf{T}}\Sigma^{-1/2}(\boldsymbol{x}-\boldsymbol{\mu})} \\ &= \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}.\end{aligned}$$

Computationally, the efficient way to compute this distance measure is as $\sqrt{\boldsymbol{z}^{\mathsf{T}}\boldsymbol{z}}$ where

$$\boldsymbol{z} = \Sigma^{-1/2}(\boldsymbol{x}-\boldsymbol{\mu}).$$

Note that it is not necessary to use a symmetric square root for this computation. Standardisation using the Cholesky factor leads to exactly the same distance, and is much cheaper to compute and then solve than using a symmetric square root. Confirmation of this fact is left as an exercise.

It is also worth noting that if the variance is defined empirically, as the sample variance matrix S of a data matrix X, then the Mahalanobis distance can be calculated yet more stably and efficiently using the QR factorisation of X.

Computing the Mahalanobis distance for each observation in a data set is the most commonly used technique for identifying outlying observations in multivariate data.

Example: Galaxy data

Continuing our previous example, we have already constructed a standardised data matrix `Z` using a Mahalanobis transformation. We can use it to construct a Mahalanobis distance for each observation as follows.

```
d=sqrt(apply(Z*Z,1,sum))
```

Note that regular element-wise multiplication was used to square the elements in `Z`, rather than matrix multiplication, which is not even valid here due to `Z` being non-square. A histogram of the distances (not shown) can be obtained with `hist(d,20)`, and this shows a fairly uninteresting distribution, suggesting no particularly outlying values. There are a few distances larger than 4, so it may be worth looking at those in more detail. We can inspect the cases individually using

```
> galaxy[d>4,]
    east.west north.south angle radial.position velocity
286  24.85322    49.84784  63.5            55.7     1531
287  23.82696    47.78950  63.5            53.4     1533
288  22.80071    45.73114  63.5            51.1     1539
330 -21.46211   -43.04634  63.5           -48.1     1642
331 -22.48837   -45.10469  63.5           -50.4     1616
```

and we can highlight them on a scatterplot using

```
> plot(galaxy$radial.position,galaxy$velocity,pch=19,col=4)
> points(galaxy$radial.position[d>4],galaxy$velocity[d>4],pch=19,col=2)
```

resulting in the plot show in Figure 2.1. Although it is difficult to tell much from a single 2d projection, it does appear as though the 5 values highlighted are indeed some way away from the centre of the cloud of data points.

Note that once again, **R** has a built-in function for computing the (squared) Mahalanobis distance, so our vector of distances could have been constructed much more straightforwardly using

```
d=sqrt(mahalanobis(galaxy,colMeans(galaxy),var(galaxy)))
```

2.4.6 The singular value decomposition (SVD)

The final matrix factorisation we will examine here, the singular value decomposition, can be applied to arbitrary $n \times p$ matrices (we typically consider the case $n \geq p$), and is closely related to both the QR factorisation and the spectral decomposition. As for the QR factorisation, there are a few variants, and we concentrate here on a variant known as the "thin" SVD, as it is the most commonly used in practice.

Proposition 21 *The $n \times p$ matrix* A *can be factorised as*

$$\mathsf{A} = \mathsf{U}\mathsf{D}\mathsf{V}^{\mathsf{T}},$$

where U *is an $n \times p$ matrix with orthonormal columns ($\mathsf{U}^{\mathsf{T}}\mathsf{U} = \mathrm{I}_p$), D is a $p \times p$ diagonal matrix with non-negative entries, and* V *is a $p \times p$ orthogonal matrix. The diagonal entries of* D *are known as the* singular values *of* A*. The columns of* U *are known as the* left singular vectors*, and the columns of* V *are known as the* right singular vectors*.*

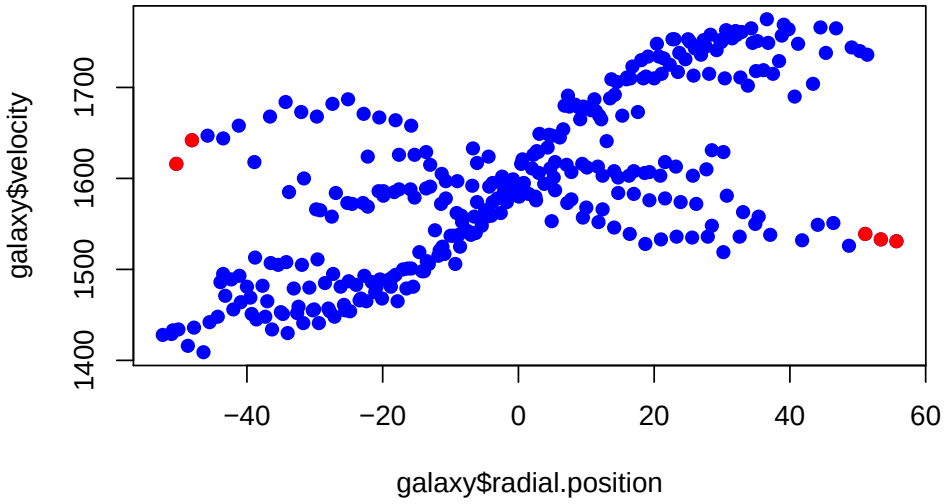


Figure 2.1: Scatterplot for the Galaxy data showing outliers identified by large Mahalanobis distance

Proof

Consider the spectral decomposition of $\mathsf{A}^\mathsf{T}\mathsf{A}$, which is clearly $p \times p$, real, symmetric, and positive semi-definite. We can therefore write the spectral decomposition in the form

$$\mathsf{A}^\mathsf{T}\mathsf{A} = \mathsf{V}\mathsf{D}^2\mathsf{V}^\mathsf{T},$$

as the eigenvalues are non-negative. By construction, V is an orthogonal matrix. We now put

$$\mathsf{U} = \mathsf{A}\mathsf{V}\mathsf{D}^{-1},$$

to get the SVD $\mathsf{A} = \mathsf{U}\mathsf{D}\mathsf{V}^\mathsf{T}$, and we verify that the columns of U are orthogonal as

$$\begin{aligned}
\mathsf{U}^\mathsf{T}\mathsf{U} &= (\mathsf{A}\mathsf{V}\mathsf{D}^{-1})^\mathsf{T}(\mathsf{A}\mathsf{V}\mathsf{D}^{-1}) \\
&= \mathsf{D}^{-1}\mathsf{V}^\mathsf{T}\mathsf{A}^\mathsf{T}\mathsf{A}\mathsf{V}\mathsf{D}^{-1} \\
&= \mathsf{D}^{-1}\mathsf{V}^\mathsf{T}\mathsf{V}\mathsf{D}^2\mathsf{V}^\mathsf{T}\mathsf{V}\mathsf{D}^{-1} \\
&= \mathsf{D}^{-1}\mathsf{D}^2\mathsf{D}^{-1} \\
&= \mathrm{I}_p\,.
\end{aligned}$$

□

It must be emphasised that this is a mathematical construction to demonstrate the existence of the factorisation, and its relationship to the spectral decomposition, but this is not how the factorisation is constructed in practice (and in any case, there are issues with this construction when D is singular). There are very efficient methods which can be used for this, often starting from a QR factorisation of A. Typically $\mathsf{A}^\mathsf{T}\mathsf{A}$ is not constructed at all, as this is a numerically unstable computation.

The SVD and spectral decomposition

From our construction it is clear that the columns of V are the eigenvectors of A^TA, so the right singular vectors are eigenvectors of A^TA. Similarly, the $n \times n$ matrix $AA^T = UD^2U^T$. Now U is $n \times p$, but post-multiplying by U gives

$$(AA^T)U = UD^2,$$

and so the columns of U are eigenvectors of AA^T corresponding to the p eigenvalues in D^2, which includes all of the non-zero eigenvalues of AA^T. So the left singular vectors are (the important) eigenvectors of AA^T.

It should be clear that if the SVD is applied directly to a real symmetric positive semi-definite matrix, that the spectral decomposition is obtained. Numerically, however, the algorithms used to calculate the SVD are quite different to those typically used for eigen-solvers. The SVD will typically be slightly slower than a symmetric eigen-solver but faster than an asymmetric eigen-solver. The SVD may be more numerically stable if the input matrix is close to singular, and is therefore very commonly used in practice.

SVD for least squares

The SVD provides an alternative approach to solving the normal equations

$$X^TX\boldsymbol{\beta} = X^T\boldsymbol{y}$$

for $\boldsymbol{\beta}$. Let $X = UDV^T$ be the SVD of X, and then

$$\begin{aligned} X^TX\boldsymbol{\beta} &= X^T\boldsymbol{y} \\ \Rightarrow (UDV^T)^T(UDV^T)\boldsymbol{\beta} &= (UDV^T)^T\boldsymbol{y} \\ \Rightarrow VDU^TUDV^T\boldsymbol{\beta} &= VDU^T\boldsymbol{y} \\ \Rightarrow VD^2V^T\boldsymbol{\beta} &= VDU^T\boldsymbol{y} \\ \Rightarrow DV^T\boldsymbol{\beta} &= U^T\boldsymbol{y} \\ \Rightarrow \boldsymbol{\beta} &= VD^{-1}U^T\boldsymbol{y}. \end{aligned}$$

This turns out to be a numerically stable way of solving (full rank) least squares problems, but since the SVD is slower than the QR factorisation, the previously described QR method is usually to be preferred.

Example: Galaxy data

We can form the SVD of the variance matrix for the `galaxy` data set using

```
> svd(var(galaxy))
$d
[1] 9642.07343 1466.48964  487.88910   68.87222   22.52020

$u
              [,1]        [,2]        [,3]       [,4]        [,5]
[1,] -0.051396600 -0.02254908 -0.45279536 -0.1122205  0.88274167
```

```
[2,]  0.208493672  0.01789604 -0.31495251 -0.8879502 -0.26183862
[3,] -0.002463535  0.99826125 -0.04727651  0.0346832  0.00551559
[4,] -0.182722226 -0.05000384 -0.82508240  0.3634307 -0.38893366
[5,] -0.959424462  0.01205694  0.11307185 -0.2562542 -0.03013096

$v
             [,1]        [,2]        [,3]       [,4]        [,5]
[1,] -0.051396600 -0.02254908 -0.45279536 -0.1122205  0.88274167
[2,]  0.208493672  0.01789604 -0.31495251 -0.8879502 -0.26183862
[3,] -0.002463535  0.99826125 -0.04727651  0.0346832  0.00551559
[4,] -0.182722226 -0.05000384 -0.82508240  0.3634307 -0.38893366
[5,] -0.959424462  0.01205694  0.11307185 -0.2562542 -0.03013096
```

We see that it is exactly the same as the spectral decomposition obtained earlier, modulo some sign changes on some of the columns, which are arbitrary.

2.5 Principal components analysis (PCA)

2.5.1 Derivation from the spectral decomposition

Principal components analysis, or PCA as it is more commonly known, is a technique for finding interesting low dimensional projections of a multivariate random variable or, more commonly, data matrix. It is very closely related to the spectral decomposition and SVD. We will begin by understanding principal components in relation to the eigen-decomposition of a variance matrix. To keep things as simple as possible, we begin by considering a p-dimensional random vector $\boldsymbol{X}$ with $\mathrm{E}(\boldsymbol{X}) = \boldsymbol{\mu}$ and $\mathrm{Var}(\boldsymbol{X}) = \Sigma$. Write the spectral decomposition of Σ as

$$\Sigma = \mathsf{V}\mathsf{D}\mathsf{V}^{\mathsf{T}}.$$

Now consider a linear combination $\boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{X}$, or one-dimensional projection of $\boldsymbol{X}$, a scalar random quantity for given fixed p-dimensional $\boldsymbol{\alpha}$. We know that the variance of this linear combination is $\boldsymbol{\alpha}^{\mathsf{T}}\Sigma\boldsymbol{\alpha}$. Let us now consider the linear combination represented by the eigenvectors in the columns of V. The ith eigenvector, $\boldsymbol{v}_i$, corresponding to the eigenvalue λ_i, has variance

$$\boldsymbol{v}_i{}^{\mathsf{T}}\Sigma\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i{}^{\mathsf{T}}\boldsymbol{v}_i = \lambda_i.$$

So the eigenvalues are the variances of the linear combinations represented by the corresponding eigenvectors. Similarly, if we consider the correlation between the random quantities represented by two distinct eigenvectors, $\boldsymbol{v_i}$, $\boldsymbol{v}_j$, we have

$$\mathrm{Cov}\left(\boldsymbol{v}_i{}^{\mathsf{T}}\boldsymbol{X}, \boldsymbol{v}_j{}^{\mathsf{T}}\boldsymbol{X}\right) = \boldsymbol{v}_i{}^{\mathsf{T}}\,\mathrm{Cov}(\boldsymbol{X}, \boldsymbol{X})\,\boldsymbol{v}_j = \boldsymbol{v}_i{}^{\mathsf{T}}\Sigma\boldsymbol{v}_j = \lambda_j\boldsymbol{v}_i{}^{\mathsf{T}}\boldsymbol{v}_j = 0.$$

So the eigenvectors form a basis for the space of linear combinations of $\boldsymbol{X}$, and represent a collection of uncorrelated random quantities whose variances are given by their respective eigenvalues.

Formally we characterise principal components as linear combinations of variables that explain the greatest variation in the random quantity or data set. However, we need to place a constraint on the linear combination for this to be well defined. For example,

if $\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{X}$ is a linear combination with variance σ^2, then it is clear that $k\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{X}$ has variance $k^2\sigma^2$, so we can make the variance of a linear combination as large as we wish simply by multiplying it by a large enough scalar. However, we are really interested in the *direction* with largest variation, and so we impose the constraint $\|\boldsymbol{\alpha}\| = 1$, or $\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{\alpha} = 1$. To maximise subject to a non-linear constraint we use a Lagrange multiplier, and begin by optimising

$$\begin{aligned} f &= \boldsymbol{\alpha}^\mathsf{T}\Sigma\boldsymbol{\alpha} - \lambda(\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{\alpha} - 1) \\ &= \boldsymbol{\alpha}^\mathsf{T}(\Sigma - \lambda\,\mathrm{I})\boldsymbol{\alpha} + \lambda. \end{aligned}$$

Now, recalling that for a symmetric matrix A, we have

$$\boldsymbol{\nabla}\boldsymbol{x}^\mathsf{T}\mathsf{A}\boldsymbol{x} = \frac{\partial}{\partial\boldsymbol{x}}\boldsymbol{x}^\mathsf{T}\mathsf{A}\boldsymbol{x} = 2\mathsf{A}\boldsymbol{x},$$

we can differentiate and equate to zero to get

$$\frac{\partial f}{\partial\boldsymbol{\alpha}} = 2(\Sigma - \lambda\,\mathrm{I})\boldsymbol{\alpha} = 0.$$

In other words,

$$|\Sigma - \lambda\,\mathrm{I}\,| = 0,$$

and λ is an eigenvalue of Σ, and $\boldsymbol{\alpha}$ is the (normalised) eigenvector corresponding to λ. Since we typically construct the spectral decomposition with eigenvalues in decreasing order, we pick the solution with largest variance as

$$\lambda = \lambda_1, \quad \boldsymbol{\alpha} = \boldsymbol{v}_1.$$

So the first eigenvector of the spectral decomposition represents the linear combination of variables with the largest variance, and its variance is given by its corresponding eigenvalue. This is the first principal component.

The second principal component is defined to be the linear combination uncorrelated with the first principal component which has the greatest variance. But since we have seen that the linear combinations corresponding to the eigenvectors are uncorrelated, it follows that the set of linear combinations corresponding to quantities uncorrelated with the first principal component is spanned by the 2nd to pth eigenvectors, and hence the maximum variance will be obtained at λ_2, corresponding to the 2nd eigenvector, $\boldsymbol{v}_2$. The third principal component is the linear combination with greatest variance uncorrelated with the first two principal components, and so on. So we see that there is a direct correspondence between the spectral decomposition of the variance matrix and the principal components of the random vector.

We can associate each principal component with a random quantity, $Y_i = \boldsymbol{v}_i{}^\mathsf{T}\boldsymbol{X},\ i = 1, 2, \ldots, p$, and use these to construct a new random vector

$$\boldsymbol{Y} = \mathsf{V}^\mathsf{T}\boldsymbol{X},$$

the vector of principal components. The variance matrix of Y is clearly given by

$$\begin{aligned} \mathrm{Var}(\boldsymbol{Y}) &= \mathrm{Var}\left(\mathsf{V}^\mathsf{T}\boldsymbol{X}\right) \\ &= \mathsf{V}^\mathsf{T}\,\mathrm{Var}(\boldsymbol{X})\,\mathsf{V} \\ &= \mathsf{V}^\mathsf{T}\mathsf{V}\mathsf{D}\mathsf{V}^\mathsf{T}\mathsf{V} \\ &= \mathsf{D}. \end{aligned}$$

So the components of Y are uncorrelated, and each has the correct variance, as required.

2.5.2 Total variation and variance explained

We know that the variance matrix is a very rich summary of the variability of a random vector, $\boldsymbol{X}$. However, sometimes it is useful to have a single scalar summary which captures some aspect of the overall variability of the random vector. There are obviously many ways that this could be done, and none will be perfect, but a commonly used summary of variation is the sum of the variances of the individual variables,

$$t(\boldsymbol{X}) = \sum_{i=1}^{p} \mathrm{Var}(X_i) = \mathrm{Tr}\left(\mathrm{Var}(\boldsymbol{X})\right),$$

often known as the *total variation* of the random vector. The total variation has the desirable property that it is preserved by orthogonal rotation. In particular, if we consider transforming $\boldsymbol{X}$ to its principal components via

$$\boldsymbol{Y} = \mathsf{V}^{\mathsf{T}}\boldsymbol{X}$$

we have $t(\boldsymbol{X}) = t(\boldsymbol{Y})$, since

$$\begin{aligned} t(\boldsymbol{X}) &= \mathrm{Tr}\left(\mathrm{Var}(\boldsymbol{X})\right) \\ &= \mathrm{Tr}\left(\mathsf{V}\mathsf{D}\mathsf{V}^{\mathsf{T}}\right) \\ &= \mathrm{Tr}\left(\mathsf{D}\mathsf{V}^{\mathsf{T}}\mathsf{V}\right) \\ &= \mathrm{Tr}\left(\mathsf{D}\right) \\ &= t(\boldsymbol{Y}), \end{aligned}$$

using the fact that $\mathrm{Tr}\left(\mathsf{AB}\right) = \mathrm{Tr}\left(\mathsf{BA}\right)$ for square matrices.

One motivation for undertaking principal components analysis is to try and find low-dimensional projections of the original variables which contain most of the information in the original higher-dimensional vectors. One way of characterising the information is by its variation. Since the principal components are ordered with most variable first, it makes sense to consider the first q principal components ($q \leq p$). The total variation of the first q principal components is clearly $\sum_{i=1}^{q} \lambda_i$. The proportion of variance explained is therefore

$$\frac{\sum_{i=1}^{q} \lambda_i}{\sum_{i=1}^{p} \lambda_i} = \frac{1}{\mathrm{Tr}\left(D\right)} \sum_{i=1}^{q} \lambda_i,$$

with the proportion increasing to 1 by $q = p$. Therefore an obvious strategy for dimension reduction is to work with the first q principal components, where q is the smallest number such that the proportion of variance explained exceeds a given pre-specified threshold. The actual threshold will be problem dependent — there are no hard-and-fast rules, as principal components are primarily an exploratory tool.

2.5.3 Principal components from a sample variance matrix

To keep the notation simple we have introduced the idea of principal components for a random vector. However, in practice principal components analysis is mainly used in the

analysis of multivariate data. So now suppose that we have an $n \times p$ data matrix, X, and wish to use principal components analysis to find interesting low-dimensional projections. Let us suppose that we have the sample variance matrix

$$\mathsf{S} = \frac{1}{n-1}\mathsf{X}^\mathsf{T}\mathsf{H}_n\mathsf{X}$$

and form its spectral decomposition as

$$\mathsf{S} = \mathsf{V}\mathsf{D}\mathsf{V}^\mathsf{T}.$$

Then all of the properties we described for random variables easily follow. For example, the eigenvectors in the columns of V correspond to linear combinations of the variables whose sample variance is the corresponding eigenvalue, and distinct eigenvectors correspond to linear combinations of variables whose sample covariance is zero. So we can define the first principal component of X to be the linear combination of variables whose sample variance is largest (subject to the unit norm constraint), and it trivially follows that this corresponds to the first eigenvector, $\boldsymbol{v}_1$, and its sample variance is λ_1. Similarly the second principal component corresponds to $\boldsymbol{v}_2$, with sample variance λ_2, and so on.

We can obviously transform observations to their principal component representation using

$$\boldsymbol{y} = \mathsf{V}^\mathsf{T}\boldsymbol{x}$$

and so we can transform X to Y, the data rotated to the principal component axes, using

$$\mathsf{Y} = \mathsf{X}\mathsf{V}.$$

However, in practice the data are usually centered before rotating, so in practice the transformation

$$\mathsf{Y} = \mathsf{H}_n\mathsf{X}\mathsf{V}$$

is used.[§] In the context of principal components analysis, the matrix of eigenvectors, V, is often known as the *loadings matrix*, and the transformed observations in Y are often known as *component scores*.

Example: Galaxy data

We will begin by constructing a PCA "by hand" using the spectral decomposition of the sample variance matrix. We will use the objects `v` (the sample variance matrix), `W` (the centered data matrix) and `e` (the spectral decomposition of `v`) from the previous example. We can examine the variance explained by the principal components using

```
> cumsum(e$values)/sum(e$values)
[1] 0.8249659 0.9504373 0.9921806 0.9980732 1.0000000
```

So we see that the first principal component already explains over 80% of the total variation, and the first two together explain over 95% of the variation. We should therefore get a very adequate representation of the data by projection into the first two components, and essentially all of the variation is explained by the first 3 components. We can construct the principal component scores with

[§]The usual caveat about not actually using the centering matrix to center the data applies.

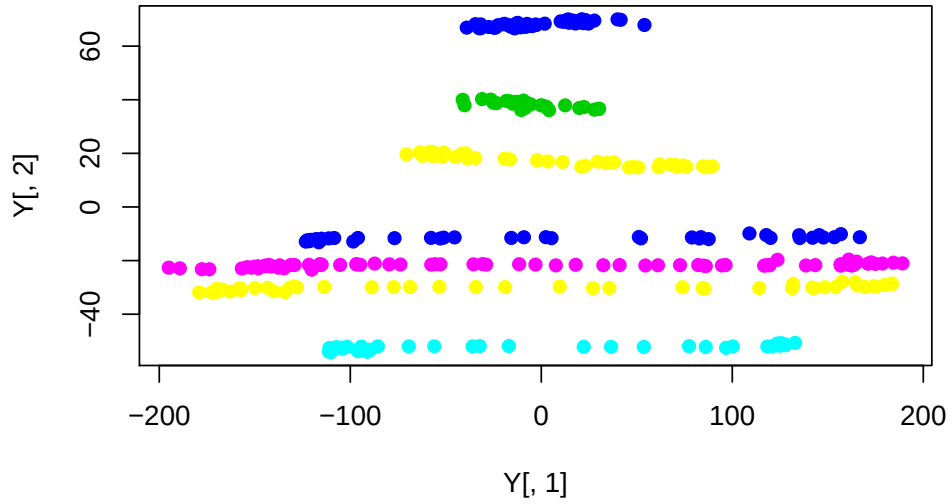


Figure 2.2: Scatterplot for the Galaxy data showing the second principal component plotted against the first

```
Y=as.matrix(W)%*%e$vectors
```

We can plot the second principal component against the first using

```
plot(Y[,1],Y[,2],pch=19,col=galaxy$angle)
```

giving the plot shown in Figure 2.2.

As can be verified by inspecting the factor loading matrix, the first principal component is dominated by `velocity` and the second is dominated by `angle`. We have rotated the data to find a 2d projection where the observations are as "spread out" as possible. We can also plot the third principal component against the first (not shown) in order to get information "orthogonal" to the `angle` dominated principal component. In fact, if the `rgl` package is installed, we can produce an interactive 3d plot of the first 3 principal components as

```
require(rgl)
plot3d(Y[,1:3],col=galaxy$angle)
```

Since the first 3 principal components explain over 99% of the variation in the data, essentially no information is lost in this 3d projection, and the ability to interact with the plot, and drag it around to look at it from different angles gives a great deal of insight into the structure of the data. A screen grab of this plot is shown in Figure 2.3.

We have investigated the principal components for this data by hand, but **R** has built-in functions for carrying out principal components analysis. The classic function for carrying out PCA was the `princomp()` function, which works similarly to how we have just been doing things "by hand".

```
> pca=princomp(galaxy)
> pca
Call:
```

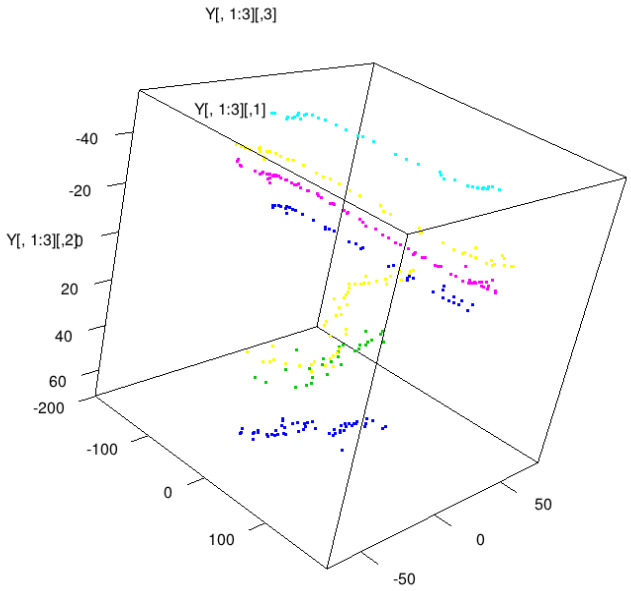


Figure 2.3: Screen grab of a 3d interactive plot of the first three principal components for the galaxy data

```
princomp(x = galaxy)

Standard deviations:
   Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
98.041939 38.235447 22.053993  8.286072  4.738194

 5  variables and  323 observations.
> pca$loadings

Loadings:
               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
east.west                     0.453  0.112  0.883
north.south    -0.208         0.315  0.888 -0.262
angle                 -0.998
radial.position  0.183         0.825 -0.363 -0.389
velocity         0.959        -0.113  0.256

               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings       1.0    1.0    1.0    1.0    1.0
Proportion Var    0.2    0.2    0.2    0.2    0.2
Cumulative Var    0.2    0.4    0.6    0.8    1.0
> dim(pca$scores)
[1] 323   5
> plot(pca$scores[,1],pca$scores[,2],pch=19,col=galaxy$angle)
```

The final command produces a plot very similar to Figure 2.2. Also note that `summary`

`(pca)` will print some additional information and that `plot(pca)` will produce a barplot indicating the variances of the principal components. This plot is sometimes referred to as a *scree plot*, and can also be produced by directly calling `screeplot(pca)`.

2.5.4 Construction from the SVD

Although the spectral decomposition of the variance matrix is a conceptually elegant way to understand principal components, it turns out that constructing them this way from a sample variance matrix is inefficient and numerically unstable. Starting from a data matrix X, we first compute the centred data

$$W = H_n X.$$

Then the sample variance matrix is given by

$$S = \frac{1}{n-1} W^T W.$$

If we were to follow the usual approach, we could go on to form the spectral decomposition of S and construct principal components as before. However, it turns out that we can avoid the computation of S completely using a SVD of the centered data W. Putting

$$W = UDV^T$$

we see that

$$\begin{aligned} S &= \frac{1}{n-1} W^T W \\ &= \frac{1}{n-1} VDU^T UDV^T \\ &= \frac{1}{n-1} VD^2 V^T \\ &= V\Lambda V^T, \end{aligned}$$

where

$$\Lambda = \frac{1}{n-1} D^2.$$

We have therefore constructed the spectral decomposition of S directly from the SVD of W, and the loadings matrix of the principal components corresponds precisely to the right singular vectors of W. We can use this loadings matrix together with the diagonal matrix of eigenvalues, Λ just as if we had constructed S and then carried out a spectral decomposition.

Example: galaxy data

First we can confirm that the SVD gives the correct loading matrix, using

```
> svd(W)$v
              [,1]         [,2]         [,3]       [,4]         [,5]
[1,] -0.051396600 -0.02254908  0.45279536 -0.1122205  0.88274167
```

```
[2,]  0.208493672  0.01789604  0.31495251 -0.8879502 -0.26183862
[3,] -0.002463535  0.99826125  0.04727651  0.0346832  0.00551559
[4,] -0.182722226 -0.05000384  0.82508240  0.3634307 -0.38893366
[5,] -0.959424462  0.01205694 -0.11307185 -0.2562542 -0.03013096
```

We see that apart from some arbitrary sign changes, everything is as it should be. We looked previously at the R command `princomp()` which computes principal components the classical way, via spectral decomposition of the sample covariance matrix. R also has a command `prcomp()` which instead uses the SVD of the data matrix, as described above. Use of this command is similar to `princomp()`, but slightly different. The following R session demonstrates its use.

```
> pca=prcomp(galaxy)
> pca
Standard deviations:
[1] 98.194060 38.294773 22.088212  8.298929  4.745546

Rotation:
                          PC1         PC2         PC3        PC4
east.west       -0.051396600 -0.02254908  0.45279536 -0.1122205
north.south      0.208493672  0.01789604  0.31495251 -0.8879502
angle           -0.002463535  0.99826125  0.04727651  0.0346832
radial.position -0.182722226 -0.05000384  0.82508240  0.3634307
velocity        -0.959424462  0.01205694 -0.11307185 -0.2562542
                         PC5
east.west         0.88274167
north.south      -0.26183862
angle             0.00551559
radial.position  -0.38893366
velocity         -0.03013096
> pca$rotation
                          PC1         PC2         PC3        PC4
east.west       -0.051396600 -0.02254908  0.45279536 -0.1122205
north.south      0.208493672  0.01789604  0.31495251 -0.8879502
angle           -0.002463535  0.99826125  0.04727651  0.0346832
radial.position -0.182722226 -0.05000384  0.82508240  0.3634307
velocity        -0.959424462  0.01205694 -0.11307185 -0.2562542
                         PC5
east.west         0.88274167
north.south      -0.26183862
angle             0.00551559
radial.position  -0.38893366
velocity         -0.03013096
> dim(pca$x)
[1] 323   5
> plot(pca$x[,1],pca$x[,2],pch=19,col=galaxy$angle)
```

The final command again gives a plot similar to Figure 2.2. Since the SVD provides a more numerically stable approach to PCA, use of the command `prcomp()` is recommended unless there is a good reason not to.

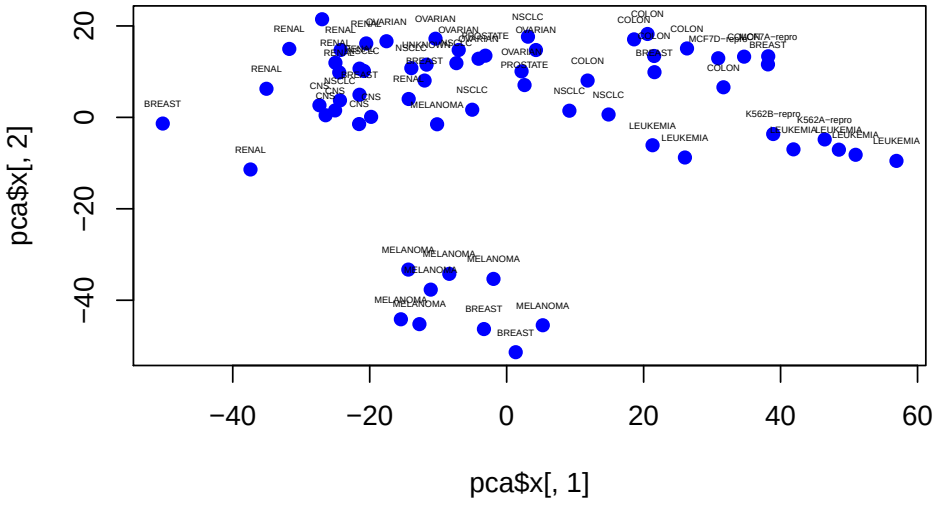


Figure 2.4: Scatterplot of the first two principal components of the `nci` microarray data

Example: microarray data

We can carry out PCA for the `nci` microarray data using the following commands

```
pca=prcomp(t(nci))
plot(pca$x[,1],pca$x[,2],pch=19,col=4)
text(pca$x[,1],pca$x[,2],colnames(nci),cex=0.3,pos=3)
```

This results in the plot shown in Figure 2.4.

Analysing the plot, we see some interesting structure immediately. There seems to be a cluster of Melanoma samples at the bottom, with two Breast cancers seemingly part of the same cluster. There also seems to be a cluster of Leukaemia samples on the right hand side of the plot, and a cluster of Renal cancers towards the left hand side of the plot. The first two principal components therefore provide us with a 2d projection of the 6,830 dimensional data that immediately allows us to begin to understand some of the underlying structure.

Note that calling `princomp(t(nci))` will give an error, as this command doesn't work in the $p > n$ case. This is another reason for preferring `prcomp()`.

Example: ZIP code digits

We can carry out a basic PCA for the ZIP digits as follows:

```
pca=prcomp(zip.train[,-1])
plot(pca$x[,1],pca$x[,2],pch=19,col=zip.train[,1])
```

This gives the plot shown in Figure 2.5.

Although there is clearly some structure in the plot, the individual digits are not well separated in this 2d projection. Note that it is not the goal of PCA to find projections to do

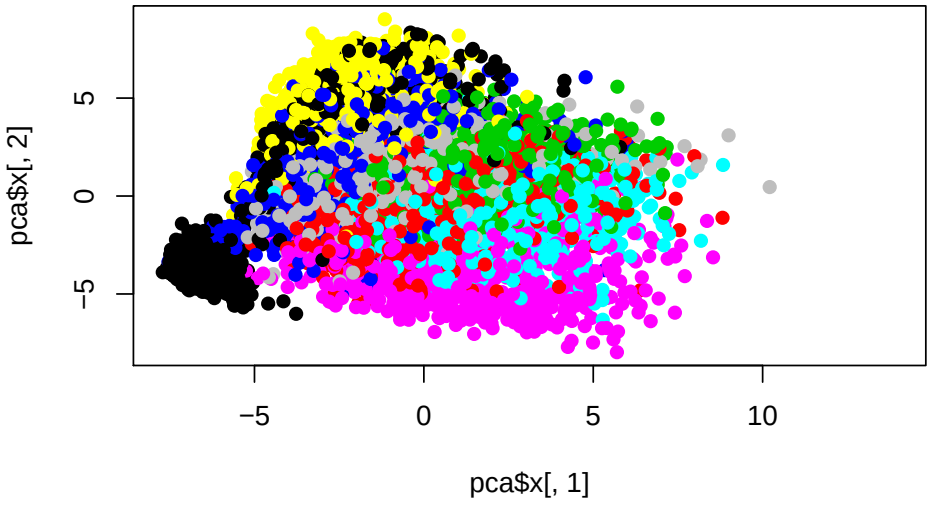


Figure 2.5: Scatterplot of the first two principal components of the `zip.train` digit image data

this — that is the goal of *discrimination*, which we shall analyse later in the course. However, PCA often happens to throw up projections which do discriminate, since they often correspond to the projections with greatest variation. However, analysis of a 3d plot of the first 3 principal components (not shown) shows that there are no obvious 2d projections of the first 3 principal components that discriminate between the digits particularly well.

So far we have not been paying much attention to the loadings for the principal components. Note that these are linear combinations of the variables, and hence can be interpreted as observations. Since here the observations are images, we can visualise the loadings of the principal components as images, also. We can generate images corresponding to the loadings of the first 4 principal components with

```
op=par(mfrow=c(2,2))
for (i in 1:4) {
  image(matrix(pca$rotation[,i],ncol=16)[,16:1],col=grey(15:0/15))
}
par(op)
```

leading to the plot shown in Figure 2.6.

This provides an effective way to understand the loading vectors in this particular case.

2.6 Conclusion

Numerical linear algebra and matrix computations are the key tools needed to make sense of multivariate data sets. The methods covered in this chapter provide the foundation for much of the rest of the course.

Note that the standard reference on numerical linear algebra is:

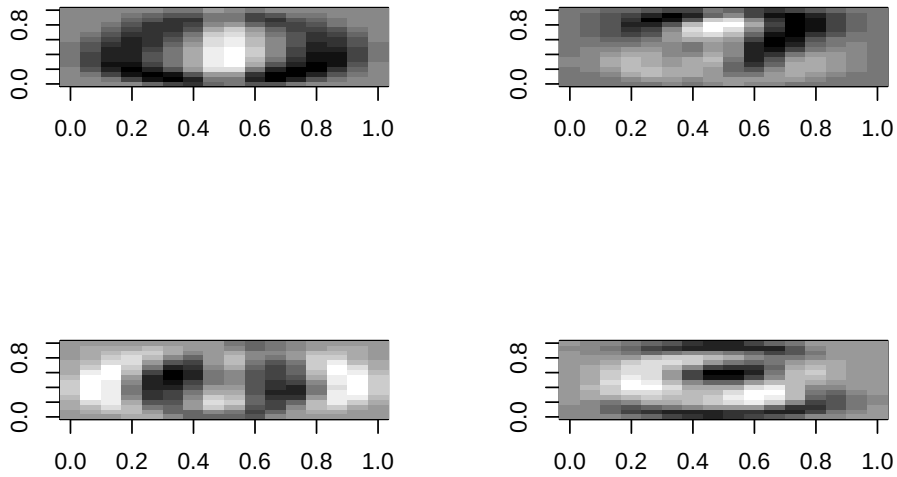


Figure 2.6: Images of the first 4 principal component loading vectors for the `zip.train` digit data

Golub, G.H. and Loan, C.F.V. (1996) Matrix computations, Johns Hopkins University Press.

Some of the examples in this chapter were based on corresponding examples in the above text. In addition, see section 3.2.3 (p.52) and 3.9 (p.93), as well as section 14.5 (p.534) of **[ESL]** for further details.

Chapter 3

Inference, the MVN and multivariate regression

3.1 Inference and estimation

So far we have considered the analysis of data matrices X, including the computation of the sample mean vector $\bar{\boldsymbol{x}}$ and variance matrix S_X. We have also considered random vectors $\boldsymbol{X}$, including the expectation vector $\mathrm{E}(\boldsymbol{X})$ and variance matrix $\mathrm{Var}(\boldsymbol{X})$. However, although we have informally noted a relationship between sample and population quantities, we have not yet formally established a connection between the two.

Proposition 22 *Suppose that we form an $n \times p$ data matrix X with rows $\boldsymbol{X}_i^\mathsf{T}$, $i = 1, 2, \ldots, n$ where the $\boldsymbol{X}_i$ are independent realisations of a multivariate random quantity with $\mathrm{E}(\boldsymbol{X}_i) = \boldsymbol{\mu}$ and $\mathrm{Var}(\boldsymbol{X}_i) = \Sigma$. Assuming that the elements of $\boldsymbol{\mu}$ and Σ are all finite, we have the following:*

1. *The sample mean $\bar{\boldsymbol{X}}$ is unbiased for $\boldsymbol{\mu}$, that is*

$$\mathrm{E}\big(\bar{\boldsymbol{X}}\big) = \boldsymbol{\mu}.$$

2. *The variance of $\bar{\boldsymbol{X}}$ is given by*

$$\mathrm{Var}\big(\bar{\boldsymbol{X}}\big) = \frac{1}{n}\Sigma.$$

3. *$\bar{\boldsymbol{X}}$ is a consistent estimator of $\boldsymbol{\mu}$, that is*

$$\mathrm{P}\big(\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\| < \varepsilon\big) \xrightarrow[\infty]{n} 1, \ \forall \varepsilon > 0.$$

4. *The sample variance S_X is* unbiased *for Σ, that is*

$$\mathrm{E}(\mathsf{S}_\mathsf{X}) = \Sigma.$$

Before demonstrating these results, it should be noted that under some additional (fairly mild) assumptions it can be shown that the sample variance matrix is also a *consistent* estimator of Σ, but we will not pursue this here.

Proof

1.

$$\begin{aligned}
\mathrm{E}\big(\bar{\boldsymbol{X}}\big) &= \mathrm{E}\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{X}_i\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathrm{E}(\boldsymbol{X}_i) \\
&= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\mu} \\
&= \boldsymbol{\mu}.
\end{aligned}$$

2.

$$\begin{aligned}
\mathrm{Var}\big(\bar{\boldsymbol{X}}\big) &= \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{X}_i\right) \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}(\boldsymbol{X}_i) \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\Sigma \\
&= \frac{1}{n}\Sigma.
\end{aligned}$$

3. Consistency is intuitively obvious, since 1. and 2. tell us that $\mathrm{E}\big(\bar{\boldsymbol{X}}\big) \to \mu$ and $\mathrm{Var}\big(\bar{\boldsymbol{X}}\big) \to 0$ as $n \to \infty$. However, we can establish consistency more formally by applying Markov's inequality to $\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\|^2$. First note that

$$\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\|^2 = (\bar{\boldsymbol{X}} - \boldsymbol{\mu})^{\mathsf{T}}(\bar{\boldsymbol{X}} - \boldsymbol{\mu}) = \sum_{i=1}^{p}(\bar{X}_i - \mu_i)^2 \geq 0.$$

Now we have

$$\begin{aligned}
\mathrm{E}\big(\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\|^2\big) &= \mathrm{E}\left(\sum_{i=1}^{p}(\bar{X}_i - \mu_i)^2\right) \\
&= \sum_{i=1}^{p}\mathrm{E}\big((\bar{X}_i - \mu_i)^2\big) \\
&= \sum_{i=1}^{p}\mathrm{Var}\big(\bar{X}_i\big) \\
&= \sum_{i=1}^{p}\frac{1}{n}\mathrm{Var}(X_i) \\
&= \frac{1}{n}\mathrm{Tr}\left(\Sigma\right).
\end{aligned}$$

Then Markov's inequality tells us that for any $a > 0$ we have

$$\begin{aligned} & \mathrm{P}\big(\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\|^2 \geq a\big) \leq \frac{\mathrm{Tr}\,(\Sigma)}{na} \\ \Rightarrow\ & \mathrm{P}\big(\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\|^2 < a\big) \geq 1 - \frac{\mathrm{Tr}\,(\Sigma)}{na} \end{aligned}$$

and putting $a = \varepsilon^2$ we get

$$\begin{aligned} & \mathrm{P}\big(\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\|^2 < \varepsilon^2\big) \geq 1 - \frac{\mathrm{Tr}\,(\Sigma)}{n\varepsilon^2} \\ \Rightarrow\ & \mathrm{P}\big(\|\bar{\boldsymbol{X}} - \boldsymbol{\mu}\| < \varepsilon\big) \geq 1 - \frac{\mathrm{Tr}\,(\Sigma)}{n\varepsilon^2} \\ & \xrightarrow[\infty]{n} 1. \end{aligned}$$

4. First note that

$$\begin{aligned} \mathrm{E}(\mathsf{S_X}) &= \mathrm{E}\left(\frac{1}{n-1}\sum_{i=1}^{n}[\boldsymbol{X}_i - \bar{\boldsymbol{X}}][\boldsymbol{X}_i - \bar{\boldsymbol{X}}]^\mathsf{T}\right) \\ &= \frac{1}{n-1}\sum_{i=1}^{n}\mathrm{E}\big([\boldsymbol{X}_i - \bar{\boldsymbol{X}}][\boldsymbol{X}_i - \bar{\boldsymbol{X}}]^\mathsf{T}\big) \end{aligned}$$

Now since

$$\mathrm{E}\big(\boldsymbol{X}_i - \bar{\boldsymbol{X}}\big) = \boldsymbol{\mu} - \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\mu} = \mathbf{0}$$

we have

$$\begin{aligned} \mathrm{E}\big([\boldsymbol{X}_i - \bar{\boldsymbol{X}}][\boldsymbol{X}_i - \bar{\boldsymbol{X}}]^\mathsf{T}\big) &= \mathrm{Var}\big(\boldsymbol{X}_i - \bar{\boldsymbol{X}}\big) \\ &= \mathrm{Var}\left(\frac{n-1}{n}\boldsymbol{X}_i - \frac{1}{n}\sum_{j\neq i}\boldsymbol{X}_j\right) \\ &= \frac{(n-1)^2}{n^2}\mathrm{Var}(\boldsymbol{X}_i) + \frac{1}{n^2}\sum_{j\neq i}\mathrm{Var}(\boldsymbol{X}_j) \\ &= \frac{(n-1)^2}{n^2}\Sigma + \frac{n-1}{n^2}\Sigma \\ &= \frac{n-1}{n}\Sigma, \end{aligned}$$

and so

$$\mathrm{E}(\mathsf{S_X}) = \frac{1}{n-1}\sum_{i=1}^{n}\frac{n-1}{n}\Sigma = \Sigma.$$

□

These results now make explicit the relationship between the data summaries and associated population quantities, and give us confidence that, at least in the case of "large n", our sample summary statistics should be "good" estimators of the corresponding summaries for the population from which the data is sampled.

3.2 Multivariate regression

In this section we will examine the problem of regression for a multivariate observation. We shall begin by reminding ourselves about multiple regression for a univariate output as a "least squares" procedure, before going on to see how this can be generalised to the case of multivariate output. Key to understanding regression as a "least squares" problem is the idea of minimising an error function with respect to a vector (or matrix) of parameters. There are various ways to approach this problem, but often the simplest is to use calculus to find a stationary point of the cost function (by differentiating wrt the parameters and equating to zero). For this purpose it is helpful to use some results on differentiating various scalar functions with respect to vector and matrix parameters. We state the following results without proof.*

Proposition 23 (Derivatives wrt vectors and matrices) *In each of the following results, the derivative is of a scalar function, and is wrt a vector $\boldsymbol{x}$ or matrix X, where the components of the vector or matrix are assumed to be algebraically independent.*

1. $$\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{a}^\mathsf{T} \boldsymbol{x} = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^\mathsf{T} \boldsymbol{a} = \boldsymbol{a}$$

2. $$\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^\mathsf{T} \mathsf{A} \boldsymbol{x} = (\mathsf{A} + \mathsf{A}^\mathsf{T}) \boldsymbol{x}$$

   and note that this reduces to $2\mathsf{A}\boldsymbol{x}$ when A is symmetric.

3. $$\frac{\partial}{\partial \mathsf{X}} \operatorname{Tr}(\mathsf{X}\mathsf{A}) = \mathsf{A}^\mathsf{T}$$

4. $$\frac{\partial}{\partial \mathsf{X}} \operatorname{Tr}\left(\mathsf{X}^\mathsf{T}\mathsf{A}\mathsf{X}\right) = \mathsf{A}\mathsf{X} + \mathsf{A}^\mathsf{T}\mathsf{X}$$

   and note that this reduces to $2\mathsf{A}\mathsf{X}$ when A is symmetric.

5. $$\frac{\partial}{\partial \mathsf{X}} \operatorname{Tr}\left(\mathsf{X}^\mathsf{T}\mathsf{A}\mathsf{X}\mathsf{B}\right) = \frac{\partial}{\partial \mathsf{X}} \operatorname{Tr}\left(\mathsf{B}\mathsf{X}^\mathsf{T}\mathsf{A}\mathsf{X}\right) = \mathsf{A}\mathsf{X}\mathsf{B} + \mathsf{A}^\mathsf{T}\mathsf{X}\mathsf{B}^\mathsf{T}$$

   and note that this reduces to $2\mathsf{A}\mathsf{X}\mathsf{B}$ when both A and B are symmetric.

6. $$\frac{\partial}{\partial \mathsf{X}} |\mathsf{X}| = |\mathsf{X}| \mathsf{X}^{-\mathsf{T}}$$

7. $$\frac{\partial}{\partial \mathsf{X}} \log |\mathsf{X}| = \mathsf{X}^{-\mathsf{T}}$$

*The results can all be found in The Matrix Cookbook, to which you are referred for further details and many other interesting results and examples. Note that you are not expected to memorise these results, and their derivations are not examinable in this course.

8.
$$\frac{\partial}{\partial \mathsf{X}} \boldsymbol{a}^\mathsf{T} \mathsf{X} \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^\mathsf{T}$$

9.
$$\frac{\partial}{\partial \mathsf{X}} \boldsymbol{a}^\mathsf{T} \mathsf{X}^{-1} \boldsymbol{b} = -\mathsf{X}^{-\mathsf{T}} \boldsymbol{a}\boldsymbol{b}^\mathsf{T} \mathsf{X}^{-\mathsf{T}}$$

Now we have the necessary technical results, we can think about the regression problem.

3.2.1 Univariate multiple linear regression

Let's start with classical multiple linear regression where we model each output y_i as a linear combination of a corresponding covariate vector $\boldsymbol{x}_i$, subject to error, as

$$y_i = \boldsymbol{x}_i^\mathsf{T} \boldsymbol{\beta} + \varepsilon_i, \ i = 1, 2, \ldots, n.$$

We assume that the covariates are the rows of an $n \times p$ data matrix X, and so $\boldsymbol{\beta}$ is a p-dimensional vector. We can write this in matrix form as

$$\boldsymbol{y} = \mathsf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

Note that we are not explicitly including an intercept term in the model. This can be accommodated by making the first column of X equal to $\mathbf{1}_n$. Since we typically consider the case $n > p$, the presence of the error term $\boldsymbol{\varepsilon}$ is necessary, otherwise the system would be over-determined, and then typically no $\boldsymbol{\beta}$ would exist in order to solve the linear system

$$\boldsymbol{y} = \mathsf{X}\boldsymbol{\beta}.$$

However, by introducing $\boldsymbol{\varepsilon}$, we ensure that *every* $\boldsymbol{\beta}$ is a potential solution, for an appropriate choice of $\boldsymbol{\varepsilon}$. The *least squares* approach is to choose the $\boldsymbol{\beta}$ which makes the 2-norm of $\boldsymbol{\varepsilon}$ as small as possible. That is, we choose $\boldsymbol{\beta}$ in order to minimise

$$\|\boldsymbol{\varepsilon}\|^2 = \boldsymbol{\varepsilon}^\mathsf{T}\boldsymbol{\varepsilon} = \sum_{i=1}^{n} \varepsilon_i^2.$$

We can find such a minimising solution by first noting that

$$\begin{aligned} \boldsymbol{\varepsilon}^\mathsf{T}\boldsymbol{\varepsilon} &= (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})^\mathsf{T}(\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta}) \\ &= \boldsymbol{y}^\mathsf{T}\boldsymbol{y} - 2\boldsymbol{y}^\mathsf{T}\mathsf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{X}\boldsymbol{\beta}, \end{aligned}$$

and then differentiate wrt $\boldsymbol{\beta}$ as

$$\frac{\partial}{\partial \boldsymbol{\beta}} \boldsymbol{\varepsilon}^\mathsf{T}\boldsymbol{\varepsilon} = -2\mathsf{X}^\mathsf{T}\boldsymbol{y} + 2\mathsf{X}^\mathsf{T}\mathsf{X}\boldsymbol{\beta}.$$

Here we have used results 1. and 2. of Proposition 23. Then equating to zero leads to the "normal equations"

$$\mathsf{X}^\mathsf{T}\mathsf{X}\hat{\boldsymbol{\beta}} = \mathsf{X}^\mathsf{T}\boldsymbol{y}.$$

The mathematical solution to this is

$$\hat{\boldsymbol{\beta}} = (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y},$$

but there are very efficient and numerically stable ways to compute this based on matrix decompositions such as the QR factorisation, as discussed in Section 2.4.3.

Example: galaxy data

Suppose that we would like to use the variables `angle` and `radial.position` in order to predict the variable `velocity`. We can accomplish this in **R** using the built-in command `lm()`, either using **R** model notation, as

```
> lm(velocity~angle+radial.position,data=galaxy)

Call:
lm(formula = velocity ~ angle + radial.position, data = galaxy)

Coefficients:
    (Intercept)            angle  radial.position
      1586.9882           0.1076           2.4515
```

or just using vectors and matrices, as

```
> lm(galaxy[,5]~as.matrix(galaxy[,3:4]))

Call:
lm(formula = galaxy[, 5] ~ as.matrix(galaxy[, 3:4]))

Coefficients:
                             (Intercept)
                               1586.9882
          as.matrix(galaxy[, 3:4])angle
                                  0.1076
as.matrix(galaxy[, 3:4])radial.position
                                  2.4515
```

Note that `lm()` includes an intercept term by default, as this is usually what we want. However, we can tell **R** not to include an intercept term by using `0+` at the beginning of the covariate list, either using model notation

```
> lm(velocity~0+angle+radial.position,data=galaxy)

Call:
lm(formula = velocity ~ 0 + angle + radial.position, data = galaxy)

Coefficients:
          angle  radial.position
         16.163            3.261
```

or matrix notation:

```
> lm(galaxy[,5]~0+as.matrix(galaxy[,3:4]))

Call:
lm(formula = galaxy[, 5] ~ 0 + as.matrix(galaxy[, 3:4]))

Coefficients:
          as.matrix(galaxy[, 3:4])angle
                                 16.163
as.matrix(galaxy[, 3:4])radial.position
                                  3.261
```

We can compute the regression coefficients directly using a QR factorisation with

```
> QR=qr(as.matrix(galaxy[,3:4]))
> backsolve(qr.R(QR),t(qr.Q(QR))%*%galaxy[,5])
          [,1]
[1,] 16.163269
[2,]  3.261159
```

The intercept term can be found by appending $\mathbf{1}$ to the front of X, as

```
> QR=qr(cbind(rep(1,323),galaxy[,3:4]))
> backsolve(qr.R(QR),t(qr.Q(QR))%*%galaxy[,5])
             [,1]
[1,] 1586.9881822
[2,]    0.1075920
[3,]    2.4514815
```

So we see that least squares problems can be solved with a QR factorisation, a rotation, and a backwards solve. This is the method that the `lm()` function uses.

Now, just as the sample mean $\bar{\boldsymbol{X}}$ is not exactly equal to the true underlying population mean, $\boldsymbol{\mu}$, the least squares estimate, $\hat{\boldsymbol{\beta}}$ will not be exactly equal to the "true" underlying value of $\boldsymbol{\beta}$ that we are trying to estimate (under the assumption that the regression model is correct), and will depend on the particular data set used to compute it. However, there is good reason to think that $\hat{\boldsymbol{\beta}}$ will be a "good" estimate of $\boldsymbol{\beta}$, as the following result makes clear.

Proposition 24 *For the regression model*

$$\boldsymbol{y} = \mathsf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

and least squares estimator

$$\hat{\boldsymbol{\beta}} = (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y},$$

if we are prepared to make the modelling assumptions that under repeated sampling of $\boldsymbol{y}$ we have $\mathrm{E}(\boldsymbol{\varepsilon}) = \mathbf{0}$ and $\mathrm{Var}(\boldsymbol{\varepsilon}) = \sigma^2\,\mathrm{I}$ for some σ, it follows that

$$\mathrm{E}\left(\hat{\boldsymbol{\beta}}\right) = \boldsymbol{\beta} \text{ and } \mathrm{Var}\left(\hat{\boldsymbol{\beta}}\right) = \sigma^2(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}.$$

From this we can immediately conclude that $\hat{\boldsymbol{\beta}}$ is an unbiased *estimator of $\boldsymbol{\beta}$. Further, since the elements of $\mathsf{X}^\mathsf{T}\mathsf{X}$ increase with n, it is relatively easy to show that $\mathrm{Var}\left(\hat{\boldsymbol{\beta}}\right) \longrightarrow 0$ as $n \longrightarrow \infty$, and hence that $\hat{\boldsymbol{\beta}}$ is also a* consistent *estimator of $\boldsymbol{\beta}$.*

Proof

First note that

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y}, \\
&= (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}(\mathsf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}) \\
&= \boldsymbol{\beta} + (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{\varepsilon}.
\end{aligned}$$

From this we get $\mathrm{E}\left(\hat{\boldsymbol{\beta}}\right) = \boldsymbol{\beta}$ and

$$\begin{aligned}\mathrm{Var}\left(\hat{\boldsymbol{\beta}}\right) &= \mathrm{Var}\left((\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{\varepsilon}\right) \\ &= (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\,\mathrm{Var}(\boldsymbol{\varepsilon})\,\mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1} \\ &= \sigma^2(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\end{aligned}$$

□

3.2.2 The general linear model

Now consider the obvious generalisation of the previous case, where our "output", $\boldsymbol{y}_i$ is now a q-vector, rather than a scalar. We then have the model

$$\boldsymbol{y}_i = \mathsf{B}^\mathsf{T}\boldsymbol{x}_i + \boldsymbol{\varepsilon}_i,\ i = 1, 2, \ldots n,$$

where B is now a $p \times q$ matrix. We can write this in matrix form as

$$\mathsf{Y} = \mathsf{XB} + \mathsf{E},$$

where E is an $n \times q$ matrix of "errors", and this is typically known as the general linear model. The regression problem here is, given data matrices X and Y, to find the matrix B which makes the Frobenius norm of the error matrix, $\|E\|$, as small as possible. Note that

$$\|E\|^2 = \mathrm{Tr}\left(\mathsf{E}^\mathsf{T}\mathsf{E}\right) = \sum_{i=1}^{n}\sum_{j=1}^{q}\varepsilon_{ij}^2,$$

and so this is again a least squares problem, where we choose B to minimise the sum of squares of the errors. We proceed as before by noting that

$$\begin{aligned}\mathrm{Tr}\left(\mathsf{E}^\mathsf{T}\mathsf{E}\right) &= \mathrm{Tr}\left([\mathsf{Y} - \mathsf{XB}]^\mathsf{T}[\mathsf{Y} - \mathsf{XB}]\right) \\ &= \mathrm{Tr}\left(\mathsf{Y}^\mathsf{T}\mathsf{Y} - \mathsf{Y}^\mathsf{T}\mathsf{XB} - \mathsf{B}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{Y} + \mathsf{B}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{XB}\right) \\ &= \mathrm{Tr}\left(\mathsf{Y}^\mathsf{T}\mathsf{Y}\right) - \mathrm{Tr}\left(\mathsf{Y}^\mathsf{T}\mathsf{XB}\right) - \mathrm{Tr}\left(\mathsf{B}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{Y}\right) + \mathrm{Tr}\left(\mathsf{B}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{XB}\right) \\ &= \mathrm{Tr}\left(\mathsf{Y}^\mathsf{T}\mathsf{Y}\right) - 2\,\mathrm{Tr}\left(\mathsf{Y}^\mathsf{T}\mathsf{XB}\right) + \mathrm{Tr}\left(\mathsf{B}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{XB}\right).\end{aligned}$$

Now differentiating wrt the matrix B we get

$$\frac{\partial}{\partial \mathsf{B}}\,\mathrm{Tr}\left(\mathsf{E}^\mathsf{T}\mathsf{E}\right) = -2\mathsf{X}^\mathsf{T}\mathsf{Y} + 2\mathsf{X}^\mathsf{T}\mathsf{XB},$$

where here we used results 3. and 4. of Proposition 23. Equating to zero gives a matrix version of the normal equations,

$$\mathsf{X}^\mathsf{T}\mathsf{X}\hat{\mathsf{B}} = \mathsf{X}^\mathsf{T}\mathsf{Y}.$$

This has mathematical solution

$$\hat{\mathsf{B}} = (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\mathsf{Y}.$$

By considering the columns of $\hat{\mathsf{B}}$ and Y, we see that this corresponds to the usual multiple linear regression solution for each column of Y. That is, the multivariate regression problem is solved by separately regressing each column of Y on the data matrix X. However,

the full problem will be solved more efficiently and stably by directly solving the above normal equations using a QR factorisation of X (with multiple RHSs). To be clear, if we have the QR factorisation of X as $X = QR$, substituting in to the normal equations leads directly to

$$R\hat{B} = Q^T Y.$$

These equations may be solved for $\hat{B}$ as a triangular system with multiple RHSs (generally to be preferred), or explicitly solved as

$$\hat{B} = R^{-1}Q^T Y.$$

Example: microarray data

To keep things simple, let us consider using the first 5 genes in the `nci` dataset in order to predict the next 3. We therefore wish to compute $\hat{B}$, which will be a 5×3 matrix, if we do not include an intercept term, and will be 6×3 if we do, with the first row corresponding to a location shift. Recall that the `nci` data is stored as a matrix with rows representing the genes. We first solve using `lm()`, with and without an intercept

```
> X=t(nci[1:5,])
> Y=t(nci[6:8,])
> lm(Y~X)

Call:
lm(formula = Y ~ X)

Coefficients:
             [,1]       [,2]       [,3]
(Intercept)  0.005571   0.014090  -0.031221
X1           0.055359   0.011646   0.327704
X2           0.069211   0.163145  -0.031904
X3          -0.059708  -0.028433  -0.025860
X4           0.004978  -0.017083  -0.038912
X5           0.175285   0.160014   0.005762

> lm(Y~0+X)

Call:
lm(formula = Y ~ 0 + X)

Coefficients:
    [,1]        [,2]        [,3]
X1  0.0550853   0.0109542   0.3292373
X2  0.0694000   0.1636241  -0.0329654
X3 -0.0595292  -0.0279796  -0.0268643
X4  0.0034643  -0.0209107  -0.0304312
X5  0.1762379   0.1624230   0.0004232
```

We now solve directly using the QR factorisation, first without an intercept, and then with. Note how the `backsolve()` function solves multiple RHSs exactly as we would wish.

```
> QR=qr(X)
> backsolve(qr.R(QR),t(qr.Q(QR))%*%Y)
             [,1]         [,2]          [,3]
[1,]  0.055085288  0.01095420  0.3292372739
[2,]  0.069399969  0.16362409 -0.0329654306
[3,] -0.059529200 -0.02797956 -0.0268642629
[4,]  0.003464287 -0.02091073 -0.0304311643
[5,]  0.176237890  0.16242302  0.0004231837
> X=cbind(rep(1,64),X)
> QR=qr(X)
> backsolve(qr.R(QR),t(qr.Q(QR))%*%Y)
             [,1]         [,2]         [,3]
[1,]  0.005570868  0.01408965 -0.031220986
[2,]  0.055358790  0.01164593  0.327704478
[3,]  0.069210662  0.16314530 -0.031904493
[4,] -0.059708338 -0.02843263 -0.025860316
[5,]  0.004977558 -0.01708341 -0.038912041
[6,]  0.175285335  0.16001385  0.005761614
```

3.2.3 Weighted errors

One possible concern with the above solution could be that it appears to treat all errors with equal weight, even though some outputs may be measured on different scales, and some outputs may also be correlated. If we know the variance matrix associated with the errors,

$$\mathrm{Var}(\boldsymbol{\varepsilon}_i) = \Sigma,$$

we can use the symmetric square root to form the Mahalanobis transformation of the error matrix

$$\mathsf{E}' = \mathsf{E}\Sigma^{-1/2},$$

and instead minimise

$$\mathrm{Tr}\left(\mathsf{E}'^{\mathsf{T}}\mathsf{E}'\right) = \mathrm{Tr}\left(\Sigma^{-1/2}\mathsf{E}^{\mathsf{T}}\mathsf{E}\Sigma^{-1/2}\right) = \mathrm{Tr}\left(\Sigma^{-1}\mathsf{E}^{\mathsf{T}}\mathsf{E}\right).$$

We now note that

$$\begin{aligned}
\mathrm{Tr}\left(\Sigma^{-1}\mathsf{E}^{\mathsf{T}}\mathsf{E}\right) &= \mathrm{Tr}\left(\Sigma^{-1}[\mathsf{Y}-\mathsf{XB}]^{\mathsf{T}}[\mathsf{Y}-\mathsf{XB}]\right) \\
&= \mathrm{Tr}\left(\Sigma^{-1}\mathsf{Y}^{\mathsf{T}}\mathsf{Y} - \Sigma^{-1}\mathsf{Y}^{\mathsf{T}}\mathsf{XB} - \Sigma^{-1}\mathsf{B}^{\mathsf{T}}\mathsf{X}^{\mathsf{T}}\mathsf{Y} + \Sigma^{-1}\mathsf{B}^{\mathsf{T}}\mathsf{X}^{\mathsf{T}}\mathsf{XB}\right) \\
&= \mathrm{Tr}\left(\Sigma^{-1}\mathsf{Y}^{\mathsf{T}}\mathsf{Y}\right) - \mathrm{Tr}\left(\Sigma^{-1}\mathsf{Y}^{\mathsf{T}}\mathsf{XB}\right) - \mathrm{Tr}\left(\Sigma^{-1}\mathsf{B}^{\mathsf{T}}\mathsf{X}^{\mathsf{T}}\mathsf{Y}\right) + \mathrm{Tr}\left(\Sigma^{-1}\mathsf{B}^{\mathsf{T}}\mathsf{X}^{\mathsf{T}}\mathsf{XB}\right) \\
&= \mathrm{Tr}\left(\Sigma^{-1}\mathsf{Y}^{\mathsf{T}}\mathsf{Y}\right) - 2\,\mathrm{Tr}\left(\Sigma^{-1}\mathsf{Y}^{\mathsf{T}}\mathsf{XB}\right) + \mathrm{Tr}\left(\Sigma^{-1}\mathsf{B}^{\mathsf{T}}\mathsf{X}^{\mathsf{T}}\mathsf{XB}\right).
\end{aligned}$$

Now differentiating wrt the matrix B we get

$$\frac{\partial}{\partial \mathsf{B}}\,\mathrm{Tr}\left(\Sigma^{-1}\mathsf{E}^{\mathsf{T}}\mathsf{E}\right) = -2\mathsf{X}^{\mathsf{T}}\mathsf{Y}\Sigma^{-1} + 2\mathsf{X}^{\mathsf{T}}\mathsf{XB}\Sigma^{-1},$$

where here we used results 3. and 5. of Proposition 23. Equating to zero gives the same normal equations as previously, since the variance matrix cancels out. Therefore the weighted solution is in fact the same as the unweighted solution, and appropriate irrespective of any covariance structure associated with the errors.

3.2.4 Understanding regression and the general linear model

We now know that we can fit the general linear model

$$\mathsf{Y} = \mathsf{X}\mathsf{B} + \mathsf{E}$$

for given $n \times p$ covariate matrix X and $n \times q$ output matrix Y by solving the normal equations

$$\mathsf{X}^\mathsf{T}\mathsf{X}\hat{\mathsf{B}} = \mathsf{X}^\mathsf{T}\mathsf{Y}$$

for the $p \times q$ matrix $\hat{\mathsf{B}}$ representing the least squares solution. Let us now try to understand this geometrically. To do this we define $\hat{\mathsf{Y}} = \mathsf{X}\hat{\mathsf{B}}$, the $n \times q$ matrix of *fitted values*, and $\hat{\mathsf{E}} = \mathsf{Y} - \hat{\mathsf{Y}}$, the $n \times q$ matrix of *residuals*.

Proposition 25 *The p columns of X are orthogonal to the q columns of $\hat{\mathsf{E}}$. In other words,*

$$\mathsf{X}^\mathsf{T}\hat{\mathsf{E}} = 0,$$

the $p \times q$ zero matrix.

Proof

$$\begin{aligned}\mathsf{X}^\mathsf{T}\hat{\mathsf{E}} &= \mathsf{X}^\mathsf{T}(\mathsf{Y} - \hat{\mathsf{Y}}) \\ &= \mathsf{X}^\mathsf{T}(\mathsf{Y} - \mathsf{X}\hat{\mathsf{B}}) \\ &= \mathsf{X}^\mathsf{T}\mathsf{Y} - \mathsf{X}^\mathsf{T}\mathsf{X}\hat{\mathsf{B}} \\ &= 0\end{aligned}$$

by the normal equations. □

Proposition 26

$$\hat{\mathsf{Y}} = \mathsf{M}\mathsf{Y},$$

where

$$\mathsf{M} = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}.$$

The $n \times n$ matrix M is known as the hat matrix, since is puts the "hat" on Y.

Proof

$$\hat{\mathsf{Y}} = \mathsf{X}\hat{\mathsf{B}} = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\mathsf{Y} = \mathsf{M}\mathsf{Y},$$

since

$$\hat{\mathsf{B}} = (\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\mathsf{Y}.$$

□

Proposition 27 *The hat matrix*

$$\mathsf{M} = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}$$

is symmetric and idempotent.

Proof

Symmetry:

$$\mathsf{M}^\mathsf{T} = [\mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}]^\mathsf{T} = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-\mathsf{T}}\mathsf{X}^\mathsf{T} = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T} = \mathsf{M},$$

since $\mathsf{X}^\mathsf{T}\mathsf{X}$ is symmetric.

Idempotency:

$$\mathsf{M}^2 = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T} = \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T} = \mathsf{M}.$$

□

Proposition 28 *If we write*

$$\mathsf{X} = \mathsf{Q}\mathsf{R}$$

for the QR*–factorisation of* X*, when we have*

$$\mathsf{M} = \mathsf{Q}\mathsf{Q}^\mathsf{T},$$

from which it is clear that M *represents orthogonal projection onto the subspace of* $\mathbb{R}^n$ *spanned by the columns of* X*.*

Proof

$$\begin{aligned}\mathsf{M} &= \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T} \\ &= \mathsf{Q}\mathsf{R}(\mathsf{R}^\mathsf{T}\mathsf{Q}^\mathsf{T}\mathsf{Q}\mathsf{R})^{-1}\mathsf{R}^\mathsf{T}\mathsf{Q}^\mathsf{T} \\ &= \mathsf{Q}\mathsf{R}\mathsf{R}^{-1}\mathsf{R}^{-\mathsf{T}}\mathsf{R}^\mathsf{T}\mathsf{Q}^\mathsf{T} \\ &= \mathsf{Q}\mathsf{Q}^\mathsf{T}.\end{aligned}$$

□

Once we understand that the hat matrix is orthogonal projection, it should be clear that any n-vector in the column span of X will be left invariant under M. However, we can verify this directly without exploiting the QR–factorisation.

Proposition 29 *Any* n*-vector* $\boldsymbol{v}$ *in the column span of* X *is left invariant under* M*. In other words,*

$$\mathsf{M}\boldsymbol{v} = \boldsymbol{v}.$$

Proof

By definition, $\boldsymbol{v} = \mathsf{X}\boldsymbol{\beta}$ for some p-vector $\boldsymbol{\beta}$ (this is what it means to be in the column span of X). But then

$$\begin{aligned}\mathsf{M}\boldsymbol{v} &= \mathsf{M}\mathsf{X}\boldsymbol{\beta} \\ &= \mathsf{X}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\mathsf{X}\boldsymbol{\beta} \\ &= \mathsf{X}\boldsymbol{\beta} \\ &= \boldsymbol{v}.\end{aligned}$$

□

We know that to include an intercept term in the model, we just make the first column of X equal to $\mathbf{1}_n$. In this case, $\mathbf{1}_n$ is certainly in the column span of X, and so we must have $\mathsf{M}\mathbf{1}_n = \mathbf{1}_n$ (and this can be verified directly, since $\mathbf{1}_n = \mathsf{X}\boldsymbol{e}_1$). The reason for including $\mathbf{1}_n$ in X is made clear below.

Proposition 30 *Let $\bar{e}$ be the sample mean of $\hat{\mathsf{E}}$. If $\mathbf{1}_n$ is in the column span of X, then*

$$\bar{\boldsymbol{e}} = \mathbf{0}.$$

From this it immediately follows that the sample mean of $\hat{\mathsf{Y}}$ equals the sample mean of Y.

Proof

$$\begin{aligned}
\bar{\boldsymbol{e}} &= \frac{1}{n}\hat{\mathsf{E}}^{\mathsf{T}}\mathbf{1}_n \\
&= \frac{1}{n}(\mathsf{Y} - \hat{\mathsf{Y}})^{\mathsf{T}}\mathbf{1}_n \\
&= \frac{1}{n}(\mathsf{Y} - \mathsf{M}\mathsf{Y})^{\mathsf{T}}\mathbf{1}_n \\
&= \frac{1}{n}[(\mathrm{I}_n - \mathsf{M})\mathsf{Y}]^{\mathsf{T}}\mathbf{1}_n \\
&= \frac{1}{n}\mathsf{Y}^{\mathsf{T}}(\mathrm{I}_n - \mathsf{M})^{\mathsf{T}}\mathbf{1}_n \\
&= \frac{1}{n}\mathsf{Y}^{\mathsf{T}}(\mathrm{I}_n - \mathsf{M})\mathbf{1}_n \\
&= \mathbf{0},
\end{aligned}$$

since $(\mathrm{I}_n - \mathsf{M})\mathbf{1}_n = \mathbf{0}$. □

Proposition 31

$$\mathsf{Y}^{\mathsf{T}}\mathsf{Y} = \hat{\mathsf{Y}}^{\mathsf{T}}\hat{\mathsf{Y}} + \hat{\mathsf{E}}^{\mathsf{T}}\hat{\mathsf{E}}.$$

Proof

Clearly $\mathsf{Y}^{\mathsf{T}}\mathsf{Y} = (\hat{\mathsf{Y}} + \hat{\mathsf{E}})^{\mathsf{T}}(\hat{\mathsf{Y}} + \hat{\mathsf{E}}) = \hat{\mathsf{Y}}^{\mathsf{T}}\hat{\mathsf{Y}} + \hat{\mathsf{E}}^{\mathsf{T}}\hat{\mathsf{E}} + \hat{\mathsf{E}}^{\mathsf{T}}\hat{\mathsf{Y}} + \hat{\mathsf{Y}}^{\mathsf{T}}\hat{\mathsf{E}}$. Now

$$\begin{aligned}
\hat{\mathsf{E}}^{\mathsf{T}}\hat{\mathsf{Y}} &= (\mathsf{Y} - \hat{\mathsf{Y}})^{\mathsf{T}}\hat{\mathsf{Y}} \\
&= \mathsf{Y}^{\mathsf{T}}\hat{\mathsf{Y}} - \hat{\mathsf{Y}}^{\mathsf{T}}\hat{\mathsf{Y}} \\
&= \mathsf{Y}^{\mathsf{T}}\mathsf{M}\mathsf{Y} - (\mathsf{M}\mathsf{Y})^{\mathsf{T}}\mathsf{M}\mathsf{Y} \\
&= \mathsf{Y}^{\mathsf{T}}\mathsf{M}\mathsf{Y} - \mathsf{Y}^{\mathsf{T}}\mathsf{M}^{\mathsf{T}}\mathsf{M}\mathsf{Y} \\
&= \mathsf{Y}^{\mathsf{T}}\mathsf{M}\mathsf{Y} - \mathsf{Y}^{\mathsf{T}}\mathsf{M}^{2}\mathsf{Y} \\
&= \mathsf{Y}^{\mathsf{T}}\mathsf{M}\mathsf{Y} - \mathsf{Y}^{\mathsf{T}}\mathsf{M}\mathsf{Y} \\
&= 0.
\end{aligned}$$

Similarly, $\hat{\mathsf{Y}}^{\mathsf{T}}\hat{\mathsf{E}} = (\hat{\mathsf{E}}^{\mathsf{T}}\hat{\mathsf{Y}})^{\mathsf{T}} = 0^{\mathsf{T}} = 0$, and so

$$\mathsf{Y}^{\mathsf{T}}\mathsf{Y} = \hat{\mathsf{Y}}^{\mathsf{T}}\hat{\mathsf{Y}} + \hat{\mathsf{E}}^{\mathsf{T}}\hat{\mathsf{E}}.$$

□

Proposition 32 *If $\mathbf{1}_n$ is in the column span of X, then sample variance can be decomposed as*

$$\mathsf{S}_\mathsf{Y} = \mathsf{S}_{\hat{\mathsf{Y}}} + \mathsf{S}_{\hat{\mathsf{E}}}.$$

From this it follows that $t(\mathsf{Y}) = t(\hat{\mathsf{Y}}) + t(\hat{\mathsf{E}})$, and so $t(\hat{\mathsf{Y}})/t(\mathsf{Y})$ is the proportion of total variation explained by the regression. The proportion of variation explained by a statistical model is often referred to as R^2.

Proof

$$\begin{aligned}
\mathsf{S}_\mathsf{Y} &= \frac{1}{n-1}\mathsf{Y}^\mathsf{T}\mathsf{H}_n\mathsf{Y} \\
&= \frac{1}{n-1}(\hat{\mathsf{Y}} + \hat{\mathsf{E}})^\mathsf{T}\mathsf{H}_n(\hat{\mathsf{Y}} + \hat{\mathsf{E}}) \\
&= \frac{1}{n-1}\left[\hat{\mathsf{Y}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{Y}} + \hat{\mathsf{E}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{E}} + \hat{\mathsf{E}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{Y}} + \hat{\mathsf{Y}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{E}}\right] \\
&= \mathsf{S}_{\hat{\mathsf{Y}}} + \mathsf{S}_{\hat{\mathsf{E}}} + \frac{1}{n-1}\left[\hat{\mathsf{E}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{Y}} + \hat{\mathsf{Y}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{E}}\right].
\end{aligned}$$

Now $\hat{\mathsf{Y}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{E}} = \hat{\mathsf{Y}}^\mathsf{T}\hat{\mathsf{E}} = 0$, since $\mathsf{H}_n\hat{\mathsf{E}} = \hat{\mathsf{E}}$, and similarly $\hat{\mathsf{E}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{Y}} = (\hat{\mathsf{Y}}^\mathsf{T}\mathsf{H}_n\hat{\mathsf{E}})^\mathsf{T} = 0$, so

$$\mathsf{S}_\mathsf{Y} = \mathsf{S}_{\hat{\mathsf{Y}}} + \mathsf{S}_{\hat{\mathsf{E}}}.$$

□

3.3 The multivariate normal (MVN) distribution

Many courses on multivariate statistics introduce the multivariate normal distribution very early, and then make extensive use of the properties of this distribution. The problem with such an approach is that it gives the impression that almost all of multivariate statistical theory is dependent on an assumption of multivariate normality, and as we have seen, this is not the case at all. However, the MVN distribution is the most important non-trivial multivariate distribution in statistics, and has many interesting applications. It is therefore appropriate at this point in the course to consider it, and re-consider some of the results of this chapter in the context of MVN data.

We begin by defining multivariate normal random quantities to be affine transformations of standard normal random vectors.

Definition 9 *Let $\boldsymbol{Z} = (Z_1, Z_2, \ldots, Z_q)^\mathsf{T}$, where the components Z_i are iid $N(0,1)$. Then for a $p \times q$ matrix A and p-vector $\boldsymbol{\mu}$, we say that*

$$\boldsymbol{X} = \mathsf{A}\boldsymbol{Z} + \boldsymbol{\mu}$$

has a multivariate normal distribution *with mean $\boldsymbol{\mu}$ and variance matrix $\Sigma = \mathsf{A}\mathsf{A}^\mathsf{T}$, and we write*

$$\boldsymbol{X} \sim N(\boldsymbol{\mu}, \Sigma).$$

There are several issues with this definition which require clarification. The first thing to note is that the expectation and variance of $\boldsymbol{X}$ do indeed correspond to $\boldsymbol{\mu}$ and Σ, using properties of affine transformations that we considered in Chapter 1. The next concern is whether the distribution is really characterised by its mean and variance, since there are many possible affine transformations that will lead to the same mean and variance. In order to keep things simple, we will begin by considering only *invertible* transformations A. In this case, we must have $p = q$.

Proposition 33 *The density of a p-dimensional MVN random vector $\boldsymbol{X}$, with mean $\boldsymbol{\mu}$ and positive definite variance Σ is given by*

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

Proof

We choose any invertible $p \times p$ matrix A such that $\mathsf{A}\mathsf{A}^{\mathsf{T}} = \Sigma$, and put

$$\boldsymbol{X} = \mathsf{A}\boldsymbol{Z} + \boldsymbol{\mu}.$$

We first need to know the probability density function for $\boldsymbol{Z}$, which is clearly given by

$$\begin{aligned}
\phi(\boldsymbol{z}) &= \prod_{i=1}^{p}\phi(z_i) \\
&= \prod_{i=1}^{p}\frac{1}{\sqrt{2\pi}}\exp\{-z_i^2/2\} \\
&= (2\pi)^{-p/2}\exp\left\{-\frac{1}{2}\sum_{i=1}^{p}z_i^2\right\} \\
&= (2\pi)^{-p/2}\exp\left\{-\frac{1}{2}\boldsymbol{z}^{\mathsf{T}}\boldsymbol{z}\right\}.
\end{aligned}$$

Now since $\boldsymbol{X}$ is an invertible transformation of $\boldsymbol{Z}$, its density is given by

$$f(\boldsymbol{x}) = \phi(\boldsymbol{z})\left|\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right|,$$

where $\boldsymbol{z} = \mathsf{A}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$. But now

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \mathsf{A}^{-1},$$

and so

$$\left|\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right| = |\mathsf{A}^{-1}| = |\mathsf{A}|^{-1} = |\Sigma|^{-1/2},$$

giving

$$\begin{aligned}
f(\boldsymbol{x}) &= (2\pi)^{-p/2}\exp\left\{-\frac{1}{2}[\mathsf{A}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})]^{\mathsf{T}}[\mathsf{A}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})]\right\}|\Sigma|^{-1/2} \\
&= (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\}.
\end{aligned}$$

□

The crucial thing to note here is that despite the fact that we considered *any* (invertible) A leading to a required variance matrix, the density of the resulting random quantity depends only on the variance matrix (and mean vector), and not on the particular affine transformation used in the construction. Consideration of non-invertible (and non-square) transformations complicates the analysis, but essentially the same conclusion holds, which is that the distribution of the MVN random quantity depends only on its mean and variance, and not on the particular affine transformation used to construct it. It is therefore reasonable to characterise MVN random quantities by their mean and variance.

3.3.1 Evaluation of the MVN density

In some applications it is necessary to evaluate the MVN density. However, in most applications, it is sufficient to evaluate the log of the density, and this is usually safer, as it is less susceptible to numerical underflow. Clearly the log-density is given by

$$l(\boldsymbol{x}) = -\frac{p}{2}\log 2\pi - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}).$$

Note that the final term is essentially the Mahalanobis distance. We can avoid direct computation of $|\Sigma|$ and Σ^{-1} by using an appropriate matrix decomposition. In the positive definite case, the Cholesky decomposition usually provides the best solution, so suppose that Σ has Cholesky decomposition

$$\Sigma = \mathsf{G}\mathsf{G}^{\mathsf{T}},$$

where G is lower triangular. To see how this helps, first consider the determinant term

$$\frac{1}{2}\log|\Sigma| = \log|\Sigma|^{1/2} = \log|\mathsf{G}|.$$

But since G is lower triangular, its determinant is the product of its diagonal elements, and so the log of this is given by

$$\log|\mathsf{G}| = \sum_{i=1}^{p}\log g_{ii}.$$

Similarly, the final quadratic form can be simplified as

$$\begin{aligned}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) &= (\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}(\mathsf{G}\mathsf{G}^{\mathsf{T}})^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\\ &= (\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\mathsf{G}^{-\mathsf{T}}\mathsf{G}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\\ &= \boldsymbol{z}^{\mathsf{T}}\boldsymbol{z},\end{aligned}$$

where $\boldsymbol{z} = \mathsf{G}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$, and hence may be found by forward solving the triangular system

$$\mathsf{G}\boldsymbol{z} = (\boldsymbol{x}-\boldsymbol{\mu}).$$

Given this solution, we can therefore write the log density as

$$l(\boldsymbol{x}) = -\frac{p}{2}\log 2\pi - \sum_{i=1}^{p}\log g_{ii} - \frac{1}{2}\sum_{i=1}^{p} z_i^2.$$

This is an efficient way to compute the log density, as it just requires a Cholesky decomposition and a single forward solve.

3.3.2 Properties of the MVN

We now consider some very useful properties of the MVN distribution.

Proposition 34 (Affine transformation) *If* $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \Sigma)$*, and*

$$\boldsymbol{Y} = \mathrm{A}\boldsymbol{X} + \boldsymbol{b},$$

then $\boldsymbol{Y}$ *is multivariate normal, and* $\boldsymbol{Y} \sim N(\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathrm{A}\Sigma\mathrm{A}^{\mathsf{T}})$.

Proof

The mean and variance of $\boldsymbol{Y}$ are clear from our understanding of affine transformations of random vectors. The key to this result is understanding why $\boldsymbol{Y}$ is MVN. For this, note that we need to show that $\boldsymbol{Y}$ is an affine transformation of a standard normal vector. But since $\boldsymbol{X}$ is MVN, there must exist a matrix M such that

$$\boldsymbol{X} = \mathrm{M}\boldsymbol{Z} + \boldsymbol{\mu},$$

where $\boldsymbol{Z}$ is a standard normal vector. Then

$$\begin{aligned}\boldsymbol{Y} &= \mathrm{A}\boldsymbol{X} + \boldsymbol{b} \\ &= \mathrm{A}(\mathrm{M}\boldsymbol{Z} + \boldsymbol{\mu}) + \boldsymbol{b} \\ &= (\mathrm{AM})\boldsymbol{Z} + (\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}),\end{aligned}$$

and so $\boldsymbol{Y}$ is also an affine transformation of a standard normal vector, and hence MVN. □

For example, referring back to Proposition 24, if we assume that $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}, \sigma^2 \mathrm{I})$, then since $\hat{\boldsymbol{\beta}}$ is an affine transformation of $\boldsymbol{\varepsilon}$, we will have that $\hat{\boldsymbol{\beta}} \sim N(\boldsymbol{\beta}, \sigma^2(\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1})$.

Now since marginal distributions correspond to linear transformations, it follows that all marginal distributions of the MVN are (multivariate) normal.

Proposition 35 *If* $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \Sigma)$ *is partitioned as*

$$\begin{pmatrix}\boldsymbol{X}_1 \\ \boldsymbol{X}_2\end{pmatrix} \sim N\left(\begin{pmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{pmatrix}, \begin{pmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22}\end{pmatrix}\right),$$

then the marginal distribution of $\boldsymbol{X}_1$ *is given by*

$$\boldsymbol{X}_1 \sim N(\boldsymbol{\mu}_1, \Sigma_{11}).$$

Proof

The result follows simply by noting that

$$\boldsymbol{X}_1 = (\mathrm{I},\ 0)\begin{pmatrix}\boldsymbol{X}_1 \\ \boldsymbol{X}_2\end{pmatrix},$$

and using properties of linear transformations. □

Similarly, univariate marginals may be computed using the fact that $X_i = \boldsymbol{e}_i{}^{\mathsf{T}}\boldsymbol{X}$, giving $X_i \sim N(\mu_i, \sigma_{ii})$.

Proposition 36 *If* $\boldsymbol{X}_1 \sim N(\boldsymbol{\mu_1}, \Sigma_1)$ *and* $\boldsymbol{X}_2 \sim N(\boldsymbol{\mu}_2, \Sigma_2)$ *are independent, then*

$$\boldsymbol{X}_1 + \boldsymbol{X}_2 \sim N(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2, \Sigma_1 + \Sigma_2).$$

Proof

First note that $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are jointly MVN, with

$$\begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \right).$$

Then observing that $\boldsymbol{X}_1 + \boldsymbol{X}_2$ corresponds to the linear transformation

$$(\mathrm{I},\ \mathrm{I}) \begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{pmatrix}$$

leads directly to the result. □

Finally, we state without proof the key result concerning conditional distributions of the MVN. The proof is straightforward, but somewhat involved.

Proposition 37 *If* $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \Sigma)$ *is partitioned as*

$$\begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \right),$$

then the conditional distribution of $\boldsymbol{X}_1$ *given that* $\boldsymbol{X}_2$ *is observed to be* $\boldsymbol{x}_2$ *is MVN, with*

$$(\boldsymbol{X}_1 | \boldsymbol{X}_2 = \boldsymbol{x}_2) \sim N(\boldsymbol{\mu}_{1|2}, \Sigma_{1|2}),$$

where

$$\begin{aligned} \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \Sigma_{12}{\Sigma_{22}}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \\ \Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12}{\Sigma_{22}}^{-1}\Sigma_{21}. \end{aligned}$$

Note that the simple version of the result presented here assumes that Σ_{22} is invertible, but it generalises straightforwardly to the case of singular Σ_{22}. We will examine (and prove) a different version of this result later, in the Chapter on graphical models.

3.3.3 Maximum likelihood estimation

If we consider an $n \times p$ data matrix X with iid rows $\boldsymbol{X}_i \sim N(\boldsymbol{\mu}, \Sigma)$, then the likelihood for an observed data matrix is given by

$$\begin{aligned} L(\boldsymbol{\mu}, \Sigma; \mathsf{X}) &= \prod_{i=1}^{n} f(\boldsymbol{x}_i) \\ &= \prod_{i=1}^{n} (2\pi)^{-p/2} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} (\boldsymbol{x}_i - \boldsymbol{\mu})^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}) \right\} \\ &= (2\pi)^{-np/2} |\Sigma|^{-n/2} \exp\left\{ -\frac{1}{2} \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu})^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}) \right\}. \end{aligned}$$

Consequently, the log-likelihood is given by

$$l(\boldsymbol{\mu}, \Sigma; \mathsf{X}) = -\frac{np}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}).$$

Note that if we need to numerically compute this log-likelihood, we would do so using the Cholesky decomposition, as already described for the log-density. We can use the log-likelihood in order to compute maximum likelihood estimators for the MVN.

Proposition 38 *For an MVN model, the maximum likelihood estimator of the mean, $\boldsymbol{\mu}$ is given by*

$$\hat{\boldsymbol{\mu}} = \bar{\boldsymbol{x}}.$$

Proof

We proceed by differentiating the log-likelihood function wrt $\boldsymbol{\mu}$ and equating to zero. First note that

$$\frac{\partial l}{\partial \boldsymbol{\mu}} = -\frac{1}{2}\sum_{i=1}^{n}\frac{\partial}{\partial \boldsymbol{\mu}}(\boldsymbol{x}_i - \boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}).$$

Now

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\mu}}(\boldsymbol{x}_i - \boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}) &= \frac{\partial}{\partial \boldsymbol{\mu}}(\boldsymbol{x}_i{}^{\mathsf{T}}\Sigma^{-1}\boldsymbol{x}_i - 2\boldsymbol{\mu}^{\mathsf{T}}\Sigma^{-1}\boldsymbol{x}_i + \boldsymbol{\mu}^{\mathsf{T}}\Sigma^{-1}\boldsymbol{\mu}) \\ &= -2\Sigma^{-1}\boldsymbol{x}_i + 2\Sigma^{-1}\boldsymbol{\mu} \\ &= 2\Sigma^{-1}(\boldsymbol{\mu} - \boldsymbol{x}_i),\end{aligned}$$

using properties 1. and 2. of Proposition 23. So now

$$\begin{aligned}\frac{\partial l}{\partial \boldsymbol{\mu}} &= -\frac{1}{2}\sum_{i=1}^{n} 2\Sigma^{-1}(\boldsymbol{\mu} - \boldsymbol{x}_i) \\ &= \Sigma^{-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu}) \\ &= \Sigma^{-1}(n\bar{\boldsymbol{x}} - n\boldsymbol{\mu}) \\ &= n\Sigma^{-1}(\bar{\boldsymbol{x}} - \boldsymbol{\mu}),\end{aligned}$$

and so equating to zero gives the result. □

Proposition 39 *For an MVN model, the maximum likelihood estimator of the variance matrix, Σ is given by*

$$\hat{\Sigma} = \frac{n-1}{n}\mathsf{S}.$$

Consequently, the maximum likelihood estimator is biased, but asymptotically unbiased, and consistent.

Proof

We will proceed by using matrix differentiation to find a matrix Σ which maximises the likelihood.

$$\begin{aligned}\frac{\partial l}{\partial \Sigma} &= -\frac{n}{2}\frac{\partial}{\partial \Sigma}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}\frac{\partial}{\partial \Sigma}(\boldsymbol{x}_i-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}) \\ &= -\frac{n}{2}\Sigma^{-1} + \frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu})(\boldsymbol{x}_i-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1} \\ &= -\frac{n}{2}\Sigma^{-1} + \frac{1}{2}\Sigma^{-1}\left[\sum_{i=1}^{n}(\boldsymbol{x}_i-\boldsymbol{\mu})(\boldsymbol{x}_i-\boldsymbol{\mu})^{\mathsf{T}}\right]\Sigma^{-1},\end{aligned}$$

using properties 7. and 9. of Proposition 23. Equating to zero gives

$$\begin{aligned}n\hat{\Sigma}^{-1} &= \hat{\Sigma}^{-1}\left[\sum_{i=1}^{n}(\boldsymbol{x}_i-\hat{\boldsymbol{\mu}})(\boldsymbol{x}_i-\hat{\boldsymbol{\mu}})^{\mathsf{T}}\right]\hat{\Sigma}^{-1} \\ \Rightarrow \hat{\Sigma} &= \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^{\mathsf{T}} \\ &= \frac{n-1}{n}\mathsf{S}.\end{aligned}$$

□

The maximum likelihood estimator of Σ therefore uses a divisor of n, rather than $n-1$, as required to obtain an unbiased estimate. Apart from this minor issue, we see that the sample estimates $\bar{\boldsymbol{x}}$ and S of the population quantities $\boldsymbol{\mu}$ and Σ, which we have already shown to be effective without making any distributional assumptions, may also be interpreted as maximum likelihood estimators in cases where an MVN distributional assumption is felt to be appropriate.

3.3.4 MLE for the general linear model

Let us now reconsider the general linear model

$$\mathsf{Y} = \mathsf{XB} + \mathsf{E},$$

where the observations are modelled as

$$\boldsymbol{y}_i = \mathsf{B}^{\mathsf{T}}\boldsymbol{x}_i + \boldsymbol{\varepsilon}_i,$$

and we now make the modelling assumption $\boldsymbol{\varepsilon}_i \sim N(\mathbf{0}, \Sigma)$, and we assume for now that Σ is known. We have already found the least squares estimate for B, but given the additional distribution assumption we are now making, we can now construct the maximum

likelihood estimate of B. We first construct the likelihood as

$$\begin{aligned} L(\mathsf{B}) &= \prod_{i=1}^{n} (2\pi)^{-q/2} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} \boldsymbol{\varepsilon}_i^{\mathsf{T}} \Sigma^{-1} \boldsymbol{\varepsilon}_i \right\} \\ &= (2\pi)^{-nq/2} |\Sigma|^{-n/2} \exp\left\{ -\frac{1}{2} \sum_{i=1}^{n} \boldsymbol{\varepsilon}_i^{\mathsf{T}} \Sigma^{-1} \boldsymbol{\varepsilon}_i \right\} \\ &= (2\pi)^{-nq/2} |\Sigma|^{-n/2} \exp\left\{ -\frac{1}{2} \operatorname{Tr}\left(\mathsf{E} \Sigma^{-1} \mathsf{E}^{\mathsf{T}} \right) \right\} \\ &= (2\pi)^{-nq/2} |\Sigma|^{-n/2} \exp\left\{ -\frac{1}{2} \operatorname{Tr}\left(\Sigma^{-1} \mathsf{E}^{\mathsf{T}} \mathsf{E} \right) \right\}, \end{aligned}$$

where $\mathsf{E} = \mathsf{Y} - \mathsf{X}\mathsf{B}$. Consequently the log-likelihood is given by

$$l(\mathsf{B}) = -\frac{nq}{2} \log 2\pi - \frac{n}{2} \log |\Sigma| - \frac{1}{2} \operatorname{Tr}\left(\Sigma^{-1} \mathsf{E}^{\mathsf{T}} \mathsf{E} \right).$$

Since the only dependence on B in this log-likelihood is via E, it is clear that maximising this function wrt B is equivalent to minimising

$$\operatorname{Tr}\left(\Sigma^{-1} \mathsf{E}^{\mathsf{T}} \mathsf{E} \right).$$

But we have already seen that minimising this function wrt B leads to the normal equations, and the usual least squares solution

$$\hat{\mathsf{B}} = (\mathsf{X}^{\mathsf{T}} \mathsf{X})^{-1} \mathsf{X}^{\mathsf{T}} \mathsf{Y}.$$

So again, the natural estimator that we have already derived without making any distributional assumptions may be re-interpreted as a maximum likelihood estimator in cases where an assumption of multivariate normality is felt to be appropriate. Note further that since the solution we obtain is independent of Σ, it also represents the maximum likelihood estimate of B in the case of unknown Σ.

See 3.2 (p.44) and 3.7 (p.84) of **[ESL]** for further discussion of the topics discussed in this Chapter.

Chapter 4

Cluster analysis and unsupervised learning

4.1 Introduction

4.1.1 Motivation

When we did PCA for the microarray data at the end of Chapter 2, we saw in Figure 2.4 that some of the samples appeared to "cluster" together into groups. This is something that we can quite naturally do "by eye" in one or two dimensions, by examining the relative "closeness" of groups of points, but it is not so obvious how to formalise the process, or understand how it generalises to arbitrary data sets in p-dimensional space. Cluster analysis is a formalisation of this process which can be applied to any set of observations where some measure of "closeness" of observations can be defined. If one associates a label with each group or cluster, then a clustering algorithm assigns a group label to each observation in a data set. In the context of data mining and machine learning, cluster analysis is considered to be an example of unsupervised learning. This is because it is a mechanism for allocating observations to groups that does not require any training set, which provides examples of labelled observations which can be used to learn about the relationship between observations and labels. There are different approaches to cluster analysis. Some approaches assign observations to a pre-specified number of groups, and others hierarchically combine observations into small clusters and then successively into bigger clusters until the number of clusters is reduced to the desired number. Of course, *a priori* is is not always clear how many clusters will be appropriate, but methods exist to address this issue, too.

4.1.2 Dissimilarity and distance

The aim of cluster analysis is to find natural groupings of the data into subsets that are "close together" in some appropriate sense. For an $n \times p$ data matrix, with rows representing p-dimensional observations, Euclidean distance is the natural "closeness" metric, but we often want to apply clustering algorithms to data where there is no natural metric, so we simply require that for every pair of observations $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ a corresponding distance, d_{ij}, is defined. These distances are typically considered as an $n \times n$ distance matrix, D, and the distances are expected to have the following properties:

1. $d_{ij} \geq 0, \ \forall i, j$;

2. $d_{ii} = 0, \ \forall i$;

3. $d_{ij} = 0 \Rightarrow \boldsymbol{x}_i = \boldsymbol{x}_j$.

In addition to these fundamental properties, it is expected that the dissimilarities measure "closeness", so that vectors that are more "similar" have dissimilarities which are smaller than than those for vectors which are very "different". In addition, most clustering algorithms require that the dissimilarities are symmetric, so that $d_{ij} = d_{ji}$. A distance matrix which is not symmetric can be symmetrised by instead using $\mathsf{D}' = (\mathsf{D} + \mathsf{D}^\mathsf{T})/2$, which clearly is. Note however, that we typically do not require strong properties of norms or metrics, such as the triangle inequality.

For observations $\boldsymbol{x}_i \in \mathbb{R}^p$, we typically use a dissimilarity measure derived from a genuine norm. The most commonly used choices are Euclidean distance,

$$d_{ij} = d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\mathsf{T}(\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

or squared Euclidean distance

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\mathsf{T}(\boldsymbol{x}_i - \boldsymbol{x}_j).$$

Occasionally the l^1 norm, often known as Manhattan distance or the "taxi cab" norm, $d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbf{1}_p{}^\mathsf{T}|\boldsymbol{x}_i - \boldsymbol{x}_j|$, is used. In some applications it also makes sense to use the Mahalanobis distance,

$$d_{ij} = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\mathsf{T}\mathsf{S}^{-1}(\boldsymbol{x}_i - \boldsymbol{x}_j)},$$

(or the square of this), where S is the sample variance matrix of X.

If not all variables are real-valued, then usually an overall distance function $d(\cdot, \cdot)$ is constructed as a weighted sum of distances defined appropriately for each variable, as

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{k=1}^{p} w_k d_k(x_{ik}, x_{jk}).$$

For any real valued variables, the obvious choice is

$$d_k(x_{ik}, x_{jk}) = (x_{ik} - x_{jk})^2,$$

and for categorical variables, or factors, the simplest choice is given by

$$d_k(x_{ik}, x_{jk}) = \begin{cases} 0 & x_{ik} = x_{jk} \\ 1 & x_{ik} \neq x_{jk}, \end{cases}$$

though there are obviously other possibilities. The weights, w_k, could be chosen to be equal, but this probably only makes sense if the distances for the different variables are comparable, and all variables are felt to be equally important. Otherwise the weights should be chosen to ensure that each variable is weighted appropriately in the overall distance function. See section 14.3 (p.501) of **[ESL]** for further details.

Example: toy problem

Suppose that we have the data matrix

$$\mathsf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 3 \\ 4 & 2 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}.$$

We can create this in **R** using

```
X=matrix(c(1,2,3,3,4,2,2,1,3,1),ncol=2,byrow=TRUE)
```

We can compute a distance matrix for X using the **R** function `dist()`. For example

```
> dist(X)
         1        2        3        4
2 2.236068
3 3.000000 1.414214
4 1.414214 2.236068 2.236068
5 2.236068 2.000000 1.414214 1.000000
```

shows that `dist()` returns the lower triangle of the distance matrix (not including the diagonal, which is zero). It also defaults to Euclidean distance, but can use other metrics, as the following session illustrates.

```
> dist(X,method="manhattan")
  1 2 3 4
2 3
3 3 2
4 2 3 3
5 3 2 2 1
```

See `?dist` for further details about the different metrics supported. Note that `dist()` returns a `dist` object, which is *not* an **R** matrix object. If a matrix is required, it can be converted to a matrix using `as.matrix()`, as follows.

```
> as.matrix(dist(X,method="manhattan"))
  1 2 3 4 5
1 0 3 3 2 3
2 3 0 2 3 2
3 3 2 0 3 2
4 2 3 3 0 1
5 3 2 2 1 0
```

Similarly, a matrix can be converted into a `dist` object using `as.dist()`. `dist` objects are used by some of the clustering functions in **R**.

4.2 Clustering methods

4.2.1 K-means clustering

We begin by examining a clustering method which makes most sense when the observations are regarded as elements of Euclidean space, $\mathbb{R}^p$ and the distance used is Euclidean

distance (though in fact, it will work for observations belonging to any inner product space with distance defined using the induced norm), with

$$d_{ij}^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathsf{T}}(\boldsymbol{x}_i - \boldsymbol{x}_j).$$

So suppose we have an $n \times p$ data matrix X, and consider a clustering algorithm which allocates each of the n observations to one of k clusters, of size $n_1, n_2, \ldots, n_k$, where $n = \sum_{i=1}^{k} n_i$. It is helpful to introduce a new indexing notation for the observations. We will now use $\boldsymbol{x}_{ij}$ to denote the jth observation allocated to the ith cluster, and we will define the cluster means in the obvious way as

$$\bar{\boldsymbol{x}}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \boldsymbol{x}_{ij}, \quad i = 1, 2, \ldots k.$$

These are the k-means referred to in the name of the algorithm. At this point it is helpful to note that we have a sum-of-squares decomposition for the distances of observations from their centroid exactly analogous to that used in one-way ANOVA.

Proposition 40 *The sum-of-squares decomposition for the clustered observations can be written in the form*

$$SS_{TOT} = SS_W + SS_B,$$

where

$$SS_{TOT} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \|\boldsymbol{x}_{ij} - \bar{\boldsymbol{x}}\|^2$$

$$SS_W = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \|\boldsymbol{x}_{ij} - \bar{\boldsymbol{x}}_i\|^2$$

$$SS_B = \sum_{i=1}^{k} n_i \|\bar{\boldsymbol{x}}_i - \bar{\boldsymbol{x}}\|^2.$$

The overall sum-of-squares distance for the data can therefore be orthogonally decomposed into a "within groups" and "between groups" sum of squares.

Proof

$$
\begin{aligned}
\sum_{i=1}^{k}\sum_{j=1}^{n_i}\|\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}\|^2 &= \sum_{i=1}^{k}\sum_{j=1}^{n_i}\|\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i+\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}}\|^2 \\
&= \sum_{i=1}^{k}\sum_{j=1}^{n_i}\Big[\|\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i\|^2+\|\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}}\|^2 \\
&\qquad\qquad +2(\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}})^{\mathsf{T}}(\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i)\Big] \\
&= \sum_{i=1}^{k}\sum_{j=1}^{n_i}\|\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i\|^2+\sum_{i=1}^{k}\sum_{j=1}^{n_i}\|\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}}\|^2 \\
&\qquad\qquad +2\sum_{i=1}^{k}\sum_{j=1}^{n_i}(\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}})^{\mathsf{T}}(\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{n_i}\|\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i\|^2+\sum_{i=1}^{k}n_i\|\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}}\|^2 \\
&\qquad\qquad +2\sum_{i=1}^{k}(\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}})^{\mathsf{T}}\sum_{j=1}^{n_i}(\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{n_i}\|\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i\|^2+\sum_{i=1}^{k}n_i\|\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}}\|^2,
\end{aligned}
$$

since $\sum_{j=1}^{n_i}(\boldsymbol{x}_{ij}-\bar{\boldsymbol{x}}_i)=0$. □

An effective clustering algorithm with make the within-group sum-of-squares SS_W as small as possible (conditional on k), and so equivalently, will make SS_B as large as possible (since SS_{TOT} is fixed). This, then, is precisely the goal of k-means clustering: to allocate observations to clusters to minimise SS_W. This initially seems like a simple task: just consider every possible partition of the n observations into k clusters and choose the allocation resulting in the smallest value of SS_W. Unfortunately this is not practical for any data set of reasonable size, as the number of possible partitions of the data grows combinatorally. For example, for $n=100$ and $k=5$ the number of possible partitions is around 10^{68}. Clearly a more practical solution is required. The k-means algorithm is one possible approach.

The k-means algorithm

1. Start by picking k "means", $\boldsymbol{m}_1,\boldsymbol{m}_2,\ldots,\boldsymbol{m}_k$, randomly (randomly choosing k of the n observations, without replacement, often works well)
2. Go through each observation, $\boldsymbol{x}$, in turn, and allocate it to cluster i if $\boldsymbol{m}_i$ is the "closest" mean, that is, $\|\boldsymbol{x}-\boldsymbol{m}_i\|<\|\boldsymbol{x}-\boldsymbol{m}_j\|,\ j\neq i$.
3. Set each $\boldsymbol{m}_i$ to be the sample mean of observations allocated to that cluster, that is, set $\boldsymbol{m}_i=\bar{\boldsymbol{x}}_i,\ i=1,2,\ldots,k$.
4. If the procedure has not converged, return to step 2.

To understand why the k-means algorithm works, first note that once we have completed step 2. the first time through the loop, we have a set of means and an allocation of each observation to a cluster, and so we can consider the quantity

$$SS = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \|\boldsymbol{x}_{ij} - \boldsymbol{m}_i\|^2,$$

and note that at the end of step 3., each time through the loop, the value of SS coincides with SS_W for the particular cluster allocation. But the k-means algorithm minimises SS with respect to both the choice of the $\boldsymbol{m}_i$ and the allocation of observations to clusters, and hence minimises SS_W.

To see this, first consider step 3. We have a value of SS from the end of step 2., but in step 3 we alter the $\boldsymbol{m}_i$ by replacing them with the cluster means $\bar{\boldsymbol{x}}_i$. Now note that the expression

$$\sum_{j=1}^{n_i} \|\boldsymbol{x}_{ij} - \boldsymbol{m}_i\|^2$$

is minimised with respect to $\boldsymbol{m}_i$ by choosing $\boldsymbol{m}_i = \bar{\boldsymbol{x}}_i$, and hence step 3 has the effect of minimising SS with respect to the $\boldsymbol{m}_i$ conditional on the allocation of observations to clusters. So step 3 cannot increase the value of SS, and any change in any $\boldsymbol{m}_i$ will correspond to a decrease in the value of SS.

Now assuming that the algorithm has not converged, the algorithm will return to step 2. Again note that there will be a value of SS from the end of step 3. (corresponding to SS_W for the cluster allocation). Step 2. will have the effect of leaving some observations assigned to the same cluster, and moving some observations from one cluster to another. Consider an observation $\boldsymbol{x}$ that is moved from cluster i to cluster j. It is moved because $\|\boldsymbol{x} - \boldsymbol{m}_j\| < \|\boldsymbol{x} - \boldsymbol{m}_i\|$, and so thinking about the effect of this move on SS, it is clear that the increase in SS associated with cluster j is less than the decrease in SS associated with cluster i, and so the overall effect of the move is to decrease SS. This is true of every move, and so the overall effect of step 2 is to decrease SS.

Since both steps 2. and 3. have the effect of decreasing SS, and SS is bounded below by zero, the algorithm must converge to a local minimum, and this corresponds to a local minimum of SS_W. However, it must be emphasised that the algorithm is not guaranteed to converge to a *global* minimum of SS_W, and there is always the possibility that the algorithm will converge to a local minimum that is very poor compared to the true global minimum. This is typically dealt with by running the whole algorithm to convergence many times with different random starting means and then choosing the run leading to the smallest value of SS_W.

Example: microarray data

We can carry out a k-means clustering of the microarray data using **R** very simply using the `kmeans()` function. In the simplest case, we can do a clustering (with $k = 4$) using the command:

```
km=kmeans(t(nci),4)
```

The results of the clustering procedure are stored in an object that we have called `km`, and we will examine this shortly. Note that additional arguments may be passed into

the `kmeans` function to influence the way the algorithm is carried out. For example, the command

```
km=kmeans(t(nci),4,iter.max=50,nstart=20)
```

will allow the algorithm to run for up to 50 iterations before terminating, and will be run 20 times with different random starting points, returning the results from the best of the 20 runs (in terms of smallest SS_W). The following **R** session shows how to access the results of the clustering procedure.

```
> km$tot.withinss
[1] 200105.4
> km$betweenss
[1] 67757.05
> km$totss
[1] 267862.4
> km$cluster
        CNS         CNS         CNS       RENAL      BREAST
          4           4           4           4           4
        CNS         CNS      BREAST      NSCLC       NSCLC
          4           4           4           4           4
      RENAL       RENAL       RENAL       RENAL       RENAL
          4           4           4           4           4
      RENAL       RENAL      BREAST      NSCLC       RENAL
          4           4           4           4           4
    UNKNOWN     OVARIAN    MELANOMA    PROSTATE     OVARIAN
          4           4           4           1           1
    OVARIAN     OVARIAN     OVARIAN     OVARIAN    PROSTATE
          4           1           4           4           4
      NSCLC       NSCLC       NSCLC    LEUKEMIA K562B-repro
          4           4           4           2           2
K562A-repro    LEUKEMIA    LEUKEMIA    LEUKEMIA    LEUKEMIA
          2           2           2           2           2
   LEUKEMIA       COLON       COLON       COLON       COLON
          2           1           1           1           1
      COLON       COLON       COLON MCF7A-repro      BREAST
          1           1           1           1           1
MCF7D-repro      BREAST       NSCLC       NSCLC       NSCLC
          1           1           1           1           1
   MELANOMA      BREAST      BREAST    MELANOMA    MELANOMA
          3           3           3           3           3
   MELANOMA    MELANOMA    MELANOMA    MELANOMA
          3           3           3           3
```

Really it would be nice to be able to visualise the results of the clustering in some way. Obviously this is not completely trivial, due to the very high dimension of the observations. However, we saw in Chapter 2, Figure 2.4, that projecting down to the first two principal components provides for a useful visualisation of the relationship between the samples, so we can examine the results of our clustering in this context using the following commands,

```
pca=prcomp(t(nci))
plot(pca$x[,1],pca$x[,2],pch=km$cluster,col=km$cluster)
text(pca$x[,1],pca$x[,2],colnames(nci),cex=0.3,pos=3)
```

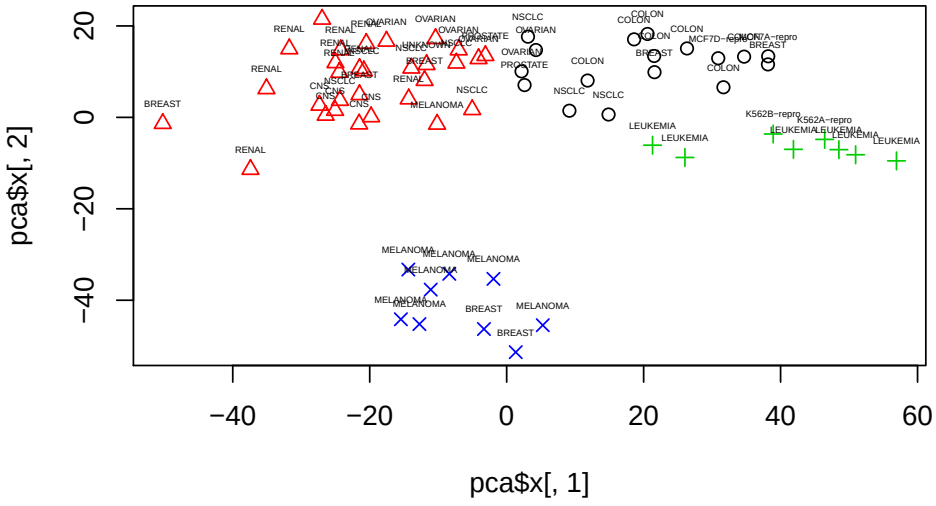


Figure 4.1: k-means clustering (with $k = 4$) overlaid on a scatterplot of the first two principal components of the `nci` microarray data

leading to the plot shown in Figure 4.1.

Choice of k

So far we haven't discussed the choice of k. In some applications it will be clear from the context exactly what k is most appropriate, and in many others, it will be clear "roughly" what values of k should be considered. But in many cases a particular choice of k will not be given *a priori*, and some indication from the data of what value might be most appropriate can be very useful.

In practice, the choice of k is handled by running the k-means algorithm for several values of k in the range of interest, and examining the way in which SS_W decreases as k increases (it should be clear that the minimum SS_W should decrease monotonically with increasing k). Ideally we look for some kind of "kink", or at least, levelling off of the sum of squares as k increases, indicating diminishing returns from increasing k.

Example: choosing k for the microarray data

We can examine the behaviour of the k-means algorithm for varying values of k using the following fragment of **R** code,

```
kmax=8
wss=numeric(kmax)
for (i in 1:kmax) {
  km=kmeans(t(nci),i,iter.max=50,nstart=20)
  wss[i]=km$tot.withinss
}
plot(1:kmax,wss,type="l",xlab="k",ylab="SS_W",lwd=2,col=2)
```

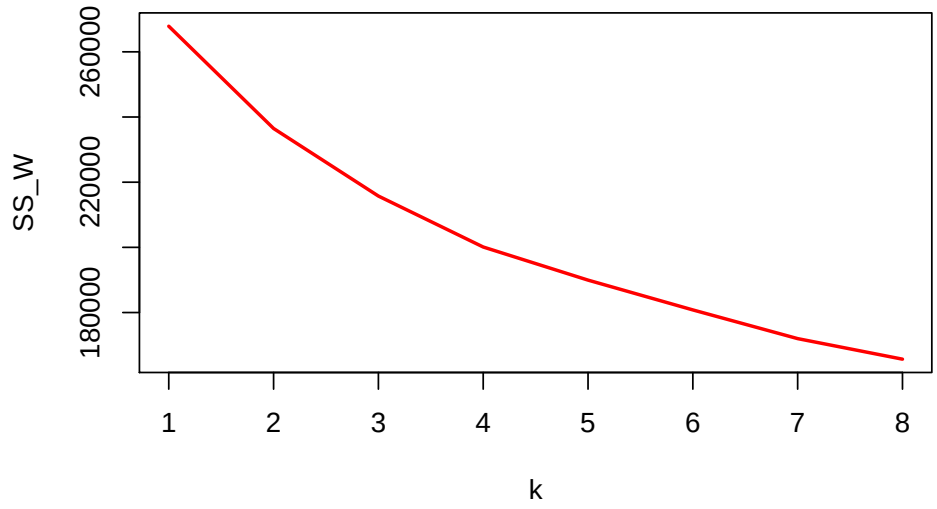


Figure 4.2: SS_W against k for k-means clustering of the `nci` microarray data

leading to the plot shown in Figure 4.2. Unfortunately, but quite typically, there does not appear to be a very distinctive kink or flattening off of the curve at any point, but our original choice of $k = 4$ seems reasonable, as there are diminishing returns for further increasing k by this point.

Example: handwritten digit images

We can also cluster the `zip.train` data. Obviously we have a label which classifies the image, but we can strip this off and run a k-means with 10 clusters to see what happens. Ideally, in doing this, we would like to see each cluster representing a different digit, as this would correspond to the different digits occupying distinct regions of $\mathbb{R}^{256}$. The command

```
km=kmeans(zip.train[,-1],10,iter.max=30,nstart=10)
```

is quite slow to run due to the size of the data set. When it is finished we can examine the content of the clusters as illustrated in the following **R** session.

```
> table(zip.train[,1][km$cluster==1])

  0   1   2   3   4   5   6   7   8   9
  7   3  46  73   5  71   2   4 433   7
> table(zip.train[,1][km$cluster==2])

   1    2    3    4    6    7    8    9
1001    3    1   42    9    3   11    5
> table(Cluster=km$cluster,Digit=zip.train[,1])
       Digit
Cluster    0    1    2    3    4    5    6    7    8    9
     1     7    3   46   73    5   71    2    4  433    7
```

```
2      0 1001    3    1   42    0    9    3   11    5
3    524    0    3    2    0   15   42    0    2    0
4      0    0   13    2    5    3    0  445    1   61
5    512    0   12    1    1    7   29    0    4    0
6     15    0   34  546    0  296    1    0   30    2
7      7    0  565   12   14    9   36    2   10    0
8     10    0   40    8  440   40   10   23   10  125
9    119    0    6    2    6  110  533    0    3    0
10     0    1    9   11  139    5    2  168   38  444
```

From the results of this analysis, we see that the cluster labelled 1 represents mainly the digit 8, together will a few other digits, especially 3 and 5, which are fairly similar to a 8. The cluster labelled 2 mainly represents the digit 1. Cluster 3 consists mainly of the digit 0, but also contains a number of 6s. *Note that the cluster labels are completely arbitrary*, and re-running the analysis will (hopefully) lead to similar clusters, but typically with a different permutation of the labels. On the basis of this very superficial analysis, it does appear that the digits do largely occupy different regions of $\mathbb{R}^{256}$, and so this gives some encouragement that one might be able to automatically classify digits given just the images (we will look at classification in more detail in the next chapter).

The following code fragment shows how to look at SS_W as a function of k.

```
kmax=15
wss=numeric(kmax)
for (i in 1:kmax) {
  print(i)
  km=kmeans(zip.train[,-1],i,iter.max=30,nstart=10)
  wss[i]=km$tot.withinss
}
plot(1:kmax,wss,type="l",xlab="k",ylab="SS_W",lwd=2,col=2)
```

The resulting plot (not shown), perhaps surprisingly, fails to show any especially noticeable kink or flattening off at $k = 10$. In fact, this is just symptomatic of the general difficulty of fixing k purely on the basis of the data. More sophisticated model-based clustering procedures provide principled methods for choosing k, but in most real data scenarios the evidence for one value of k over another is often very weak. See section 14.3.6 (p.509) of **[ESL]** for further details.

Pre-transformation, Mahalanobis distance and the QR factorisation

Note that k-means is strongly dependent on the assumption that Euclidean distance is an appropriate metric for measuring dissimilarity of observations. If that is not the case, some kind of transformation of the data prior to clustering can be used in order to try and ensure that Euclidean distance for the transformed data matrix *is* appropriate. For example, if Mahalanobis distance is felt to be a more appropriate metric (as it properly accounts for the correlation structure in the data), then the data can be standardised using a Mahalanobis transformation prior to clustering. We have seen previously that both the symmetric square root and Cholesky factor of the sample variance matrix can be used to compute Mahalanobis distance, and so it will not matter if we standardise using the symmetric square root or Cholesky factor. Further, we have shown that standardisation using the Cholesky factor is essentially equivalent to forming the QR factorisation of the

centered data matrix. Consequently, we can cluster based on Mahalanobis distance by running k-means on the Q matrix of a QR factorisation. For example, we can do this for the `zip.train` data as follows.

```
xbar=colMeans(zip.train[,-1])
W=sweep(zip.train[,-1],2,xbar)
QR=qr(W)
km=kmeans(qr.Q(QR),10,iter.max=30,nstart=10)
```

Whether or not the resulting clustering will be more appropriate will be very problem dependent. We will come back to this issue later when we look at discrimination and classification.

4.2.2 Hierarchical clustering

k-means is fast and efficient, and scales well to large data sets. However, it only makes sense for observations in Euclidean space, and requires specification of the number of clusters, k. We will now examine an approach to clustering which does not require prespecification of the desired number of clusters, and does not require a metric distance matrix. The idea behind hierarchical clustering, often also known as *agglomerative clustering*, is to start with each observation in a separate cluster, and then join the two nearest observations into a cluster containing just two observations. Then the algorithm proceeds at each stage by combining the clusters that are "closest" together into a new cluster. Clearly for this to make sense, we need a way of defining the distance between two clusters, given that we know the distances between all pairs of observations. There are several different ways that this can be done, and each method leads to a clustering algorithm with different properties. However, provided that we have some way to do this, we can describe a generic hierarchical clustering algorithm as follows.

1. Start with n clusters, $C_1, C_2, \ldots, C_n$, with C_i containing just the single observation $\boldsymbol{x}_i$. The distance matrix for the n clusters is just taken to be the distance matrix for the n observations.
2. Find the minimum distance, d_{ij} (if there is more than one minimum, pick one at random).
3. Combine clusters C_i and C_j into a single cluster, remove the ith and jth rows and columns from the distance matrix, and add a new row and column corresponding to the new cluster by calculating the distances between the new cluster and the remaining clusters.
4. Note that the number of clusters has decreased by one.
5. If more than one cluster remains, return to step 2.

To put this algorithm into practice, we need ways of computing the distance between clusters. One of the most commonly used methods, and arguably the most intuitive, is to define the distance between two clusters to be the minimum distance between observations in the clusters, that is, for clusters A and B we define

$$d_{AB} = \min_{i \in A, j \in B} d_{ij}.$$

This definition leads to the so-called single-linkage clustering method, often known as *nearest-neighbour clustering*. Alternatively, we can define the distance between clusters to be the maximum distance between observations in the clusters, that is

$$d_{AB} = \max_{i \in A, j \in B} d_{ij}.$$

This definition leads to the complete-linkage clustering method, often known as *furthest-neighbour clustering*. Both of these methods have the property that they are invariant to any monotone transformation of the distance matrix. A further possibility (which does not have this monotonicity property), is to use group average clustering, where

$$d_{AB} = \frac{1}{n_A n_B} \sum_{i \in A} \sum_{j \in B} d_{ij}.$$

Example: toy problem

Let us begin by working through single-linkage and complete-linkage clustering by hand for 5 observations, A, B, C, D, E having distance matrix

$$\mathsf{D} = \begin{pmatrix} 0 & & & & \\ 7 & 0 & & & \\ 4 & 1 & 0 & & \\ 6 & 4 & 6 & 0 & \\ 8 & 9 & 3 & 2 & 0 \end{pmatrix}.$$

Let us begin by working through the single-linkage algorithm. It is helpful to write out the distance matrix in the form of a table, as

A	0				
B	7	0			
C	4	1	0		
D	6	4	6	0	
E	8	9	3	2	0
	A	B	C	D	E

The minimum (off-diagonal) distance is clearly $d(B, C) = 1$, and so we eliminate the rows and columns for B and C, then add a new final row for the new cluster $\{B, C\}$ as follows:

A	0			
D	6	0		
E	8	2	0	
$\{B, C\}$	**4**	**4**	**3**	0
	A	D	E	$\{B, C\}$

Here the new distances have been calculated using the single-linkage minimum distance rule. Looking at the new table, we see that the smallest off-diagonal distance is $d(D, E) = 2$, and so we remove D and E and add the new row $\{D, E\}$ as follows:

A	0		
$\{B, C\}$	4	0	
$\{D, E\}$	**6**	**3**	0
	A	$\{B, C\}$	$\{D, E\}$

Inspecting this new table, we see that the minimum distance is given by $d(\{B,C\},\{D,E\}) = 3$, and this leads to the final table

A	0	
$\{B,C,D,E\}$	**4**	0
	A	$\{B,C,D,E\}$

The final step combines the remaining two clusters into a single cluster consisting of all observations. We can display the results of a clustering process in a tree diagram known as a dendrogram by successively joining each pair of clusters in the order they were merged in the clustering algorithm, and using the vertical axis to mark the distance between the clusters at the stage they were merged. The result of this for the single-linkage clustering is shown in Figure 4.3.

We now re-do the clustering procedure using the complete-linkage method, based on maximum distance. We start with the distance table as previously.

A	0				
B	7	0			
C	4	1	0		
D	6	4	6	0	
E	8	9	3	2	0
	A	B	C	D	E

The minimum (off-diagonal) distance is clearly $d(B,C) = 1$, just as before, and so we eliminate the rows and columns for B and C, then add a new final row for the new cluster $\{B,C\}$ as follows:

A	0			
D	6	0		
E	8	2	0	
$\{B,C\}$	**7**	**6**	**9**	0
	A	D	E	$\{B,C\}$

Here the new distances have been calculated using the complete-linkage maximum distance rule, and hence differ from those obtained previously. The minimum distance in this new table is $d(D,E) = 2$, and so we merge D and E to get the new table

A	0		
$\{B,C\}$	7	0	
$\{D,E\}$	**8**	**9**	0
	A	$\{B,C\}$	$\{D,E\}$

In this new table, the minimum distance is $d(A,\{B,C\}) = 7$, and so we merge those to get

$\{D,E\}$	0	
$\{A,B,C\}$	**9**	0
	$\{D,E\}$	$\{A,B,C\}$

Again, the final step is to merge the final two clusters. We can plot the results of this clustering in the dendrogram shown in Figure 4.4. Note that this dendrogram is different to that for single-linkage clustering. We can now repeat this analysis using **R**. We begin by constructing a distance matrix, as follows.

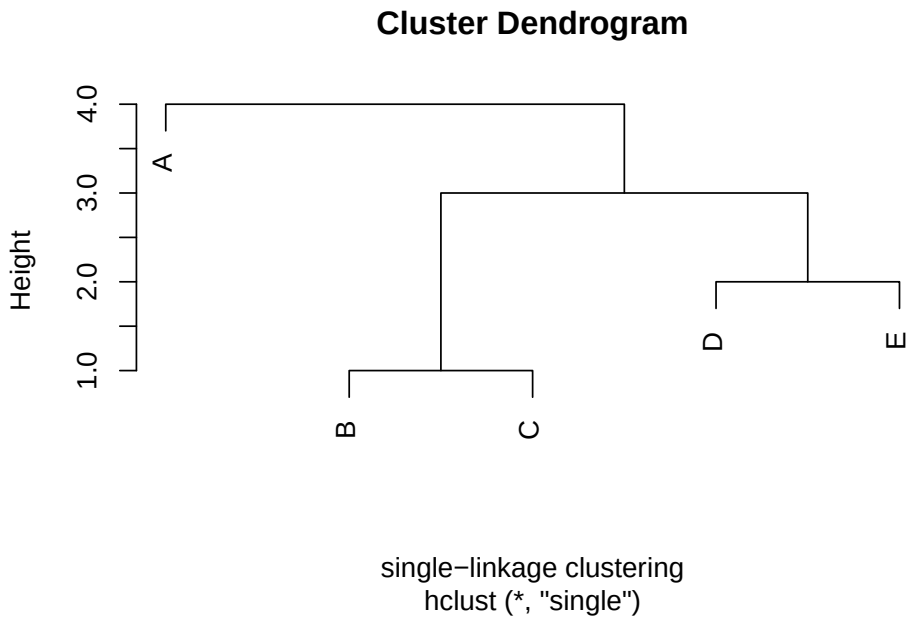


Figure 4.3: Dendrogram for a single-linkage hierarchical clustering algorithm

```
> dm=matrix(c(0,7,4,6,8,7,0,1,4,9,4,1,0,6,3,6,4,6,0,2,8,9,3,2,0),ncol
   =5)
> colnames(dm)=c("A","B","C","D","E")
> d=as.dist(dm)
> d
  A B C D
B 7
C 4 1
D 6 4 6
E 8 9 3 2
```

We can then use the `hclust()` function to carry out hierarchical clustering and plot the result as a dendrogram using

```
hc=hclust(d,method="single")
plot(hc,xlab="single-linkage clustering")
```

leading to the plot shown in Figure 4.3. Note the use of `method="single"` to force the use of single-linkage clustering. See `?hclust` for other algorithms which can be used. We can re-do the analysis using complete-linkage clustering with

```
hc=hclust(d)
plot(hc,xlab="complete-linkage clustering")
```

giving the plot shown in Figure 4.4, since the method used by `hclust()` defaults to **`complete`**.

To obtain an actual clustering, we need to decide at some point to stop merging clusters. In principle we could do this by terminating the clustering algorithm at some point. In practice however, the algorithm is usually run to completion, and the number of clusters is decided on post-hoc, usually after inspecting the dendrogram. Choosing a number of

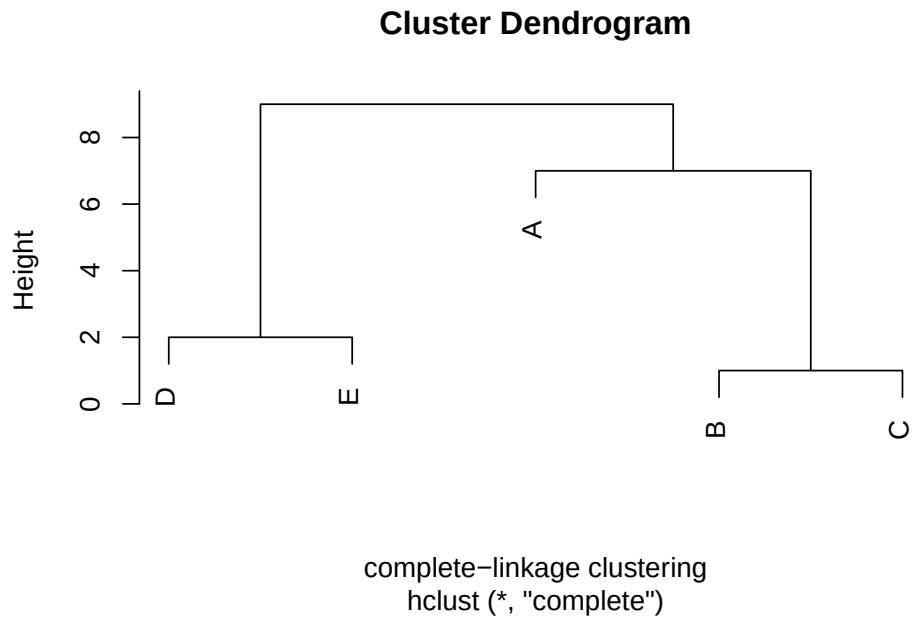


Figure 4.4: Dendrogram for a complete-linkage hierarchical clustering algorithm

clusters corresponds to "cutting" the dendrogram "tree" at some height, and the **R** command which does this is `cutree()`. The tree can be cut by specifying a desired number of clusters, as in

```
> ct=cutree(hc,3)
> ct
A B C D E
1 2 2 3 3
```

to obtain 3 clusters. Alternatively, the clustering can be obtained by specifying the height at which the tree is to be cut, for example

```
ct=cutree(hc,h=2.5)
```

Just as for k-means, *ad hoc* methods are used to justify the number of clusters. Often people try to look for a "break" in the tree, where there is a big jump in distance before the next merging, but very often no such obvious gap is present in practice.

Example: microarray data

We can easily apply our clustering methods to the `nci` microarray data. We begin with single-linkage

```
hc=hclust(dist(t(nci)),method="single")
plot(hc,cex=0.4)
```

leading to the plot shown in Figure 4.5. Alternatively, we can use complete-linkage clustering with

```
hc=hclust(dist(t(nci)))
plot(hc,cex=0.4)
```

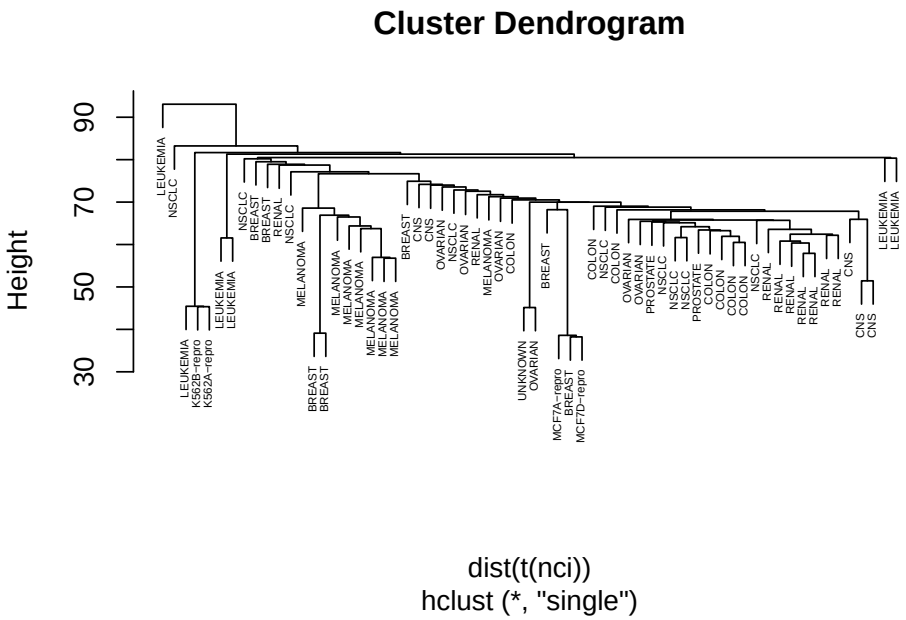


Figure 4.5: Dendrogram for a single-linkage hierarchical clustering algorithm applied to the `nci` microarray data

to get the plot shown in Figure 4.6, or average-linkage clustering with

```
hc=hclust(dist(t(nci)),method="average")
plot(hc,cex=0.4)
```

to get the plot shown in Figure 4.7.

Comparing the figures, we see that single-linkage methods are prone to chaining together nearby observations one-by-one (the algorithm is essentially the algorithm for finding a minimal spanning tree), and can lead to long, thin clusters. The complete-linkage method tends to give fairly spherical clusters, and is generally to be preferred, which is why it is the default method in **R**. Other methods tend to try and interpolate between these two extremes, including the average-linkage method.

To obtain an actual cluster allocation, we can cut the tree at 4 clusters and overlay the results on the first two principal components, just as we did for the k-means procedure.

```
hc=hclust(dist(t(nci)))
ct=cutree(hc,4)
pca=prcomp(t(nci))
plot(pca$x[,1],pca$x[,2],pch=ct,col=ct)
text(pca$x[,1],pca$x[,2],colnames(nci),cex=0.3,pos=3)
```

This gives rise to the plot shown in Figure 4.8. The clustering perhaps looks unexpected, but is difficult to interpret, since the clustering was carried out on (a distance matrix derived from) the full observation vectors, but we are looking at the clusters on a simple 2d projection.

Hierarchical clustering can be used to order a set of observations in such a way that nearby observations are typically more closely related than observations that are far away. This is exploited by the `heatmap()` function that we examined briefly in Chapter 1. Consider now the following command

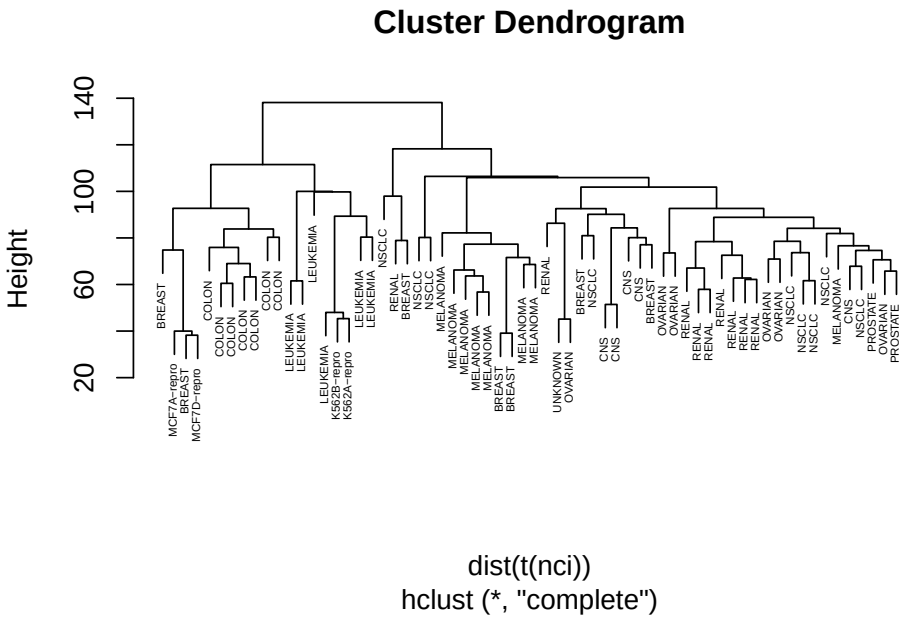


Figure 4.6: Dendrogram for a complete-linkage hierarchical clustering algorithm applied to the `nci` microarray data

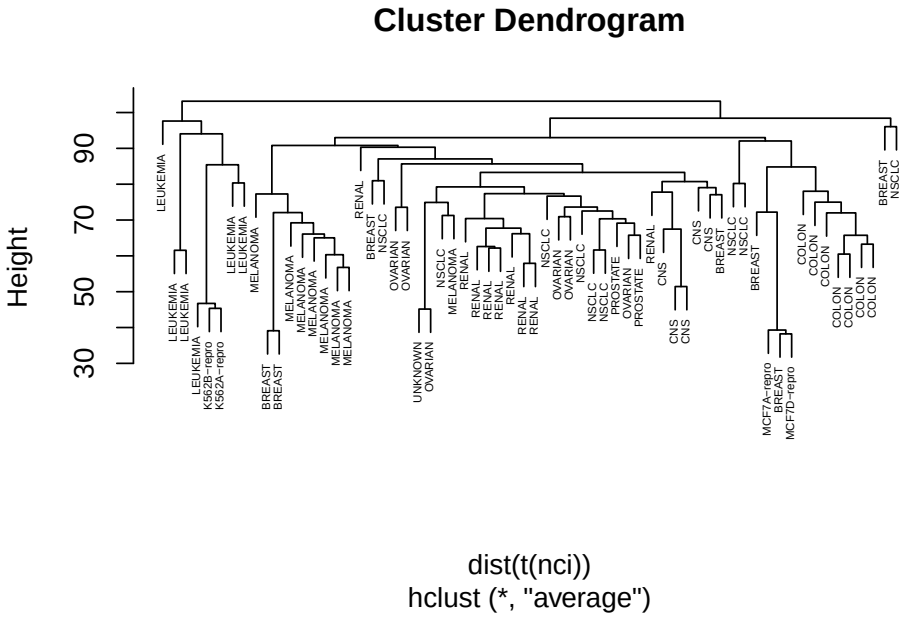


Figure 4.7: Dendrogram for a average-linkage hierarchical clustering algorithm applied to the `nci` microarray data

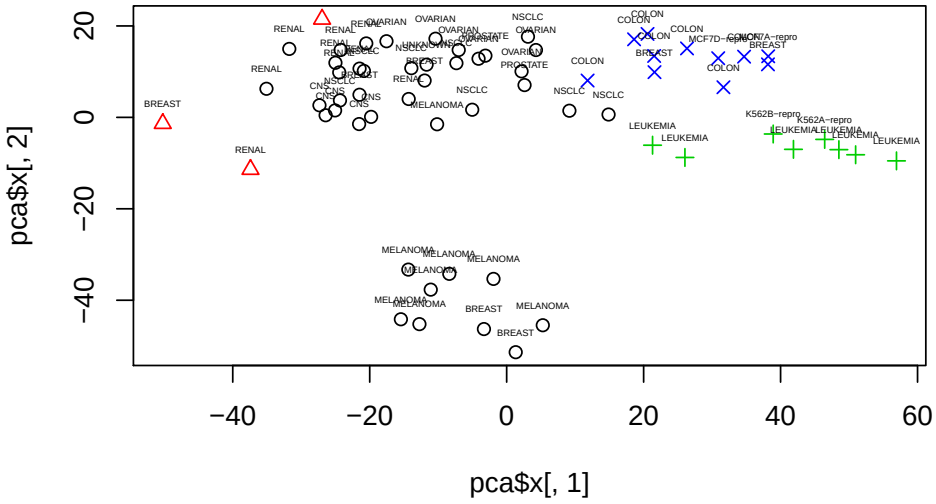


Figure 4.8: Four clusters obtained from a complete-linkage hierarchical clustering algorithm overlaid on the first two principal components of the `nci` microarray data

```
heatmap(nci,Rowv=NA,labRow=NA,col=grey((15:0)/15),cexCol=0.3)
```

which produces the plot shown in Figure 4.9. The use of the option `Rowv=NA` suppresses the clustering of the rows (which is very time consuming and not particularly useful), but since we have not explicitly suppressed clustering of the columns, this default action is executed, and ensures that the columns are ordered meaningfully. Inspection of the classification labels at the bottom of the columns suggests that the clustering agrees well with the manually obtained diagnoses.

Example: handwritten digits

In principle, we can use hierarchical clustering to cluster the handwritten digits into 10 clusters using the following **R** commands.

```
hc=hclust(dist(zip.train[,-1]))
ct=cutree(hc,10)
```

However, hierarchical clustering does not scale well to large n scenarios such as this, and the above commands will take a very long time to complete on all but the most powerful computers. This should be contrasted with the microarray data (large p, but small n), which clustered fine. Nevertheless, we can run the above commands and eventually we will get a clustering, which we can examine similarly to previously as illustrated below.

```
> table(Cluster=ct,Digit=zip.train[,1])
       Digit
Cluster    0    1    2    3    4    5    6    7    8    9
      1   21    2   33   78  104  387  451  108   23  193
      2   56 1001  110   11  176   24   84    6  166   23
      3    2    0   25   28  286   71    0  506  264  411
```

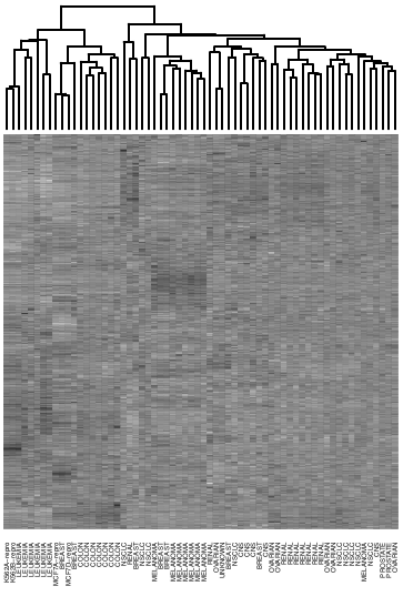


Figure 4.9: A heatmap of the `nci` microarray data with columns ordered according to a hierarchical clustering

```
4   256    0   93  303    0   24    8    0   19    0
5     3    2  285  229    7   13    0    1   52    0
6   361    0    6    1    4   10  116    0    5    0
7     1    0  175    0   29    0    0    1    3    1
8    80    0    0    4    0   25    4    0   10    0
9   413    0    1    0    0    2    0    0    0    0
10    1    0    3    4   46    0    1   23    0   16
```

Here cluster 1 consists mainly of the digits 5 and 6, cluster 2 represents the digit 1, cluster 3 is a mixture of digits 4, 7, 8 and 9, and cluster 4 is a mix of 0 and 3. Again, there is not a strong separation of digits in the clusters.

See section 14.3.12 (p.520) of **[ESL]** for further details of hierarchical clustering methods.

4.2.3 Model-based clustering

The clustering methods we have discussed so far (k-means and hierarchical clustering) are both just heuristic data analysis techniques. However useful they are for data exploration, it is difficult to use them to make strong statements about clustering in the population from which they were sampled without making some modelling assumptions. The standard modelling assumption is that each cluster has a multivariate normal distribution, but that the different clusters have different MVN distributions. It is then possible to use maximum likelihood or Bayesian methods in order to fit these models to the available data. Such techniques are beyond the scope of this course, but are becoming increasingly widely used in modern data analysis. Such methods can be found in the CRAN package `mclust`, which can be installed using `install.packages("mclust")` if it is not

already installed. The package can be loaded with `require(mclust)` and help can be obtained with `?Mclust`. Note that the methods in this package typically do not scale well to very large data sets.

In summary, we have looked in some detail at k-means clustering and at hierarchical clustering. The advantage of k-means clustering is that it exploits the geometry of Euclidean space (when this makes sense), and that it scales well to very large data sets (large n). The advantages of hierarchical clustering include that it does not require a "real" metric distance matrix, and that it provides very rich information about the clustering structure in the form of a dendrogram. We have not looked in detail at model-based clustering methods, but in general these work very well (when the modelling assumptions are reasonable), and allow much stronger inferences to be made about the population from which the data is sampled, but typically do not scale well to large data scenarios (large n or large p).

Chapter 5

Discrimination and classification

5.1 Introduction

In the previous chapter we considered the rather ambitious unsupervised learning task of automatically grouping observations together into clusters of "similar" observations. We now consider the related problem of classifying multvariate observations into one of k groups assuming that we have some *a priori* idea of what kinds of groups we expect to see (or at least, some examples of observations that have already been classified into pre-specified groups). We want to use our current understanding of the group structure in order to automatically classify new observations which have not yet been assigned to a group.

This is the statistical classification problem, and relies on partitioning the sample space so as to be able to *discriminate* between the different groups. Data that has already been classified into groups is often known as a training set, and methods which make use of such data are often known as supervised learning algorithms.

5.2 Heuristic classifiers

5.2.1 Closest group mean classifier

Let us start by assuming that our p-dimensional observations belong to one of k groups, and that there is a mean vector associated with each group, so the group means are $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$. Now suppose that we are given an observation $\boldsymbol{x} \in \mathbb{R}^p$. Which group should we allocate the observation to? One obvious method would be to allocate the observation to the group with mean that is closest to $\boldsymbol{x}$. That is, we could allocate $\boldsymbol{x}$ to group i if

$$\|\boldsymbol{x} - \boldsymbol{\mu}_i\| < \|\boldsymbol{x} - \boldsymbol{\mu}_j\|, \quad \forall j \neq i.$$

This is a very simple example of a classification rule. Note that it has the effect of partitioning $\mathbb{R}^p$ into k regions, $R_1, R_2, \ldots, R_k$, where $R_i \subseteq \mathbb{R}^p$, $i = 1, 2, \ldots, k$, $R_i \cap R_j = \emptyset$, $\forall i \neq j$ and $\bigcup_{i=1}^{k} R_i = \mathbb{R}^p$ in such a way that $\boldsymbol{x}$ is assigned to group i if $\boldsymbol{x} \in R_i$. Different classification rules lead to different partitions, and clearly some methods of choosing the partition will be more effective than others for ensuring that most observations will be assigned to the correct group. In this case, the partition is known as the Voronoi tessellation of $\mathbb{R}^p$ gen-

erated by the "seeds" $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$. The boundaries between the classes are piecewise linear, and to see why, we will begin by considering the case $k = 2$.

Binary classification

The case $k = 2$ is of particular interest, as it represents a very commonly encountered example in practice, and is known as binary classification. In this case one is typically interested in deciding whether a particular binary variable of interest is "true" or "false". Examples include deciding whether or not a patient has a particular disease, or deciding if a manufactured item should be flagged as potentially faulty.

In this binary case we will have 2 group means, which we can label $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$. Under our simple classification rule we allocate $\boldsymbol{x}$ to group 1 provided that $\|\boldsymbol{x} - \boldsymbol{\mu_1}\| < \|\boldsymbol{x} - \boldsymbol{\mu_2}\|$, otherwise we allocate to group 2. But then

$$\begin{aligned}
&\|\boldsymbol{x} - \boldsymbol{\mu_1}\| < \|\boldsymbol{x} - \boldsymbol{\mu_2}\| \\
\Leftrightarrow\ &\|\boldsymbol{x} - \boldsymbol{\mu_1}\|^2 < \|\boldsymbol{x} - \boldsymbol{\mu_2}\|^2 \\
\Leftrightarrow\ &(\boldsymbol{x} - \boldsymbol{\mu_1})^\mathsf{T}(\boldsymbol{x} - \boldsymbol{\mu_1}) < (\boldsymbol{x} - \boldsymbol{\mu_2})^\mathsf{T}(\boldsymbol{x} - \boldsymbol{\mu_2}) \\
\Leftrightarrow\ &\boldsymbol{x}^\mathsf{T}\boldsymbol{x} - 2\boldsymbol{\mu}_1{}^\mathsf{T}\boldsymbol{x} + \boldsymbol{\mu}_1{}^\mathsf{T}\boldsymbol{\mu}_1 < \boldsymbol{x}^\mathsf{T}\boldsymbol{x} - 2\boldsymbol{\mu}_2{}^\mathsf{T}\boldsymbol{x} + \boldsymbol{\mu}_2{}^\mathsf{T}\boldsymbol{\mu}_2 \\
\Leftrightarrow\ &2(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\mathsf{T}\boldsymbol{x} < \boldsymbol{\mu}_2{}^\mathsf{T}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1{}^\mathsf{T}\boldsymbol{\mu}_1 \\
\Leftrightarrow\ &(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_2)^\mathsf{T}\boldsymbol{x} < \frac{1}{2}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\mathsf{T}(\boldsymbol{\mu}_2 + \boldsymbol{\mu}_1) \\
\Leftrightarrow\ &(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\mathsf{T}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)\right] < 0 \\
\Leftrightarrow\ &(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\mathsf{T}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)\right] > 0.
\end{aligned}$$

Note carefully how the quadratic term $\boldsymbol{x}^\mathsf{T}\boldsymbol{x}$ cancels out to leave a discrimination rule that is linear in $\boldsymbol{x}$. If we think about the boundary between the two classes, this is clearly given by solutions to the equation

$$(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\mathsf{T}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)\right] = 0.$$

The first thing to note is that this boundary passes through the midpoint of the group means, $\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)$, and the next thing to note is that it represents a (shifted) hyperplane* orthogonal to the vector $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$. That is, the boundary is the separating hyperplane which perpendicularly bisects $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ at their mid-point.

For $k > 2$, it is clear that $\boldsymbol{x}$ will be allocated to group i if

$$(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\mathsf{T}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)\right] > 0, \quad \forall j \neq i.$$

It is then clear that the region R_i will be an intersection of half-spaces, and hence a convex polytope.

*Because of the shift, the boundary is technically an *affine set* and not a *hyperplane*, which should pass through the origin. However, most people ignore this distinction and refer to the boundary as a hyperplane.

Example: Toy problem

To illustrate the basic idea, we will illustrate it using an example with $k = 2$ and $p = 2$. Suppose that the group means are $\boldsymbol{\mu}_1 = (-1, 2)^\mathsf{T}$ and $\boldsymbol{\mu}_2 = (2, 1)^\mathsf{T}$. What is the classification rule? We first compute the mid-point

$$\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) = \begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix},$$

and the difference between the means

$$\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 = \begin{pmatrix} -3 \\ 1 \end{pmatrix}.$$

Then we allocate to group 1 if

$$\begin{aligned} (-3, 1) \left[\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix} \right] &> 0 \\ \Rightarrow -3x_1 + x_2 + 1.5 - 1.5 &> 0 \\ \Rightarrow x_2 &> 3x_1. \end{aligned}$$

Therefore the boundary is given by the line $x_2 = 3x_1$. Observations lying above the line will be allocated to group 1, and those falling below will be allocated to group 2.

5.2.2 Linear discriminant analysis (LDA)

The closest group mean classifier is a simple and natural way to discriminate between groups. However, it ignores the covariance structure in the data, including the fact that some variables are more variable than others. The more variable (and more highly correlated) variables are likely to dominate the Euclidean distance, and hence will have a disproportionate effect on the classification rule. We could correct for this by first applying a standardisation transformation to the data and the group means, and then carry out closest group mean classification, or we can directly adapt our rule to take covariance structure into account. The simplest case of this arises when the k groups have different means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$, but common variance matrix Σ. In this case we can standardise using (say) the symmetric square root $\Sigma^{-1/2}$, and allocate $\boldsymbol{x}$ to group i if

$$\|\Sigma^{-1/2}\boldsymbol{x} - \Sigma^{-1/2}\boldsymbol{\mu}_i\| < \|\Sigma^{-1/2}\boldsymbol{x} - \Sigma^{-1/2}\boldsymbol{\mu}_j\|, \quad \forall j \neq i.$$

Note that we are implicitly relying on linearity of expectation here to appropriately transform the group means. We can write this rule differently by noting that

$$\begin{aligned} &\|\Sigma^{-1/2}\boldsymbol{x} - \Sigma^{-1/2}\boldsymbol{\mu}_i\| < \|\Sigma^{-1/2}\boldsymbol{x} - \Sigma^{-1/2}\boldsymbol{\mu}_j\|, \quad \forall j \neq i \\ \Leftrightarrow\ &\|\Sigma^{-1/2}(\boldsymbol{x} - \boldsymbol{\mu}_i)\|^2 < \|\Sigma^{-1/2}(\boldsymbol{x} - \boldsymbol{\mu}_j)\|^2, \quad \forall j \neq i \\ \Leftrightarrow\ &(\boldsymbol{x} - \boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) < (\boldsymbol{x} - \boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_j), \quad \forall j \neq i. \end{aligned}$$

That is, we will allocate to the group whose mean is closest when measured according to Mahalanobis distance. Again, this rule simplifies considerably as

$$\begin{aligned}
&(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) < (\boldsymbol{x}-\boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j) \\
\Leftrightarrow\ &\boldsymbol{x}^\mathsf{T}\Sigma^{-1}\boldsymbol{x} - 2\boldsymbol{\mu}_i{}^\mathsf{T}\Sigma^{-1}\boldsymbol{x} + \boldsymbol{\mu}_i{}^\mathsf{T}\Sigma^{-1}\boldsymbol{\mu}_i < \boldsymbol{x}^\mathsf{T}\Sigma^{-1}\boldsymbol{x} - 2\boldsymbol{\mu}_j{}^\mathsf{T}\Sigma^{-1}\boldsymbol{x} + \boldsymbol{\mu}_j{}^\mathsf{T}\Sigma^{-1}\boldsymbol{\mu}_j \\
\Leftrightarrow\ &2(\boldsymbol{\mu}_j-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}\boldsymbol{x} < (\boldsymbol{\mu}_j-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{\mu}_j+\boldsymbol{\mu}_i) \\
\Leftrightarrow\ &(\boldsymbol{\mu}_j-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_j-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{\mu}_j+\boldsymbol{\mu}_i) < 0 \\
\Leftrightarrow\ &(\boldsymbol{\mu}_j-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_i+\boldsymbol{\mu}_j)\right] < 0 \\
\Leftrightarrow\ &(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_i+\boldsymbol{\mu}_j)\right] > 0.
\end{aligned}$$

Again note how the quadratic term, here $\boldsymbol{x}^\mathsf{T}\Sigma^{-1}\boldsymbol{x}$, cancels on both sides to give a linear discrimination rule. Again region R_i is a convex polytope corresponding to an intersection of half-spaces. In the binary case, the boundary between groups 1 and 2 is given by

$$(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^\mathsf{T}\Sigma^{-1}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2)\right] = 0,$$

which again passes through the mid-point $\frac{1}{2}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2)$, but is now the hyperplane orthogonal to $\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)$. This classification rule is known as Fisher's linear discriminant, and is one of the most fundamental results in classification theory.

Example: Toy problem

Let us reconsider the toy example introduced previously, but now assume a common variance of

$$\Sigma = \begin{pmatrix} 1 & 1 \\ 1 & 4 \end{pmatrix}.$$

What is the linear discriminant in this case? The mid-point is as before, but the normal is now given by

$$\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) = \frac{1}{3}\begin{pmatrix} 4 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} -3 \\ 1 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} -13 \\ 4 \end{pmatrix}.$$

Therefore we allocate to group 1 if

$$\begin{aligned}
&\frac{1}{3}(-13,4)\left[\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix}\right] > 0 \\
&\Rightarrow -13x_1 + 4x_2 + 0.5 > 0 \\
&\Rightarrow x_2 > \frac{13}{4}x_1 - \frac{1}{8}.
\end{aligned}$$

So the boundary is now given by the line $x_2 = 13x_1/4 - 1/8$, which is very similar, but different, to the boundary we computed by ignoring the variance structure. The 2 boundaries, together with some simulated data, are shown in Figure 5.1.

Computational considerations

Computationally, it is often most efficient to Mahalanobis transform the data and then carry out nearest group mean classification. This can be accomplished using the Cholesky factorisation of Σ.

Note that in the high-dimensional case, $p >> k$, there is an additional simplification which can arise, due to the fact that the k group means must all lie in a $(k-1)$-dimensional subspace of $\mathbb{R}^p$. Since the rule is to allocate to the closest group mean, all data may be projected down to this $(k-1)$-dimensional space, and the classification may be carried out in $\mathbb{R}^{k-1}$, since any components orthogonal to this subspace contribute the same distance to each of the group means, and therefore cancel out of the classification rule. When $k << p$ this dimension reduction can result in significant computational savings.

The natural basis vectors for this $(k-1)$-dimensional space are known as *discrimination coordinates*, or *crimcoords*, but their derivation and use is beyond the scope of this course.

5.2.3 Quadratic discrimination

When the variance matrices in the groups are unequal, so that the group means are $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ and the variance matrices are $\Sigma_1, \ldots, \Sigma_k$, an obvious first attempt at adapting the previous classification method would be to compare Mahalanobis distances for each group using a distance appropriate to each group. That is, allocate observation $\boldsymbol{x}$ to group i if

$$(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} {\Sigma_i}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i) < (\boldsymbol{x} - \boldsymbol{\mu}_j)^{\mathsf{T}} {\Sigma_j}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_j), \forall j \neq i.$$

Note that now the quadratic terms will not cancel, and so we end up with quadratic boundaries. In particular, in the binary case, we will allocate $\boldsymbol{x}$ to group 1 if

$$(\boldsymbol{x} - \boldsymbol{\mu}_1)^{\mathsf{T}} {\Sigma_1}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_1) < (\boldsymbol{x} - \boldsymbol{\mu}_2)^{\mathsf{T}} {\Sigma_2}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_2),$$

and so the boundary between the two groups will correspond to a contour of a quadratic form.

Note that although the above approach to constructing a quadratic discrimination rule is attractive, it does not *penalise* groups with large variances sufficiently. We will see later how to improve on this quadratic rule by attempting to take a more principled model-based approach.

5.2.4 Discrimination functions

We have now looked at several different methods for discrimination and classification, each leading to a different rule. However, all of the rules we have considered so far can be described in within a common framework by introducing the concept of a *discriminant function*. For each class $i = 1, 2, \ldots, k$, we define a corresponding function

$$Q_i(\cdot) : \mathbb{R}^p \longrightarrow \mathbb{R},$$

known as a *discriminant function* which determines a partition of $\mathbb{R}^p$, $R_1, R_2, \ldots, R_k$, by assigning an observation $\boldsymbol{x}$ to R_i if

$$Q_i(\boldsymbol{x}) > Q_j(\boldsymbol{x}), \quad \forall j \neq i.$$

Note the use of a $>$ inequality rather than a $<$ inequality, so instead of a measure of *distance* or *dissimilarity*, the discriminant functions represent the *likelihood* or *propensity* of an observation to belong to a particular group. This turns out to be more natural and convenient, especially in the context of model-based methods for classification. Our distance-based methods can be easily viewed within this framework by introducing a minus sign. Thus, our closest group mean classifier corresponds to the discriminant functions

$$Q_i(\boldsymbol{x}) = -\|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2, \ i = 1, 2, \ldots, k,$$

and the LDA method corresponds to

$$Q_i(\boldsymbol{x}) = -(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i), \ i = 1, 2, \ldots, k.$$

Our first attempt at a quadratic classification rule corresponds to the discriminant functions,

$$Q_i(\boldsymbol{x}) = -(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} {\Sigma_i}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i), \ i = 1, 2, \ldots, k,$$

but as previously mentioned, we will improve on this in due course.

It turns out that most classification methods of practical interest can be cast into the framework of discriminant functions, and having a common framework can be useful in order to develop generic techniques for the computation, interpretation and analysis of classification methods and procedures that are not specific to a particular discrimination technique.

There are many different approaches that can be taken to classification. Many, such as those we have been considering so far, are heuristically derived. However, given a particular set of discriminant functions, we can study the properties of the resulting classifier, such as misclassification rates, either empirically, or theoretically, to try and establish whether the method is good or bad.

Other approaches to classification start from modelling assumptions, and derive classification rules which are "optimal" in some narrowly defined sense. We will next examine one such approach, based on the assumption of multivariate normality and the principle of maximum likelihood.

5.3 Maximum likelihood discrimination

Model-based approaches to classification assume a probability model for each group, of the form $f_i(\cdot) : \mathbb{R}^p \longrightarrow \mathbb{R}, \ i = 1, 2, \ldots, k$. So for observations $\boldsymbol{x} \in \mathbb{R}^p$, each model $f_i(\boldsymbol{x})$ represents a probability density function (or likelihood) for the random variable $\boldsymbol{X}$ from group i. The *maximum likelihood discriminant rule* (MLDR) is to classify observations by assigning them to the group with the maximum likelihood. In other words, MLDR corresponds to using the discriminant functions

$$Q_i(\boldsymbol{x}) = f_i(\boldsymbol{x}), \quad i = 1, 2, \ldots, k.$$

5.3.1 LDA

The simplest case of maximum likelihood discrimination arises in the case where it is assumed that observations from all groups are multivariate normal, and all share a common

variance matrix, Σ. That is observations from group i are assumed to be iid $N(\boldsymbol{\mu}_i, \Sigma)$ random variables. In other words,

$$Q_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) = (2\pi)^{-p/2}|\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}.$$

We can simplify things by noting that

$$\begin{aligned} & Q_i(\boldsymbol{x}) > Q_j(\boldsymbol{x}) \\ \Leftrightarrow (2\pi)^{-p/2}|\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\} & > (2\pi)^{-p/2}|\Sigma|^{-1/2} \\ & \quad \times \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right\} \\ \Leftrightarrow \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\} & > \\ & \quad \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right\} \\ & \quad \Leftrightarrow -(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) > \\ & \quad\quad -(\boldsymbol{x}-\boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j), \end{aligned}$$

and so the MLDR is in fact exactly equivalent to using

$$Q_i(\boldsymbol{x}) = -(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i).$$

But this is just classification by minimising Mahalanobis distance, and hence corresponds to Fisher's linear discriminant, which allocates to group i if

$$(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^\mathsf{T}\Sigma^{-1}\left[\boldsymbol{x} - \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)\right] > 0, \ \forall j \neq i.$$

So in the case of equal variance matrices, the MLDR corresponds exactly to Fisher's linear discriminant.

5.3.2 Quadratic discriminant analysis (QDA)

It is natural to next consider how the MLDR changes when we allow the variance matrices associated with each group to be unequal. That is, we assume that observations from group i are iid $N(\boldsymbol{\mu}_i, \Sigma_i)$. In this case we have

$$Q_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) = (2\pi)^{-p/2}|\Sigma_i|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^\mathsf{T}{\Sigma_i}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}.$$

We can simplify this a little by noting that

$$\begin{aligned} & Q_i(\boldsymbol{x}) > Q_j(\boldsymbol{x}) \\ \Leftrightarrow (2\pi)^{-p/2}|\Sigma_i|^{-1/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^{\mathsf{T}}{\Sigma_i}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\} & > (2\pi)^{-p/2}|\Sigma_j|^{-1/2} \\ & \quad \times \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^{\mathsf{T}}{\Sigma_j}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right\} \\ \Leftrightarrow |\Sigma_i|^{-1/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^{\mathsf{T}}{\Sigma_i}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\} & > \\ |\Sigma_j|^{-1/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^{\mathsf{T}}{\Sigma_j}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right\} & \\ \Leftrightarrow -\log|\Sigma_i| - (\boldsymbol{x}-\boldsymbol{\mu}_i)^{\mathsf{T}}{\Sigma_i}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) & > \\ -\log|\Sigma_j| - (\boldsymbol{x}-\boldsymbol{\mu}_j)^{\mathsf{T}}{\Sigma_j}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j), & \end{aligned}$$

and so the MLDR is exactly equivalent to using the discriminant function

$$Q_i(\boldsymbol{x}) = -\log|\Sigma_i| - (\boldsymbol{x}-\boldsymbol{\mu}_i)^{\mathsf{T}}{\Sigma_i}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i).$$

Note that this is a quadratic form in $\boldsymbol{x}$, but is different to the quadratic discriminant function we derived heuristically, due to the presence of the $\log|\Sigma_i|$ term, which has the effect of appropriately penalising the distances associated with groups having large variances. This penalty term, which corresponds to the normalisation constant of the density, generally improves the performance of the classifier, and hence is the form typically used in QDA.

The binary case

In the $k = 2$ case, we assign to group 1 if $Q_1(\boldsymbol{x}) > Q_2(\boldsymbol{x})$, that is

$$\begin{aligned} & -\log|\Sigma_1| - (\boldsymbol{x}-\boldsymbol{\mu}_1)^{\mathsf{T}}{\Sigma_1}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) > -\log|\Sigma_2| - (\boldsymbol{x}-\boldsymbol{\mu}_2)^{\mathsf{T}}{\Sigma_2}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2) \\ \Leftrightarrow\ & \log|\Sigma_1| + (\boldsymbol{x}-\boldsymbol{\mu}_1)^{\mathsf{T}}{\Sigma_1}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) < \log|\Sigma_2| + (\boldsymbol{x}-\boldsymbol{\mu}_2)^{\mathsf{T}}{\Sigma_2}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2). \end{aligned}$$

Multiplying out and gathering terms gives

$$\boldsymbol{x}^{\mathsf{T}}({\Sigma_1}^{-1} - {\Sigma_2}^{-1})\boldsymbol{x} + 2(\boldsymbol{\mu}_2{}^{\mathsf{T}}{\Sigma_2}^{-1} - \boldsymbol{\mu}_1{}^{\mathsf{T}}{\Sigma_1}^{-1})\boldsymbol{x} + \boldsymbol{\mu}_1{}^{\mathsf{T}}{\Sigma_1}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2{}^{\mathsf{T}}{\Sigma_2}^{-1}\boldsymbol{\mu}_2 + \log\frac{|\Sigma_1|}{|\Sigma_2|} < 0.$$

Here we can see explicitly that the quadratic term does not cancel out, and that the boundary between the two classes corresponds to the contour of a quadratic form.

5.3.3 Estimation from data

Typically the group mean vectors and variance matrices are not known in advance, but can be estimated from data that has already been classified, typically referred to as a *training set*. For the MLDR, maximum likelihood parameter estimates should be plugged into the discriminant functions. So, one can start with the group sample means $\bar{\boldsymbol{x}}_i$ and

plug these in for the $\boldsymbol{\mu}_i$, for both LDA and QDA. The variance matrices are slightly more complicated. For QDA, the group sample variance matrices, S_i can be plugged in for Σ_i. Technically, a divisor of n_i rather than $n_i - 1$ should be used, since it is the MLE of variance that is required, but it will obviously not make much difference if n_i is large. However, if $\hat{\Sigma}_i = (n_i - 1)\mathsf{S}_i/n_i$ is used, the discriminant function becomes

$$Q_i(\boldsymbol{x}) = -\log|\hat{\Sigma}_i| - (\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^{\mathsf{T}}\hat{\Sigma}_i{}^{-1}(\boldsymbol{x} - \bar{\boldsymbol{x}}_i).$$

For LDA, a single pooled estimate of variance is required. The unbiased pooled estimate of Σ is

$$\mathsf{S}_W = \frac{1}{n-k}\sum_{i=1}^{k}(n_i - 1)\mathsf{S}_i,$$

and this can be plugged in for Σ. The corresponding MLE is

$$\hat{\Sigma} = \frac{n-k}{n}\mathsf{S}_W,$$

but note that here it doesn't matter what divisor is used, since it will cancel out of Fisher's linear discriminant. Also note that it is not appropriate to use the overall sample variance matrix, S, since the means within the groups are different. The discriminant function becomes

$$Q_i(\boldsymbol{x}) = -(\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^{\mathsf{T}}\mathsf{S}_W{}^{-1}(\boldsymbol{x} - \bar{\boldsymbol{x}}_i).$$

In this case, Fisher's linear discriminant for $k = 2$ takes the form, allocate to group 1 if

$$(\bar{\boldsymbol{x}}_1 - \bar{\boldsymbol{x}}_2)^{\mathsf{T}}\mathsf{S}_W{}^{-1}\left\{\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{x}}_1 + \bar{\boldsymbol{x}}_2)\right\} > 0.$$

Example: Toy problem

Now that we know how to use data in order to construct discriminant rules, we can see how to use **R** to do so. We will first consider simulated data based on the simple toy model considered previously. We will simulate 20 observations from each group, using the group means and common variance previously given, and store the results in a 40×2 matrix, `X`, and a group vector `Class`.

```
mu1=c(-1,2)
mu2=c(2,1)
diff=mu1-mu2
ave=0.5*(mu1+mu2)
Sigma=matrix(c(1,1,1,4),ncol=2)
Gt=chol(Sigma)
Z=matrix(rnorm(80),ncol=2)
W=Z %*% Gt
shift1=matrix(rep(mu1,each=20),ncol=2)
shift2=matrix(rep(mu2,each=20),ncol=2)
shift=rbind(shift1,shift2)
X=W+shift
Class=rep(1:2,each=20)
```

We can plot the data, and overlay the decision boundaries we computed earlier as follows.

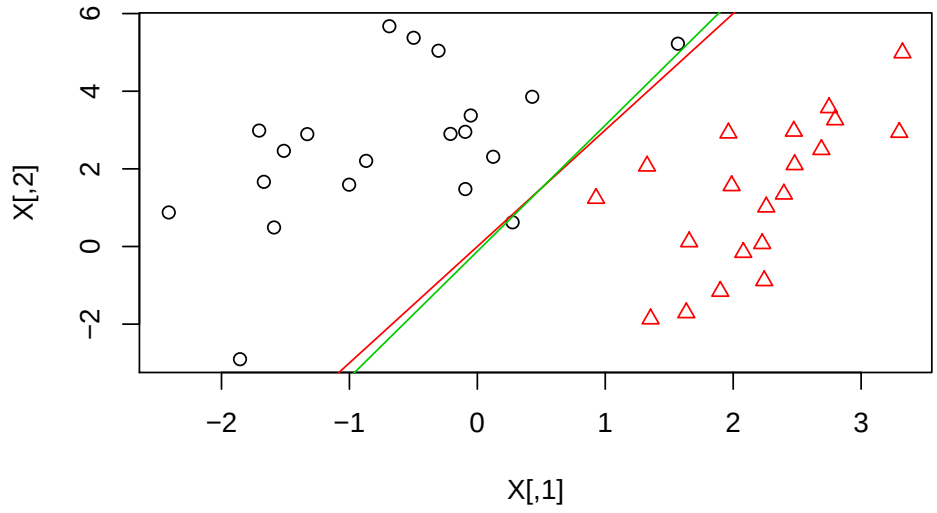


Figure 5.1: Simulated data with two different classification boundaries overlaid

```
plot(X,pch=Class,col=Class)
abline(0,3,col=2)
abline(-1/8,13/4,col=3)
```

This gives the plot shown in Figure 5.1.

We can compute the sample means and pooled variance estimate, and then construct the normal for Fisher's discriminant rule using:

```
> xbar1=colMeans(X[Class==1,])
> xbar2=colMeans(X[Class==2,])
> V1=var(X[Class==1,])
> V2=var(X[Class==2,])
> V=(19*V1+19*V2)/(40-2)
> normal=solve(V,xbar1-xbar2)
> normal
[1] -6.41715  1.76622
```

We can then also use the `lda()` function from the `MASS` package to automate the process, as

```
> require(MASS)
> X.lda=lda(X,Class,prior=c(0.5,0.5))
> X.lda
Call:
lda(X, grouping = Class, prior = c(0.5, 0.5))

Prior probabilities of groups:
  1   2
0.5 0.5

Group means:
```

```
           1        2
1 -0.6744606 2.554725
2  2.1881006 1.346963

Coefficients of linear discriminants:
           LD1
[1,]  1.4540364
[2,] -0.4002007
```

Setting the `prior` option to be uniform over the two classes ensures that we obtain the MLDR. However, leaving this option out allows the `lda()` function to estimate the group frequencies and use this in the classification rule. This turns out to be very helpful in general — see the section on Bayesian classification at the end of the chapter.

By comparing the ratios of elements, it should be clear that the normal computed by the `lda()` function is in exactly the same direction as that which we computed explicitly. The object `X.lda` can be passed into the **`predict()`** function in order to make actual classifications on new data. For example, to classify the single observation $\boldsymbol{x} = (-3, 1)$, we can use **`predict`**`(X.lda,`**`c`**`(-3,1))$`**`class`**, but we can also pass in a 2-column matrix and get back a vector of classifications. For example, if we want to classify the data we used in order to build the discriminant function (in order to see how well the classifier works), then we can do so using

```
> predict(X.lda,X)$class
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
[34] 2 2 2 2 2 2 2
Levels: 1 2
```

We can also use **R** to carry out QDA for us, using the `qda()` function (also from the `MASS` package).

```
X.qda=qda(X,Class)
X.qda=qda(X,Class,prior=c(0.5,0.5))
X.qda
```

The object returned from the `qda()` function can also be passed into the **`predict()`** function in order to classify new data.

5.4 Misclassification

Obviously, whatever discriminant functions we use, we will not characterise the group of interest perfectly, and so some future observations will be classified incorrectly. An obvious way to characterise a classification scheme is by some measure of the degree of misclassification associated with the scheme. For a fully specified model, we can define a $k \times k$ matrix P of classification probabilities with elements

$$p_{ij} = \mathrm{P}(\text{allocate to group } i | \text{observation from group } j)\,.$$

Ideally we would like this to be I_k, but in practice the best we can hope for is that the diagonal elements are close to one, and that the off-diagonal elements are close to zero.[†]

[†]Note that P is a *left* stochastic matrix, having similar properties (such as columns summing to one) as the transition matrix of a Markov chain.

One way of characterising the overall quality of a classification scheme would be using $\mathrm{Tr}(\mathrm{P})/k$, with larger values of this quality score representing better schemes, and a maximum value of 1 indicating a perfect classification scheme.

For simple fully specified models and classification schemes, it may be possible to compute the p_{ij} directly, analytically (see homework assignment). For models where the parameters have been estimated from data, it may be possible to estimate $\hat{p}_{ij}$, the probability obtained from the model by plugging in MLEs for the model parameters. Even where this is possible, it is likely to lead to over-estimates of the diagonal elements and under-estimates of the off-diagonal elements, due to over-fitting, and ignoring sampling variability in the parameter estimates.

Where there is no underlying model, or the model or classification scheme is complex, it is quite likely that it will not be possible to compute P directly. In this case we can obtain empirical estimates of the p_{ij} directly from data. The plug-in method estimates the p_{ij} by re-using the data originally used to derive the classification rules. So, if we define n_{ij} to be the number of class j observations allocated to class i (so that $n_j = \sum_{i=1}^{k} n_{ij}$), we can estimate p_{ij} by

$$\hat{p}_{ij} = \frac{n_{ij}}{n_j}.$$

That is, we estimate the probability that a class j observation is classified as class i by the observed proportion of class j observations that are classified as class i. Although this method is very simple, and very widely used, it also leads to over-optimistic estimates of the quality of the classification scheme by testing the classifier on the data used in its construction (over-fitting).

Example: Toy problem

We can create classification tables in **R** for the simple examples we considered earlier by using the `table()` command.

```
> tab=table(Predicted=predict(X.lda)$class,Actual=Class)
> tab
         Actual
Predicted  1  2
        1 20  0
        2  0 20
>
> tab=table(Predicted=predict(X.qda)$class,Actual=Class)
> tab
         Actual
Predicted  1  2
        1 20  0
        2  0 20
```

A more reliable way of estimating the performance of a classifier in the case of very large n is to split the data set (randomly) into two, and then use one half of the data set to train the classifier and the other half to test the performance of the classifier on independent data.

Example: handwritten digits

We can use the data in the file `zip.train` in order to build a linear classifier for the handwritten digits, and then test this classifier on the data `zip.test`.

```
> zip.lda=lda(zip.train[,-1],zip.train[,1],prior=rep(0.1,10))
> tab=table(Predicted=predict(zip.lda,zip.test[,-1])$class,Actual=zip.
   test[,1])
> tab
         Actual
Predicted   0   1   2   3   4   5   6   7   8   9
        0 342   0   7   3   1   4   0   0   5   0
        1   0 251   2   0   4   0   0   1   0   0
        2   0   0 155   3   6   0   3   0   2   0
        3   4   2   4 142   0  17   0   2  11   0
        4   3   5  12   3 174   3   3   7   7   4
        5   1   0   2   9   0 125   3   0   4   0
        6   4   3   1   0   2   0 158   0   0   0
        7   0   0   1   1   2   0   0 129   0   5
        8   4   1  14   4   1   6   3   1 135   3
        9   1   2   0   1  10   5   0   7   2 165
> tab2=t(t(tab)/colSums(tab))
> round(tab2,digits=2)
         Actual
Predicted    0    1    2    3    4    5    6    7    8    9
        0 0.95 0.00 0.04 0.02 0.00 0.02 0.00 0.00 0.03 0.00
        1 0.00 0.95 0.01 0.00 0.02 0.00 0.00 0.01 0.00 0.00
        2 0.00 0.00 0.78 0.02 0.03 0.00 0.02 0.00 0.01 0.00
        3 0.01 0.01 0.02 0.86 0.00 0.11 0.00 0.01 0.07 0.00
        4 0.01 0.02 0.06 0.02 0.87 0.02 0.02 0.05 0.04 0.02
        5 0.00 0.00 0.01 0.05 0.00 0.78 0.02 0.00 0.02 0.00
        6 0.01 0.01 0.01 0.00 0.01 0.00 0.93 0.00 0.00 0.00
        7 0.00 0.00 0.01 0.01 0.01 0.00 0.00 0.88 0.00 0.03
        8 0.01 0.00 0.07 0.02 0.00 0.04 0.02 0.01 0.81 0.02
        9 0.00 0.01 0.00 0.01 0.05 0.03 0.00 0.05 0.01 0.93
> sum(diag(tab2))/10
[1] 0.8745323
```

We see that the classification is generally good, but '2's are sometimes misclassified as '4's or '8's, and similarly, '5's are sometimes misclassified as '3's or '8's, etc.

We can repeat the analysis without requiring a uniform prior as follows.

```
> zip.lda=lda(zip.train[,-1],zip.train[,1])
> tab=table(Predicted=predict(zip.lda,zip.test[,-1])$class,Actual=zip.
   test[,1])
> tab
         Actual
Predicted   0   1   2   3   4   5   6   7   8   9
        0 342   0   7   3   1   6   1   0   5   0
        1   0 251   2   0   4   0   0   1   0   0
        2   0   0 157   3   6   0   3   0   2   0
        3   4   2   4 142   0  16   0   2  11   0
```

```
         4   3   5  12   3 174   3   3   7   7   4
         5   1   0   2   9   0 125   3   0   4   0
         6   5   3   1   0   2   0 157   0   0   0
         7   0   0   1   1   2   0   0 129   0   5
         8   3   1  12   4   1   5   3   1 135   3
         9   1   2   0   1  10   5   0   7   2 165
> tab2=t(t(tab)/colSums(tab))
> round(tab2,digits=2)
          Actual
Predicted    0    1    2    3    4    5    6    7    8    9
         0 0.95 0.00 0.04 0.02 0.00 0.04 0.01 0.00 0.03 0.00
         1 0.00 0.95 0.01 0.00 0.02 0.00 0.00 0.01 0.00 0.00
         2 0.00 0.00 0.79 0.02 0.03 0.00 0.02 0.00 0.01 0.00
         3 0.01 0.01 0.02 0.86 0.00 0.10 0.00 0.01 0.07 0.00
         4 0.01 0.02 0.06 0.02 0.87 0.02 0.02 0.05 0.04 0.02
         5 0.00 0.00 0.01 0.05 0.00 0.78 0.02 0.00 0.02 0.00
         6 0.01 0.01 0.01 0.00 0.01 0.00 0.92 0.00 0.00 0.00
         7 0.00 0.00 0.01 0.01 0.01 0.00 0.00 0.88 0.00 0.03
         8 0.01 0.00 0.06 0.02 0.00 0.03 0.02 0.01 0.81 0.02
         9 0.00 0.01 0.00 0.01 0.05 0.03 0.00 0.05 0.01 0.93
> sum(diag(tab2))/10
[1] 0.8749542
```

Here the predictive performance improves only slightly (two additional '2's were correctly classified), due to the fact that the groups are approximately equally likely. However, in cases where groups are very unbalanced, estimation of prior probabilities can make a large difference to classification performance.

Using either method we see that we are able to automatically classify around 90% of digit images correctly using a simple LDA classifier. This is probably not good enough to be usable in practice, but nevertheless illustrates that simple linear algebraic methods can be used to solve very high-dimensional classification problems well.

5.5 Bayesian classification

The MLDR is often simple to implement, but only really works well when the k groups are all *a priori* equally likely. To see this, consider the binary classification case being applied to testing for a very rare disease. The MLDR blindly classifies observations to maximise the probability of the test result given disease status, when what we really want to maximise is the probability of disease status given the observation. We need to use Bayes theorem to do this, and this requires knowing the *a priori* group probabilities, which can also be estimated from data. This has the effect of introducing additional terms into the discriminant functions. Bayesian classifiers typically perform much better than heuristic methods, or methods based on MLDR, whenever groups are not approximately equally likely, which is almost always the case in practice.

We again assume that if an observation is from class i it has probability model $f_i(\boldsymbol{x})$, $i = 1, 2, \ldots, k$. We now introduce G to be the random variable denoting the unknown class of a given observation $\boldsymbol{x}$. From a Bayesian perspective, we are interested in $\mathrm{P}(G = i|\boldsymbol{X} = \boldsymbol{x})$, $i =$

$1, 2, \ldots, k$. Using Bayes Theorem we have

$$\begin{aligned} \mathrm{P}(G = i | \boldsymbol{X} = \boldsymbol{x}) &= \frac{\mathrm{P}(G = i)\, f_i(\boldsymbol{x})}{\sum_{j=1}^{k} \mathrm{P}(G = j)\, f_j(\boldsymbol{x})} \\ &\propto \mathrm{P}(G = i)\, f_i(\boldsymbol{x}) \\ &= p_i f_i(\boldsymbol{x}), \end{aligned}$$

where $p_i = \mathrm{P}(G = i)\,,\ i = 1, 2, \ldots, k$. So if we know the prior probabilities, we can assign an observation to the class with highest posterior probability by using the discriminant functions

$$Q_i(\boldsymbol{x}) = p_i f_i(\boldsymbol{x}),\ i = 1, 2, \ldots, k.$$

We can estimate the p_i from training data as the observed frequencies of observations from the different classes.

5.5.1 Bayesian LDA

In the case where $f_i(\boldsymbol{x}) = N(\boldsymbol{\mu}_i, \Sigma)$, we have

$$\begin{aligned} Q_i(\boldsymbol{x}) &= p_i f_i(\boldsymbol{x}) \\ &= p_i (2\pi)^{-p/2} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i) \right\} \end{aligned}$$

So

$$\begin{aligned} & Q_i(\boldsymbol{x}) > Q_j(\boldsymbol{x}) \\ & \quad \Leftrightarrow p_i \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i) \right\} > p_j \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_j)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_j) \right\} \end{aligned}$$

and hence the Bayes classification rule is equivalent to using the discriminant functions

$$Q_i(\boldsymbol{x}) = \log p_i - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i).$$

The decision boundaries for this rule are also linear, since

$$\begin{aligned} Q_i(\boldsymbol{x}) > Q_j(\boldsymbol{x}) &\Leftrightarrow \log p_i - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i) > \log p_j - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_j)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_j) \\ &\Leftrightarrow \log \frac{p_i}{p_j} - \frac{1}{2} \left[2(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)^{\mathsf{T}} \Sigma^{-1} \boldsymbol{x} + (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) \right] > 0 \\ &\Leftrightarrow (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^{\mathsf{T}} \Sigma^{-1} \left[\boldsymbol{x} - \frac{1}{2} (\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) \right] > \log \frac{p_j}{p_i}. \end{aligned}$$

This is similar to the MLDR, but with a shift in the linear boundary when the prior class probabilities are not equal.

5.6 Conclusion

See section 4.3 (p.106) of **[ESL]** for further information about techniques for classification.

Chapter 6

Graphical modelling

6.1 Introduction

Variables of multivariate distributions and data sets are often correlated in complex ways. It would be convenient if many variables were independent, leading to a sparse independence structure, as this would lead to many simplifications of much multivariate theory and methodology. Unfortunately variables tend not to be independent in practice. It is more often the case that variables are conditionally independent. Although not quite as strong a property as independence, this too turns out to be enough to lead to considerable simplification of a general dependence structure, and is useful for both conceptual and theoretical understanding, and also for computational efficiency. Sparse conditional independence structures can be represented using graphs, and this leads to the theory of graphical models. But before we explore graphical models, we must first ensure that we understand the notion of conditional independence.

6.2 Independence, conditional independence and factorisation

Let us begin by reminding ourselves of the essential notion of independence of events and random variables. For events A and B, we say that A and B are independent, and write $A \perp B$, if

$$\mathrm{P}(A \cap B) = \mathrm{P}(A)\ \mathrm{P}(B)\,.$$

Note that for $\mathrm{P}(B) > 0$ we get

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

which captures the intuitive notion of independence, that learning the outcome of B should not affect the probability of A.

The notion extends straightforwardly to (continuous) random quantities, where now we say that variables X and Y are independent and write $X \perp Y$ if their joint density factorises as

$$f_{X,Y}(x, y) = f_X(x) f_Y(y).$$

Again, for $f_Y(y) > 0$ we have

$$f_{X|Y}(x|y) = \frac{f_{X,Y}(x, y)}{f_Y(y)} = f_X(x),$$

capturing the intuitive notion that the conditional distribution of X given $Y = y$ should not depend on y. That is, the conditional and marginal distributions should be the same.

It turns out that independence is a special case of the more general notion of *conditional independence*. For events A, B and C, we say that A and B are conditionally independent given C, and write $A \perp\!\!\!\perp B|C$, if

$$\mathrm{P}(A \cap B|C) = \mathrm{P}(A|C)\ \mathrm{P}(B|C)\,.$$

Now, when $\mathrm{P}(B|C) > 0$ we have

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

and so conditional on C, learning the outcome of B does not change the probability of A.

This notion again generalises easily to (continuous) random quantities, so that for random variables X, Y and Z, we have that X and Y are conditionally independent given Z, written $X \perp\!\!\!\perp Y|Z$ if the conditional density of X and Y given $Z = z$ factorises as

$$f_{X,Y|Z}(x, y|z) = f_{X|Z}(x|z) f_{Y|Z}(y|z).$$

So then if $f_{Y|Z} > 0$ we have

$$f_{X|Y,Z}(x|y, z) = \frac{f_{X,Y|Z}(x, y|z)}{f_{Y|Z}(y|z)} = f_{X|Z}(x|z).$$

There are numerous other important properties of this factorisation for the joint density of X, Y and Z, $f_{X,Y,Z}(x, y, z)$.

Proposition 41 *If $X \perp\!\!\!\perp Y|Z$ then the joint density factorises as*

$$f_{X,Y,Z}(x, y, z) = f_Z(z) f_{X|Z}(x|z) f_{Y|Z}(y|z).$$

This factorisation property turns out to be key to understanding directed acyclic graph (DAG) models.

Proof

$$f_{X,Y,Z}(x, y, z) = f_Z(z) f_{X,Y|Z}(x, y|z) = f_Z(z) f_{X|Z}(x|z) f_{Y|Z}(y|z).$$

□

Proposition 42 *If $X \perp\!\!\!\perp Y|Z$ and $f_Z(z) > 0$, then the joint density factorises as*

$$f_{X,Y,Z}(x, y, z) = \frac{f_{X,Z}(x, z) f_{Y,Z}(y, z)}{f_Z(z)}.$$

This factorisation property turns out to be important for understanding undirected graphical models.

Proof

$$\begin{aligned} f_{X,Y,Z} &= f_Z(z) f_{X|Z}(x|z) f_{Y|Z}(y|z) \\ &= f_Z(z) \frac{f_{X,Z}(x,z)}{f_Z(z)} \frac{f_{Y,Z}(y,z)}{f_Z(z)} \\ &= \frac{f_{X,Z}(x,z) f_{Y,Z}(y,z)}{f_Z(z)}. \end{aligned}$$

□

Proposition 43 *Assuming that $f_Z(z) > 0$, we have that $X \perp\!\!\!\perp Y|Z$ if and only if the joint density factorises in the form*

$$f_{X,Y,Z}(x,y,z) = h(x,z)k(y,z),$$

for some bivariate functions $h(\cdot,\cdot)$ and $k(\cdot,\cdot)$.

Proof

First note that the forward implication follows from Proposition 42. That is, if $X \perp\!\!\!\perp Y|Z$, then there exist functions $h(\cdot,\cdot)$ and $k(\cdot,\cdot)$ such that

$$f_{X,Y,Z}(x,y,z) = h(x,z)k(y,z).$$

For example, we could choose $h(x,z) = f_{X,Z}(x,z)$ and $k(y,z) = f_{Y,Z}(y,z)/f_Z(z)$, but there are obviously many other possibilities.

The reverse direction is less clear. We assume that we have the factorisation

$$f_{X,Y,Z}(x,y,z) = h(x,z)k(y,z),$$

and want to prove that

$$f_{X,Y|Z}(x,y|z) = f_{X|Z}(x|z) f_{Y|Z}(y|z).$$

This can be demonstrated with straightforward algebraic manipulation, but is left as an exercise. □

Note that although we may have been thinking about univariate random quantities X, Y and Z, nothing we have discussed is specific to the univariate case, and all applies similarly to the case of multivariate random quantities $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$.

The above factorisation results are very powerful, and can be used to prove a number of general algebraic properties associated with independence and conditional independence of random quantities.

Proposition 44

1. $X \perp\!\!\!\perp Y|Z \Leftrightarrow Y \perp\!\!\!\perp X|Z$
2. $X \perp\!\!\!\perp Y|Z \Rightarrow h(X) \perp\!\!\!\perp Y|Z$
3. $X \perp\!\!\!\perp Y|Z \Rightarrow X \perp\!\!\!\perp Y|(Z, h(X))$
4. $X \perp\!\!\!\perp Y|Z$ *and* $X \perp\!\!\!\perp W|(Y, Z) \Leftrightarrow X \perp\!\!\!\perp (W, Y)|Z$
5. *If all densities are positive, then*

$$X \perp\!\!\!\perp Y|Z \text{ and } X \perp\!\!\!\perp Z|Y \Rightarrow X \perp (Y, Z).$$

Proof
The proofs of these results are mainly straightforward, and left as an exercise. □

6.3 Undirected graphs

6.3.1 Graph theory

It turns out that when we have a large number of variables with many associated conditional independence statements, it is useful to represent known statements using a graph. The graphs used can be *directed* or *undirected*, and can be given several different interpretations. We begin with a reminder of some basic properties of undirected graphs, and then later examine how such graphs may be associated with conditional independence properties.

A *graph* $\mathcal{G}$ is a tuple $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of *vertices* (or *nodes*), and E is a finite set of *undirected edges*, with each edge $e \in E$ being of the form $e = \{v_i, v_j\}$, $v_i, v_j \in V$, $i \neq j$. There are clearly $\binom{n}{2} = n(n-1)/2$ possible edges of this form, and we write $\binom{V}{2}$ for the set of all such edges, so that $E \subseteq \binom{V}{2}$. A graph is said to be complete if $E = \binom{V}{2}$.

Example

Consider the graph $\mathcal{G} = (V, E)$ where $V = \{A, B, C\}$ and $E = \{\{A, C\}, \{B, C\}\}$. We can draw a pictorial representation of the graph as given in Figure 6.1. Note that this graph is not complete, since the edge $\{A, B\}$ is missing. Also note that we can plot this graph in **R** using the `ggm` package, using the following commands

```
require(ggm)
drawGraph(UG(~A*C+B*C))
```

Note that the `drawGraph()` function allows moving nodes around using a very simple point-and-click interface. The command `plotGraph()` can also be used, and has a more sophisticated interface, but may not be available on all systems.

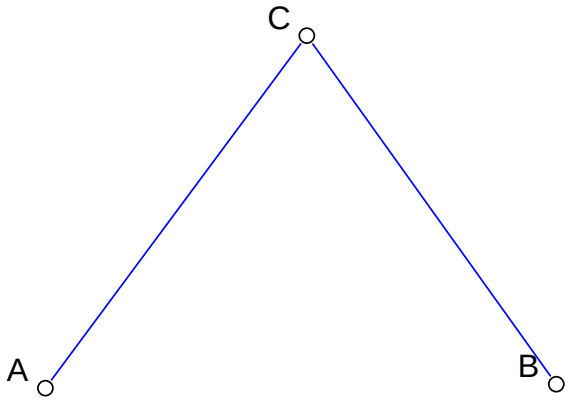


Figure 6.1: A graph with 3 vertices and 2 edges

Vertices $v_i, v_j \in V$ are said to be *adjacent*, written $v_i \sim v_j$, if $\{v_i, v_j\} \in E$. Any subset of vertices $U \subseteq V$ induces a *subgraph* $\mathcal{G}_U = (U, F)$, where

$$F = \{\{u, v\} \in E | u, v \in U\}.$$

Example

For the graph considered in the previous example, nodes A and C are adjacent ($A \sim C$), but nodes A and B are not. Similarly, the subgraph $\mathcal{G}_{\{A,C\}}$ is complete, but the subgraph $\mathcal{G}_{\{A,B\}}$ is not.

A subset $U \subseteq V$ is a (maximal) clique if it is *maximally complete*. That is, the associated subgraph $\mathcal{G}_U$ is complete, and for all strict supersets of U, W (so that $U \subset W \subseteq V$), the induced subgraph $\mathcal{G}_W$ is not complete.

Example

Again, considering again the previous example, the graph $\mathcal{G}$ has two cliques, $\{A, C\}$ and $\{B, C\}$.

Example

The previous example is somewhat uninteresting in that the cliques correspond with edges. So consider now the graph $\mathcal{G} = (V, E)$ where $V = \{D, E, F, G\}$ and

$$E = \{\{D, E\}, \{D, F\}, \{E, F\}, \{E, G\}\}.$$

Draw the graph and write down the cliques. How many cliques are there? The cliques

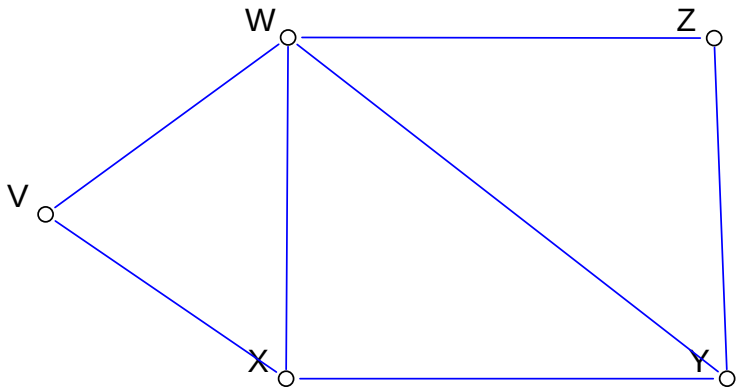


Figure 6.2: A graph with 5 vertices and 3 cliques

are $\{D, E, F\}$ and $\{E, G\}$. There are two cliques.

Note that the set of cliques defines the graph. That is, complete knowledge of all cliques of the graph allows construction of the graph.

Example

Consider the graph $\mathcal{G}$ which has three cliques: $\{V, W, X\}$, $\{W, X, Y\}$ and $\{W, X, Z\}$. We can construct and plot this graph in **R** using

```
drawGraph(UG(~V*W*X+W*X*Y+W*Y*Z))
```

leading to the plot shown in Figure 6.2.

The *neighbours* of a node $v \in V$, often known as the *boundary* of v, written $\mathrm{bd}(v)$, is $\{u \in V | u \sim v\}$. The *closure* of v, written $\mathrm{cl}(v)$, is given by $\mathrm{cl}(v) = v \cup \mathrm{bd}(v)$. A path is a sequence $x_0, x_1, \ldots, x_m \in V$ such that $x_{i-1} \sim x_i$, $i = 1, 2, \ldots, m$. A graph is connected if there exists a path between every pair of vertices. A *component* is a maximal connected subgraph. A cycle is a (non-trivial) path starting and ending at the same vertex. A connected graph is a tree if it does not contain any cycles. A *leaf* is any node of a tree connected to just one edge. A *forest* is a graph with components which are all trees.

For any undirected graph $\mathcal{G} = (V, E)$ with vertex subsets $A, B, C \subseteq V$, we say that C separates A and B if all paths from a node in A to a node in B pass through a node in C.

6.3.2 Graphical models

Now, given a collection of conditional independence statements relating to a collection of random variables, we can define a graph by associating nodes with variables, and using

the edge structure to encode the conditional independence structure. This is useful for both visualisation and computational analysis. It turns out that there are several different ways that we can do this, but they all turn out to be equivalent in practice in most cases.

Given a set of random variables $X_1, X_2, \ldots, X_n$, we can define an associated graph $\mathcal{G} = (V, E)$, where $V = \{X_1, X_2, \ldots, X_n\}$, and E is a set of edges.

Definition 10 *We say that $\mathcal{G}$ has the* factorisation property *(F) if the joint density of the random variables factorises in the form*

$$f_X(x) = \prod_{c \in \mathcal{C}} \phi_c(x_c),$$

for some functions $\phi_c(\cdot)$, where $\mathcal{C}$ denotes the set of cliques associated with the graph $\mathcal{G}$.

Definition 11 *We say that $\mathcal{G}$ has the* global Markov property *(G) if for any disjoint vertex subsets A, B and C such that C separates A and B in G we have $A \perp\!\!\!\perp B | C$.*

Definition 12 *We say that $\mathcal{G}$ has the* local Markov property *(L) if for all $v \in V$ we have*

$$v \perp\!\!\!\perp V \setminus \mathrm{cl}(v) | \mathrm{bd}(v).$$

Definition 13 *We say that $\mathcal{G}$ has the* pairwise Markov property *(P) if for all $u, v \in V$ such that $u \not\sim v$ we have*

$$u \perp\!\!\!\perp v | V \setminus \{u, v\}.$$

We have just presented four different ways that one can associate a graph with a conditional independence structure of a set of random variables. If these different interpretations all turn out to be fundamentally different, this is potentially confusing. Fortunately, it turns out that these different interpretations are all closely linked.

Proposition 45 *(F) $\Rightarrow$ (G) $\Rightarrow$ (L) $\Rightarrow$ (P)*

This important result tells us that if a graph satisfies the factorisation property (F), then the other properties all follow.

Proof

Let's start with the simplest result, (G) $\Rightarrow$ (L): This is trivial, since by definition, $\mathrm{bd}(v)$ separates v from $V \setminus \mathrm{cl}(v)$.

Now let us consider (L) $\Rightarrow$ (P): Suppose (L), so that we know

$$v \perp\!\!\!\perp V \setminus \mathrm{cl}(v) | \mathrm{bd}(v).$$

Now using property 3 of Proposition 44, we can "copy" vertices from $V \setminus \mathrm{cl}(v)$ to the conditioning set as desired in order to deduce that

$$v \perp\!\!\!\perp V \setminus \mathrm{cl}(v) | V \setminus \{u, v\}.$$

Finally, since u and v are not adjacent, we know that $u \in V \setminus \mathrm{cl}(v)$, and so using property 2 of Proposition 44, we conclude that

$$v \perp\!\!\!\perp u | V \setminus \{u, v\},$$

and (P) is satisfied.

Finally, let us consider (F) $\Rightarrow$ (G): Assume (F), and suppose that A, B and C are disjoint subsets of V with C separating A and B. The graph $\mathcal{G}_{V\setminus C}$ can't be connected, so let $\tilde{A}$ be the components of $\mathcal{G}_{V\setminus C}$ containing vertices in A, so that $A \subseteq \tilde{A}$ and $B \cap \tilde{A} = \emptyset$ due to the separation assumption. Now put $\tilde{B} = V\setminus(\tilde{A} \cup C)$ so that $B \subseteq \tilde{B}$, and V is the disjoint union of $\tilde{A}$, $\tilde{B}$ and C. Again, due to the separation assumption, every $c \in \mathcal{C}$ must be such that $c \subseteq \tilde{A} \cup C$ or $c \subseteq \tilde{B} \cup C$. Call the set of all cliques in $\tilde{A} \cup C$, $\mathcal{C}_A$, and the rest $\mathcal{C}_B$. Using (F), we have

$$f(x) = \prod_{c\in\mathcal{C}} \phi_c(x) = \prod_{c\in\mathcal{C}_A} \phi_c(x) \prod_{c\in\mathcal{C}_B} \phi_c(x) = h(x_{\tilde{A}\cup C})k(x_{\tilde{B}\cup C}),$$

and so $\tilde{A} \perp\!\!\!\perp \tilde{B}|C$. Now using property 2 of Proposition 44 we conclude first that $A \perp\!\!\!\perp \tilde{B}|C$, since $A \subseteq \tilde{A}$ and then that $A \perp\!\!\!\perp B|C$, since $B \subseteq \tilde{B}$. That is, (G) is satisfied. □

Proposition 46 (Hammersley-Clifford) *If all densities are positive, then (P) $\Rightarrow$ (F).*

This important (and difficult) result is stated without proof. However, the implication is that in most practical cases, all four interpretations of a graph of conditional independence structures are equivalent, and may be considered interchangeably.

Corollary 1 *If all densities are positive, then (F) $\Leftrightarrow$ (G) $\Leftrightarrow$ (L) $\Leftrightarrow$ (P).*

6.4 Gaussian graphical models (GGMs)

We now turn attention to the simplest parametrised class of graphical models — Gaussian graphical models (GGMs). These are simply multivariate normal distributions where the variables are subject to conditional independence properties. It turns out to be very straightforward to characterise GGMs in terms of the pairwise Markov property (P), but to understand why, we must first understand a little more about the MVN, and its parametrisation in terms of the *precision matrix*. If $\boldsymbol{X}$ is a random vector with variance matrix $\mathrm{Var}(\boldsymbol{X})$, then $\mathsf{Q} = \mathrm{Var}(\boldsymbol{X})^{-1}$ is the *precision matrix* for $\boldsymbol{X}$. So, if $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \Sigma)$, we can put $\mathsf{Q} = \Sigma^{-1}$, and then $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \mathsf{Q}^{-1})$. Note that we can express the density of the MVN in terms of $\boldsymbol{\mu}$ and Q as

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

Writing the density in this way allows us to partition $\boldsymbol{x}$, $\boldsymbol{\mu}$ and Q, and derive the conditional distribution of the first part of $\boldsymbol{x}$ given the rest.

Proposition 47 *For $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \mathsf{Q}^{-1})$, partition*

$$\boldsymbol{X} = \begin{pmatrix}\boldsymbol{X}_1\\ \boldsymbol{X}_2\end{pmatrix},\ \boldsymbol{x} = \begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{pmatrix},\ \boldsymbol{\mu} = \begin{pmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{pmatrix},\ \mathsf{Q} = \begin{pmatrix}\mathsf{Q}_{11} & \mathsf{Q}_{12}\\ \mathsf{Q}_{21} & \mathsf{Q}_{22}\end{pmatrix}.$$

Then

$$(\boldsymbol{X}_1|\boldsymbol{X}_2 = \boldsymbol{x}_2) \sim N(\boldsymbol{\mu}_{1|2}, {\mathsf{Q}_{11}}^{-1}),$$

where

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 - {\mathsf{Q}_{11}}^{-1}\mathsf{Q}_{12}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2).$$

Proof

$$
\begin{aligned}
f(\boldsymbol{x}_1|\boldsymbol{x}_2) &\propto f(\boldsymbol{x}) \\
&\propto \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\mathsf{Q}(\boldsymbol{x}-\boldsymbol{\mu})\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left[(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^{\mathsf{T}}\mathsf{Q}_{11}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)+\right.\right. \\
&\qquad + (\boldsymbol{x}_2-\boldsymbol{\mu}_2)^{\mathsf{T}}\mathsf{Q}_{22}(\boldsymbol{x}_2-\boldsymbol{\mu}_2) \\
&\qquad \left.\left. +2(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^{\mathsf{T}}\mathsf{Q}_{12}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)\right]\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{x}_1{}^{\mathsf{T}}\mathsf{Q}_{11}\boldsymbol{x}_1 - 2\boldsymbol{x}_1{}^{\mathsf{T}}\mathsf{Q}_{11}\boldsymbol{\mu}_1 + 2\boldsymbol{x}_1{}^{\mathsf{T}}\mathsf{Q}_{12}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)\right]\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{x}_1{}^{\mathsf{T}}\mathsf{Q}_{11}\boldsymbol{x}_1 - 2\boldsymbol{x}_1{}^{\mathsf{T}}\mathsf{Q}_{11}\boldsymbol{\mu}_{1|2}\right]\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left[(\boldsymbol{x}_1-\boldsymbol{\mu}_{1|2})^{\mathsf{T}}\mathsf{Q}_{11}(\boldsymbol{x}_1-\boldsymbol{\mu}_{1|2})\right]\right\}.
\end{aligned}
$$

□

Using this result, it is straightforward to see why zeroes of Q correspond to pairwise conditional independence statements.

First consider the case $q_{12} = 0\ (= q_{21})$. This is wlog, since we can always re-order the variables to ensure that the zero of interest is in this position.* Partition $\boldsymbol{X}$ as

$$
\boldsymbol{X} = \begin{pmatrix}\boldsymbol{X}_1\\ \boldsymbol{X}_2\end{pmatrix}, \text{ where } \boldsymbol{X}_1 = \begin{pmatrix}X_1\\ X_2\end{pmatrix} \text{ and } \boldsymbol{X}_2 = \begin{pmatrix}X_3\\ X_4\\ \vdots\\ X_p\end{pmatrix}.
$$

Then $\boldsymbol{X}_1|\boldsymbol{X}_2 = \boldsymbol{x}_2$ has variance $\mathsf{Q}_{11}{}^{-1}$, which must be diagonal, since Q_{11} is. Now this means that X_1 and X_2 are conditionally uncorrelated, but for the MVN, uncorrelated and independent are equivalent. Consequently $X_1 \perp\!\!\!\perp X_2|(X_3,\ldots,X_p)$, and so $q_{12} = 0$ leads directly to this CI statement. In general, we have the following result.

Proposition 48 *If $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \mathsf{Q}^{-1})$, then for $i \neq j$ we have*

$$
q_{ij} = 0\ (= q_{ji}) \Leftrightarrow X_i \perp\!\!\!\perp X_j|\boldsymbol{X}\backslash\{X_i, X_j\}.
$$

Therefore, there is a direct correspondence between the zero structure of Q and an undirected graph with the pairwise Markov property (P), where zeroes in Q correspond to missing edges in the graph. Further, provided that Q (or equivalently, the variance matrix, Σ) is strictly positive definite, the densities are all positive, and we can conclude that all of our properties (F), (G), (L) and (P) are satisfied by the associated graph.

*Note that if P is a permutation matrix, then P is orthogonal. Then $\mathrm{Var}(\mathsf{P}\boldsymbol{X}) = \mathsf{P}\,\mathrm{Var}(\boldsymbol{X})\,\mathsf{P}^{\mathsf{T}}$, and so $\mathrm{Var}(\mathsf{P}\boldsymbol{X})^{-1} = \mathsf{P}\,\mathrm{Var}(\boldsymbol{X})^{-1}\mathsf{P}^{\mathsf{T}}$, as can be verified by direct multiplication. In other words, the precision matrix for the re-ordered variables is just the precision matrix for the variables in their original order, but with rows and columns re-ordered to match.

6.4.1 Partial covariance and correlation

We have shown that the MVN has the important and interesting property that $\mathrm{Var}(\boldsymbol{X}_1|\boldsymbol{X}_2 = \boldsymbol{x}_2)$ does not depend on the observed value of $\boldsymbol{x}_2$. This allows us to consider the conditional variance matrix without reference to the observed values of the variables we are conditioning on. We define the *partial (co)variance matrix* for $\boldsymbol{X}_1$ (given $\boldsymbol{X}_2$) to be the conditional covariance matrix $\mathrm{Q}_{11}{}^{-1}$. In particular, in the case $\boldsymbol{X}_1 = (X_1, X_2)^\mathsf{T}$, we have

$$\mathrm{Q}_{11}{}^{-1} = \begin{pmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{pmatrix}^{-1} = \frac{1}{q_{11}q_{22} - q_{12}^2} \begin{pmatrix} q_{22} & -q_{12} \\ -q_{12} & q_{11} \end{pmatrix},$$

and so we define the *partial covariance* between X_1 and X_2 to be $-q_{12}/(q_{11}q_{22} - q_{12}^2)$. The corresponding *partial correlation* is therefore given by $-q_{12}/\sqrt{q_{11}q_{22}}$. In general, the partial correlation between X_i and X_j, for $i \neq j$ is given by

$$\frac{-q_{ij}}{\sqrt{q_{ii}q_{jj}}}.$$

Consequently, we can define the *partial correlation matrix* to be the matrix of partial correlations, and this is clearly given by

$$\mathrm{PCorr}(\boldsymbol{X}) = 2\,\mathrm{I}_p - \mathrm{D}^{-1/2}\mathrm{Q}\mathrm{D}^{-1/2}, \quad \text{where } \mathrm{D} = \mathrm{diag}\left\{q_{11}, \ldots, q_{pp}\right\}.$$

Note that in the typical case of positive definite Q, the zeroes of Q match those of $\mathrm{PCorr}(\boldsymbol{X})$. Consequently, we can understand the CI structure of a GGM by studying the zero structure of the partial correlation matrix.

Example

Suppose that $\boldsymbol{X} = (X_1, X_2, X_3)^\mathsf{T} \sim N(\boldsymbol{0}, \Sigma)$, where

$$\Sigma = \begin{pmatrix} 3 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 3 \end{pmatrix}.$$

Are there any conditional independence relations between the variables?

We can use **R** to answer this question. Note that as there are no zeroes in the variance matrix, Σ, there are no *marginal* independence relations between the variables. To look for *conditional* independence relations, we can compute the corresponding partial correlation matrix. We could do this by hand, by first using Gaussian elimination to compute the precision matrix, but it will be easier to use **R**.

```
> Sigma=matrix(c(3,-2,1,-2,4,-2,1,-2,3),ncol=3)
> Q=solve(Sigma)
> Q
     [,1] [,2] [,3]
[1,] 0.50 0.25 0.00
[2,] 0.25 0.50 0.25
[3,] 0.00 0.25 0.50
> 2*diag(3)-diag(diag(Q)^(-0.5))%*%Q%*%diag(diag(Q)^(-0.5))
```

```
     [,1] [,2] [,3]
[1,]  1.0 -0.5  0.0
[2,] -0.5  1.0 -0.5
[3,]  0.0 -0.5  1.0
```

Note that the `ggm` package includes the command `parcor()` for computing the partial correlation matrix directly from a variance matrix:

```
> parcor(Sigma)
     [,1] [,2] [,3]
[1,]  1.0 -0.5  0.0
[2,] -0.5  1.0 -0.5
[3,]  0.0 -0.5  1.0
```

which is much easier. We now see that the partial correlation in position (1,3) and (3,1) is zero, and this implies the pairwise Markov property, $X_1 \perp\!\!\!\perp X_3 | X_2$. Thus, the associated undirected graph will be a chain with X_2 separating X_1 and X_3.

This also gives us a potential strategy for estimating a GGM from data: compute the *sample* partial correlation matrix for the data set, and then use the zero structure of this estimated matrix in order to identify missing edges in the GGM. Now, following this procedure directly will typically lead to very dense graphs, since sample estimates of zero partial correlations will often be small, but not exactly zero. In this case we can apply a threshold to the sample partial correlation matrix, or attempt to apply statistical tests for whether or not particular elements are significantly different from zero.

Example: Mathematics students exam marks

The `ggm` package includes a very famous (old!) data set on exam marks of mathematics students in 5 subject areas. It can be loaded and plotted with

```
data(marks)
pairs(marks)
```

giving the plot shown in Figure 6.3. The plot shows a general positive correlation between marks in different subject areas (unsurprisingly), and this can be confirmed by inspecting the sample variance matrix for the data.

```
> Sigma=var(marks)
> round(Sigma,digits=2)
           mechanics vectors algebra analysis statistics
mechanics     305.69  127.04  101.47   106.32     117.49
vectors       127.04  172.84   85.16    94.67      99.01
algebra       101.47   85.16  112.89   112.11     121.87
analysis      106.32   94.67  112.11   220.38     155.54
statistics    117.49   99.01  121.87   155.54     297.76
```

There is clearly no indication that any of these variables are marginally independent. To investigate conditional independence structure, we inspect the partial correlation matrix.

```
> PCorr=parcor(Sigma)
> round(PCorr,digits=2)
           mechanics vectors algebra analysis statistics
mechanics       1.00    0.33    0.23     0.00       0.03
```

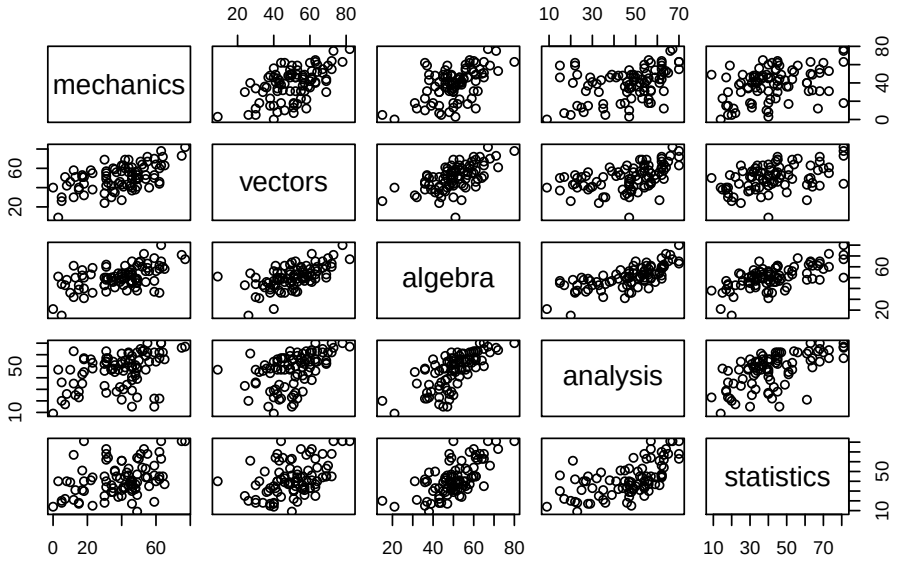


Figure 6.3: Pairwise scatter-plots of exam marks

```
vectors          0.33    1.00    0.28      0.08        0.02
algebra          0.23    0.28    1.00      0.43        0.36
analysis         0.00    0.08    0.43      1.00        0.25
statistics       0.03    0.02    0.36      0.25        1.00
```

We see immediately that, to 2dps, the partial correlation between mechanics and analysis is zero. However, it is also clear that the sample partial correlations between mechanics and statistics, and between vectors and statistics, are also very small, and probably not significantly different from zero. In fact, even the partial correlation between vectors and analysis is smaller than 0.1. If we arbitrarily threshold the partial correlations at a value of 0.1, then we immediately see the clique structure in the adjacency matrix:

```
> adj=1*(abs(PCorr)>0.1)
> adj
           mechanics vectors algebra analysis statistics
mechanics          1       1       1        0          0
vectors            1       1       1        0          0
algebra            1       1       1        1          1
analysis           0       0       1        1          1
statistics         0       0       1        1          1
```

and so we can plot the corresponding conditional independence graph with

```
drawGraph(adj)
```

giving the plot shown in Figure 6.4. What we see is that algebra separates mechanics and vectors from analysis and statistics. It is not that (say) ability in mechanics and statistics is uncorrelated, but that they are uncorrelated *given* ability in algebra. That is, although knowing how good someone is at mechanics is useful for predicting how good someone is at statistics, if we already knew how good they were at algebra, learning how good

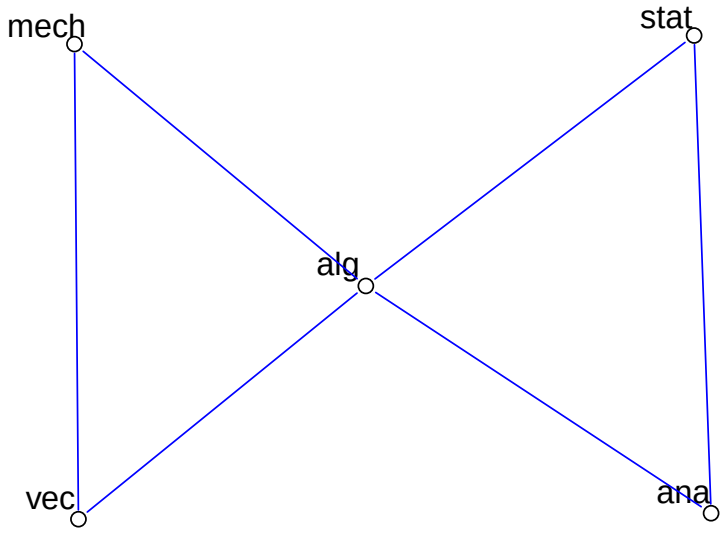


Figure 6.4: Possible conditional independence graph describing the exam marks data

they are at mechanics will give us no additional information about how good they are at statistics.

Obviously, there exist a range of statistical methods for estimating the variance and precision matrix of a GGM conditional on a given CI structure, and also methods for statistical testing of partial correlations, and methods for searching model space in a principled way. Unfortunately we do not have time to explore these methods in detail in this course, but note that the `ggm` package includes functions such as `fitConGraph()` and `pcor.test()` which can be useful in this context.

Example: galaxy data

Before moving on, it is worth looking briefly at the conditional independence structure of the variables in the `galaxy` data from the `ElemStatLearn` package. We can follow the procedure used above for the exam marks, as follows, leading to the plot shown in Figure 6.5.

```
> require(ElemStatLearn)
> Sigma=var(galaxy)
> PCorr=parcor(Sigma)
> round(PCorr,digits=2)
                east.west north.south angle radial.position velocity
east.west            1.00        0.37 -0.04            0.80     0.14
north.south          0.37        1.00  0.15           -0.03    -0.88
angle               -0.04        0.15  1.00           -0.05     0.16
radial.position      0.80       -0.03 -0.05            1.00     0.30
velocity             0.14       -0.88  0.16            0.30     1.00
> adj=1*(abs(PCorr)>0.1)
> drawGraph(adj)
```

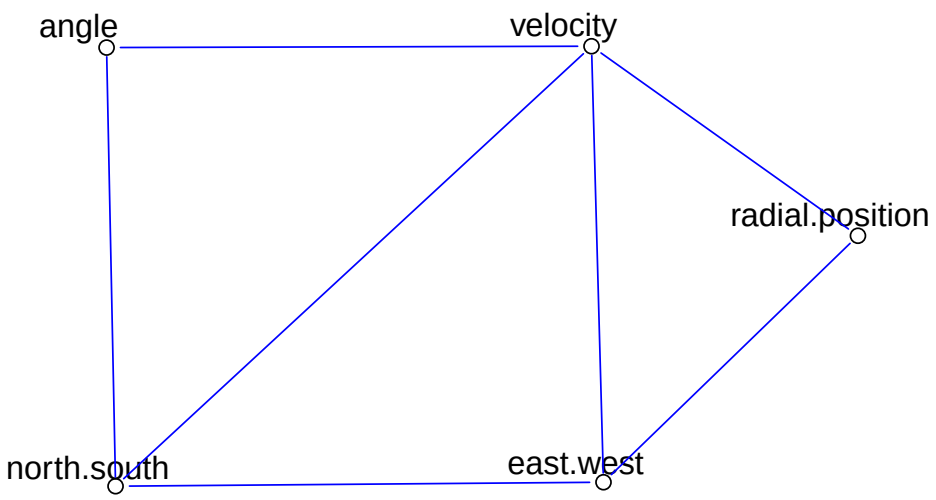


Figure 6.5: Possible conditional independence graph describing the `galaxy` data

6.4.2 Efficient computation of the sample precision matrix

In the case where we are estimating sample variance and precision using data, we are interested in computing

$$\widehat{\mathsf{Q}} = \mathsf{S}^{-1},$$

where S is the sample variance matrix. The obvious way to compute this is to first compute S, somehow, and then invert it to get $\widehat{\mathsf{Q}}$. Indeed, we used this approach in some of the above examples. However, this is a numerically unstable and inefficient approach. There are many better ways to compute $\widehat{\mathsf{Q}}$, but the simplest is to make use of the SVD of the centered data matrix. To understand how this works, first note that

$$\mathsf{S} = \frac{1}{n-1}\mathsf{X}^{\mathsf{T}}\mathsf{H}_n\mathsf{X} = \frac{1}{n-1}\mathsf{W}^{\mathsf{T}}\mathsf{W},$$

where $\mathsf{W} = \mathsf{H}_n\mathsf{X}$ is the centered data matrix. It is then clear that

$$\widehat{\mathsf{Q}} = \mathsf{S}^{-1} = \left(\frac{1}{n-1}\mathsf{W}^{\mathsf{T}}\mathsf{W}\right)^{-1} = (n-1)(\mathsf{W}^{\mathsf{T}}\mathsf{W})^{-1}.$$

Assume now that we have computed the SVD of W as

$$\mathsf{W} = \mathsf{U}\mathsf{D}\mathsf{V}^{\mathsf{T}}.$$

We know that in this case we have $\mathsf{W}^{\mathsf{T}}\mathsf{W} = \mathsf{V}\mathsf{D}^2\mathsf{V}^{\mathsf{T}}$, and hence

$$\widehat{\mathsf{Q}} = (n-1)\mathsf{V}\mathsf{D}^{-2}\mathsf{V}^{\mathsf{T}}.$$

This provides us with an efficient and numerically stable way to evaluate $\widehat{\mathsf{Q}}$, since the only inversion is of a diagonal matrix.

See section 17.3 (p.630) of **[ESL]** for further details of estimating undirected GGMs from data.

6.5 Directed acyclic graph (DAG) models

6.5.1 Introduction

Given a conditional independence statement such as $X \perp\!\!\!\perp Y|Z$, we know that the joint density for the variables X, Y and Z must factorise in a special way, but it does not tell us what exactly the "natural" factorisation is. For example, given the above statement, the following three factorisations are all consistent with the conditional independence statement:

$$f(x,y,z) = f(x)f(z|x)f(y|z)$$
$$f(x,y,z) = f(y)f(z|y)f(x|z)$$
$$f(x,y,z) = f(z)f(x|z)f(y|z).$$

But these factorisations are different, and in the context of conditionally specified statistical models, one of these choices may be much more natural or convenient than another. In many applications, a particular choice of factorisation will be needed, and so it is useful to have a method of encoding a particular factorisation and associated conditional independence statements in a formal way. This turns out to be very convenient using directed acyclic graphs (DAGs).

6.5.2 Directed graphs

A directed graph, or *digraph*, $\mathcal{G}$, is a tuple (V, E) where V is a finite set of vertices and E a set of *directed* edges, with each edge $e \in E$ being an ordered pair of the form $e = (v_i, v_j)$, $v_i, v_j \in V$, $i \neq j$, representing an edge *from* v_i *to* v_j, often written $v_i \to v_j$. Clearly $E \subseteq V \times V = V^2$.

A *directed path* is a sequence $x_0, x_1, \ldots, x_m \in V$ such that $x_{i-1} \to x_i,\ i = 1, 2, \ldots, m$. A *directed cycle* is a (non-trivial) directed path starting and ending at the same vertex.

A *directed acyclic graph* (DAG) is a digraph which does not contain any directed cycles. DAGs turn out to be a very important concept in many areas of discrete mathematics and computing science.

x is a *parent* of y if $x \to y$. The set of all parents of a node y is denoted $\mathrm{pa}(y)$. x is a *child* of y if $y \to x$. The set of all children of a node y is denoted $\mathrm{ch}(y)$.

If there exists a directed path from x to y, then x is said to be an *ancestor* of y, and y is said to be a *descendant* of x. The set of all ancestors of a node y is written $\mathrm{an}(y)$, and the set of all descendants of a node x is written $\mathrm{de}(x)$. The *non-descendants* of x are defined by $\mathrm{nd}(x) \equiv V \backslash \{x \cup \mathrm{de}(x)\}$. Note that for a DAG, we must have $\mathrm{an}(x) \subseteq \mathrm{nd}(x)$ (easy exercise).

A *list ordering* of a DAG is a sequential ordering of the nodes, $x_1, x_2, \ldots, x_p$ in such a way that $x_i \in \mathrm{an}(x_j) \Rightarrow i < j$. A list ordering can always be constructed for a DAG, but they are usually not unique.

Example: DAG

Consider the graph $\mathcal{G} = (V, E)$, where $V = \{X, Y, Z\}$ and $E = \{(X, Z), (Z, Y)\}$. We can plot this in **R** using

Figure 6.6: Simple DAG with 3 nodes and 2 edges

```
drawGraph(DAG(Z ~ X, Y ~ Z))
```

giving the graph shown in Figure 6.6. Note, for example, that Z is a child of X and a parent of Y. Similarly, Y is a descendant of X and X is an ancestor of Y. Also note that here, $\mathrm{an}(Y) = \mathrm{nd}(Y) = \{X, Z\}$. X, Z, Y is a directed path, but Y, Z, X is not. Similarly, X, Z, Y is the only valid list ordering for this DAG.

From a DAG $\mathcal{G}$, it is possible to construct an associated undirected graph, $\mathcal{G}^m$, known as the *moral graph* of $\mathcal{G}$, by joining (marrying) all pairs of parents of each node, and then dropping arrows from all edges. This graph has the property that for each node v and its parents, $\mathrm{pa}(v)$ in the DAG, the corresponding subgraph in the moral graph induced by $v \cup \mathrm{pa}(v)$ is complete. The marrying of parents ensures that this is the case, and is a key property of the moral graph, as we shall see.

Example: DAG

For the simple DAG previously considered, each node has just one parent, so no parents need marrying, and the moral graph is obtained simply by dropping arrows from the DAG.

6.5.3 DAG models

As with undirected graphs, we now wish to associate factorisation and conditional independence properties of random variables with graphs, and as for the undirected case, there are several ways to do this. We consider the two most fundamental here.

Definition 14 *We say that the DAG $\mathcal{G}$ has the* recursive factorisation *or* directed factorisation *(DF) property if the joint density of the random variables factorises in the form*

$$f(x) = \prod_{v \in V} f(v|\,\mathrm{pa}(v)).$$

Models constructed in this way are sometimes referred to as Bayesian networks, due to their occurrence in many applications of Bayesian inference.

Example: DAG

For the DAG we have been considering, the implied factorisation is

$$f(x, y, z) = f(x)f(z|x)f(y|z).$$

Definition 15 *We say that the DAG $\mathcal{G}$ has the* directed local *(DL) Markov property if for all $v \in V$ we have*

$$v \perp\!\!\!\perp \mathrm{nd}(v)|\mathrm{pa}(v).$$

Example: DAG

For the DAG we have been considering, the only non-trivial implied CI statement is

$$Y \perp\!\!\!\perp (X, Z)|Z,$$

or equivalently,

$$Y \perp\!\!\!\perp X|Z.$$

Lemma 1 *(DF) $\Rightarrow$ (DL).*

Proof

We assume (DF), and pick an arbitrary $u \in V$. We wish to show that $u \perp\!\!\!\perp \mathrm{nd}(u)|\mathrm{pa}(u)$. First partition $V = u \cup \mathrm{nd}(u) \cup \mathrm{de}(u)$, and write our factorisation in the form

$$f(x) = f(u|\,\mathrm{pa}(u)) \prod_{v \in \mathrm{nd}(u)} f(v|\,\mathrm{pa}(v)) \prod_{v \in \mathrm{de}(u)} f(v|\,\mathrm{pa}(v)).$$

We are not interested in the descendants of u, so we can marginalise those away to leave

$$f(u \cup \mathrm{nd}(u)) = f(u|\,\mathrm{pa}(u)) \prod_{v \in \mathrm{nd}(u)} f(v|\,\mathrm{pa}(v)).$$

But now the only term involving u is the first, giving

$$f(u|\,\mathrm{nd}(u)) = f(u|\,\mathrm{pa}(u)),$$

or in other words, $u \perp\!\!\!\perp \mathrm{nd}(u)|\mathrm{pa}(u)$. □

Lemma 2 *(DL) ⇒ (DF).*

Proof

Start with a list ordering of the nodes $x_1, x_2, \ldots, x_p$, then factorise the joint density in the form

$$\begin{aligned} f(x) &= f(x_1)f(x_2|x_1)\cdots f(x_p|x_1,\ldots,x_{p-1}) \\ &= \prod_{i=1}^{p} f(x_i|x_1,\ldots,x_{i-1}). \end{aligned}$$

Considering the ith term in this product, we know that since we have a list ordering we must have $\{x_1,\ldots,x_{i-1}\} \subseteq \mathrm{nd}(x_i)$, and since we are assuming that $x_i \perp\!\!\!\perp \mathrm{nd}(x_i)|\mathrm{pa}(x_i)$, we have that $x_i \perp\!\!\!\perp x_1,\ldots,x_{i-1}|\mathrm{pa}(x_i)$, and so $f(x_i|x_1,\ldots,x_{i-1}) = f(x_i|\,\mathrm{pa}(x_i))$, giving the factorisation

$$f(x) = \prod_{i=1}^{p} f(x_i|\,\mathrm{pa}(x_i)),$$

and so (DF) is satisfied. □

Therefore the two properties we have considered are equivalent.

Corollary 2 *(DF) ⇔ (DL).*

We now link DAG models back to undirected graphical models via the moral graph.

Proposition 49 *If the DAG $\mathcal{G}$ satisfies (DF), then the associated moral graph $\mathcal{G}^m$ satisfies (F).*

Proof

We start with the factorisation

$$\begin{aligned} f(x) &= \prod_{i=1}^{p} f(x_i|\,\mathrm{pa}(x_i)) \\ &= \prod_{i=1}^{p} h_i(x_i, \mathrm{pa}(x_i)), \end{aligned}$$

but by construction of the moral graph, $x_i \cup \mathrm{pa}(x_i)$ is complete in $\mathcal{G}^m$, and hence will be contained in a clique of $\mathcal{G}^m$. By allocating each term to a clique, we see that the density factorises according to the clique structure of $\mathcal{G}^m$, and (F) is satisfied. □

This is a key result, since using this, and the results we already have, we can immediately conclude that

Corollary 3 *The moral graph $\mathcal{G}^m$ associated with a DAG $\mathcal{G}$ satisfying (DL) or (DF) satisfies the Markov properties (F), (G), (L) and (P).*

Note that none of the above discussion relies on the positivity assumption necessary for the Hammersley-Clifford theorem. In summary, one way to understand the conditional independence assumptions associated with a DAG is to form the associated moral graph, and then read off conditional assumptions from the moral graph according to any of the interpretations (F), (G), (L) or (P). Note that this process loses information, in that not all of the (conditional) independence statements associated with the original DAG model are present in the corresponding moral graph, but we will not pursue this issue in detail here.

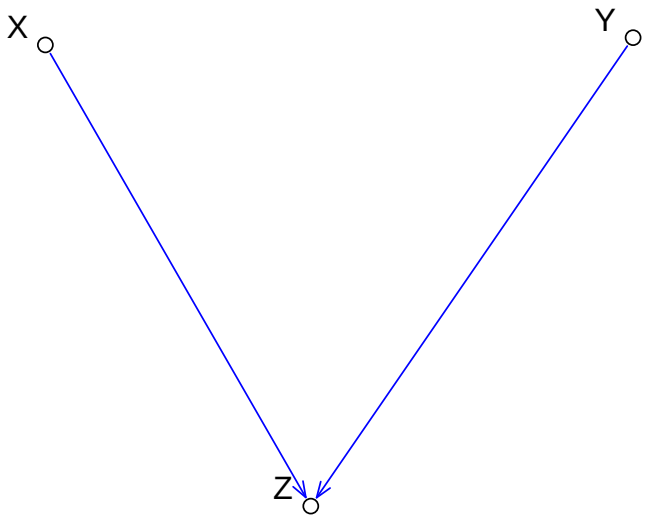


Figure 6.7: DAG for a "collider"

Example: DAG

For our simple example, we have already seen that the moral graph is just the undirected version of the DAG, encoding the CI statement $X \perp\!\!\!\perp Y|Z$, as we have already deduced. Note however, that the three different factorisations consistent with this statement, considered at the start of this section, all encapsulate the same CI statement and all have the same moral graph. This is an important point to understand — there will typically be many DAGs consistent with a particular moral graph, and with a particular set of conditional independence statements.

Example: collider

At the beginning of this section we considered three different factorisations corresponding to the conditional independence statement $X \perp\!\!\!\perp Y|Z$. Each of these factorisations corresponds to a graph with 3 nodes and 2 directed edges, with Z in the middle. There is a graph with two edges with Z in the middle we have not considered, known as a *collider*. It is the graph with edges $X \to Z$ and $Y \to Z$. We can draw it using **R** with

```
drawGraph(DAG(Z ~ X+Y))
```

giving the plot shown in Figure 6.7.

Note that this DAG corresponds to the factorisation

$$f(x, y, z) = f(x)f(y)f(z|x, y).$$

There is no (non-trivial) CI statement associated with this DAG. In particular, since X and Y are both parents of Z, they get "married" in forming the moral graph, and so the moral graph for this DAG is complete. However, it should be noted that this factorisation and the

corresponding DAG do encode the *marginal* independence of X and Y. So here X and Y are marginally independent, but not conditionally independent given Z. It is not possible to encode this information in an undirected graphical model. It is therefore possible to encode information in DAG models that is not possible to encode in undirected graphs.

Example

Draw the DAG for the random variables X_1, X_2, X_3 and X_4 given the following factorisation of the joint density,

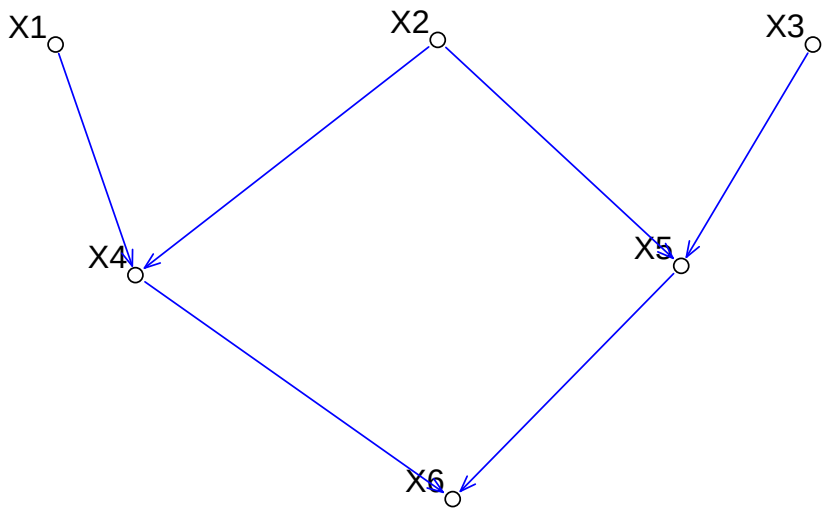
$$f(x_1, x_2, x_3, x_4) = f(x_1)f(x_2)f(x_3|x_1, x_2)f(x_4|x_3).$$

Draw the associated moral graph. Is it true that $X_1 \perp\!\!\!\perp X_2|X_3$? Is it true that $X_2 \perp\!\!\!\perp X_4|X_3$? Can draw DAG with `drawGraph(DAG(x3~x1+x2,x4~x3))`. Can draw moral graph with `drawGraph(UG(~x1*x2*x3+x3*x4))`.

It is *not* true that $X_1 \perp\!\!\!\perp X_2|X_3$, since X_1 and X_2 are married in the moral graph. It is true that $X_2 \perp\!\!\!\perp X_4|X_3$, since X_3 separates X_2 and X_4 in the moral graph.

Example

Write down the factorisation of the full joint density implied by the following DAG:



$$f(x) = f(x_1)f(x_2)f(x_3)f(x_4|x_1, x_2)f(x_5|x_2, x_3)f(x_6|x_4, x_5)$$

6.5.4 Fitting to data

Given a particular DAG structure, we know the form of the factorisation of the joint density, and a variety of methods may be used to estimate the individual factors. In the graphical Gaussian case, this essentially involves linearly regressing each variable on its parents. See the function `fitDag()` in the `ggm` package.

Estimating DAG structure from data is an altogether more delicate matter, which we will not consider in detail here. Note, however, that in general it is not possible to infer directionality from observational data. Consider first the bivariate case for variables X and Y. We know that we can factor the joint density either as $f(x, y) = f(x)f(y|x)$ or $f(x, y) = f(y)f(x|y)$. The first of these corresponds to $X \rightarrow Y$ and the second to $X \leftarrow Y$. We cannot distinguish these without some additional information. If we have a particular parametric form for the terms in the factorisation, that could in principle help, but in the graphical Gaussian case we assume that the variables are bivariate normal, and the two factorisations are indistinguishable from data. In general, the best strategy is often to estimate an undirected graph structure from data, and then explore the set of directed graphs consistent with this undirected graph, informally.

6.6 Conclusion

Graphical models are one of the most important tools in modern multivariate statistical modelling and analysis. In particular, they are central to Bayesian hierarchical modelling, and many modern Bayesian computational methods. We do not have time to explore

these ideas here, but the basic properties of undirected and directed graphs, and graphical Gaussian models forms a foundation for further study in this area.

Note that the standard reference on the theory of graphical models is:

Lauritzen, S. L. (1996) Graphical Models, Oxford Science Publications.

Some of the material in this chapter was derived from the above text.

Chapter 7

Variable selection and multiple testing

7.1 Regularisation and variable selection

7.1.1 Introduction

Let us reconsider the problem of multiple linear regression, in the form

$$\boldsymbol{y} = \mathsf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where X is an $n \times p$ data matrix, and we choose $\boldsymbol{\beta}$ in order to minimise $\|\boldsymbol{\varepsilon}\|^2$. If p is large, so that X contains many variables, there may be many choices of $\boldsymbol{\beta}$ that *almost* minimise our loss function

$$L(\boldsymbol{\beta}) = (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})^{\mathsf{T}}(\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta}).$$

We know that the $\boldsymbol{\beta}$ minimising $L(\boldsymbol{\beta})$ is given by the solution of the normal equations

$$\mathsf{X}^{\mathsf{T}}\mathsf{X}\hat{\boldsymbol{\beta}} = \mathsf{X}^{\mathsf{T}}\boldsymbol{y}.$$

When $n \geq p$ and X has full column rank, there is a unique solution given by

$$\hat{\boldsymbol{\beta}} = (\mathsf{X}^{\mathsf{T}}\mathsf{X})^{-1}\mathsf{X}^{\mathsf{T}}\boldsymbol{y}.$$

However, if X does not have full column rank (for example, because some of the variables are co-linear), or $p > n$, then $\mathsf{X}^{\mathsf{T}}\mathsf{X}$ will not be invertible, and there will be many solutions to the normal equations, corresponding to many different minimisers of the loss function $L(\boldsymbol{\beta})$. Even if X is full rank, if it is close to rank degenerate, then solution of the normal equations may be numerically unstable, and the "optimal" $\boldsymbol{\beta}$ may turn out to have poor predictive properties.

Regularisation and variable selection are two ways to deal with the above problem, which turn out to be closely related. Regularisation is concerned with smoothing the loss function, ensuring it has a unique minimum, and "shrinking" the solution towards a sensible default, often zero. Regularisation methods are often termed shrinkage methods for this reason. We begin by looking at the classical method of regularising regression problems, often known as *ridge regression*, or Tikhonov regularisation.

7.1.2 Ridge regression

There are several ways to think about ridge regression. We start by viewing it as a *constrained* minimisation problem, where we now find a minimum of $L(\boldsymbol{\beta})$ subject to the constraint that $\|\boldsymbol{\beta}\|$ does not exceed some threshold size. This way we ensure that we do not consider unrealistic choices of $\boldsymbol{\beta}$. Now, we often tackle constrained optimisation problems using a Lagrange multiplier approach. In this case we will minimise the loss function

$$L^r(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|^2,$$

and then later use λ in order to impose the required constraint. However, this gives us a different, and perhaps more natural, way to think about ridge regression. We modify our loss function in order to include a penalty term which penalises choices of $\boldsymbol{\beta}$ with large norm. Small values of λ lead to small penalties, and hence solutions similar to those that would be obtained through regular least squares approaches. However, larger values of λ will lead to strong penalties for $\boldsymbol{\beta}$ with large norm, and hence will have the effect of "shrinking" the optimal solution towards the origin. Let us now consider how to solve for the optimal $\boldsymbol{\beta}$ for given fixed $\lambda \geq 0$.

Proposition 50 *The $\boldsymbol{\beta}$ minimising $L^r(\boldsymbol{\beta})$ is given by the solution to the equations*

$$(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})\boldsymbol{\beta} = \mathsf{X}^\mathsf{T}\boldsymbol{y}.$$

For $\lambda > 0$, these equations are guaranteed to have a unique solution given by

$$\boldsymbol{\beta} = (\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y}.$$

Proof

First let us expand and simplify the loss function

$$\begin{aligned} L^r(\boldsymbol{\beta}) &= L(\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|^2 \\ &= (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})^\mathsf{T}(\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^\mathsf{T}\boldsymbol{\beta} \\ &= \boldsymbol{y}^\mathsf{T}\boldsymbol{y} + \boldsymbol{\beta}^\mathsf{T}\mathsf{X}^\mathsf{T}\mathsf{X}\boldsymbol{\beta} - 2\boldsymbol{\beta}^\mathsf{T}\mathsf{X}^\mathsf{T}\boldsymbol{y} + \lambda\boldsymbol{\beta}^\mathsf{T}\boldsymbol{\beta} \\ &= \boldsymbol{y}^\mathsf{T}\boldsymbol{y} + \boldsymbol{\beta}^\mathsf{T}(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})\boldsymbol{\beta} - 2\boldsymbol{\beta}^\mathsf{T}\mathsf{X}^\mathsf{T}\boldsymbol{y}. \end{aligned}$$

Now differentiating wrt $\boldsymbol{\beta}$ and equating to zero gives

$$(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})\boldsymbol{\beta} = \mathsf{X}^\mathsf{T}\boldsymbol{y}.$$

For $\lambda > 0$, the matrix $\lambda\,\mathrm{I}$ is strictly positive definite, and since $\mathsf{X}^\mathsf{T}\mathsf{X}$ is non-negative definite, $\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I}$ is strictly positive definite, and hence is invertible. This gives the final result. □

The attraction of ridge regression is that irrespective of the column rank of X, or the relative sizes of n and p, for any $\lambda > 0$ the ridge regression has a unique solution, and for large values of λ, the numerical solution is computationally very stable. It is in this sense that the problem has been regularised.

Example: galaxy data

We can look at how to implement ridge regression using R for the `galaxy` data. We being by thinking about regressing `velocity` on the other 4 variables, and start using `lm()`.

```
> require(ElemStatLearn)
> y=galaxy[,5]
> X=as.matrix(galaxy[,-5])
> lm(y~X)

Call:
lm(formula = y ~ X)

Coefficients:
     (Intercept)        Xeast.west      Xnorth.south             Xangle
       1589.4229            0.7741           -3.1918            0.1245
Xradial.position
          0.9012
```

As usual, we can append a column of 1s to the front of `X` and then compute the least squares solution either using `lm()` or by solving the normal equations directly.

```
> X0=cbind(rep(1,length(y)),X)
> lm(y~0+X0)

Call:
lm(formula = y ~ 0 + X0)

Coefficients:
               X0         X0east.west      X0north.south
                  X0angle
        1589.4229             0.7741            -3.1918
           0.1245
X0radial.position
           0.9012

> solve(t(X0)%*%X0,t(X0)%*%y)
                         [,1]
                1589.4229455
east.west          0.7740997
north.south       -3.1917877
angle              0.1245414
radial.position    0.9011797
> QR=qr(X0)
> solve(qr.R(QR),t(qr.Q(QR))%*%y)
                         [,1]
                1589.4229455
east.west          0.7740997
north.south       -3.1917877
angle              0.1245414
radial.position    0.9011797
```

We can avoid the complications associated with intercepts by first centering the output and the predictor matrix.

```
> y=y-mean(y)
> W=sweep(X,2,colMeans(X))
> solve(t(W)%*%W,t(W)%*%y)
                      [,1]
east.west         0.7740997
north.south      -3.1917877
angle             0.1245414
radial.position   0.9011797
>
> QR=qr(W)
> solve(qr.R(QR),t(qr.Q(QR))%*%y)
                      [,1]
east.west         0.7740997
north.south      -3.1917877
angle             0.1245414
radial.position   0.9011797
```

We can now carry out ridge regression (with $\lambda = 100$) by direct solution of the regularised normal equations

```
> solve(t(W)%*%W+100*diag(4),t(W)%*%y)
                      [,1]
east.west         0.7646416
north.south      -3.1881190
angle             0.1244684
radial.position   0.9059122
```

We see that the predictor is very similar to the usual least squares solution, but the addition of a diagonal term will ensure that the numerical solution of the equations will be very stable. Note that we could also compute the optimal predictor by numerically optimising the loss function.

```
> loss<-function(beta)
+ {
+   eps=y-W%*%beta
+   return(sum(eps*eps)+lambda1*sum(abs(beta))+lambda2*sum(beta*beta))
+ }
> lambda1=0
> lambda2=0
> optim(rep(0,4),loss,control=list(maxit=10000,reltol=1e-12))
$par
[1]  0.7740986 -3.1917888  0.1245412  0.9011769

$value
[1] 288650.8

$counts
function gradient
     537       NA
```

```
$convergence
[1] 0

$message
NULL
```

Note that in the case `lambda1=0`, the loss is exactly as we require for ridge regression. The other term in this loss will be explained in the following section. Note that direct numerical optimisation of multivariate functions is fraught with difficulty, and so whenever a direct alternative exists, it is almost always to be preferred. In this case, direct solution of the regularised normal equations is a much better way to compute the solution.

7.1.3 The LASSO and variable selection

Ridge regression is one way to regularise an ill-conditioned least squares problem, but by no means the only one. In the case of p being large, another obvious approach would be to select just a subset of the available variables. The intuition here is that if there are many variables, it is likely that not all will be useful for predicting $\boldsymbol{y}$, and that since variables are likely to be correlated anyway, it will not be necessary to include all variables in the regression even if all happen to be marginally predictive, since the column span of the subset of variables will be similar to that of a larger subset if the variables dropped are highly correlated with some of those that remain. There are many possible approaches to variable selection, including forwards and backwards selection. Perhaps initially variable selection may appear to have little to do with penalised regularisation methods, but this is not the case, and the connection between them becomes clear in the context of a regularisation method known as the LASSO (least absolute shrinkage and selection operator), which is actually rather closely related to the method of ridge regression we have already considered.

The Lasso uses the penalty

$$L^l(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_1,$$

where

$$\|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{p} |\beta_i|$$

is the l_1-norm of $\boldsymbol{\beta}$. The switch from 2-norm to 1-norm seems at first to be unlikely to make a significant difference to the shrinkage behaviour of the estimator, but in fact it does, and importantly, encourages "sparsity" in the optimal solution. It also complicates analysis somewhat, since it is no longer possible to compute a simple closed-form solution for the optimal $\boldsymbol{\beta}$, and since the l_1-norm is not everywhere differentiable (it is not differentiable on the coordinate axes), methods of optimisation which assume smooth functions cannot be used. In fact, it is precisely this lack of differentiability of the l_1-norm on the axes which causes the method to often have minima including components of $\boldsymbol{\beta}$ which are exactly zero. Zeroes in the optimal $\boldsymbol{\beta}$ correspond to variables which are dropped out of the regression, and similarly, the non-zero elements of $\boldsymbol{\beta}$ correspond to the selected variables. Thus, the simple switch from an l_2 to an l_1 regularisation penalty leads directly to a method which simultaneously regularises and selects variables for analysis.

Clearly for $\lambda = 0$ and for small values of $\lambda > 0$, the method will behave very like ordinary least squares regression. However, as the value of $\lambda > 0$ increases, the effect of the l_1 penalty takes effect, and variables of weak predictive effect begin to drop out of the optimal predictor.

Example: galaxy data

We can now think about using the Lasso in order to compute an optimal predictor for the galaxy data. We can begin by trying to directly optimise the loss as follows.

```
> lambda1=100000
> lambda2=0
> optim(rep(0,4),loss,control=list(maxit=10000,reltol=1e-12))
$par
[1] -1.784425e-11 -2.837611e+00  7.050503e-03  1.103656e+00

$value
[1] 713727.1

$counts
function gradient
     469       NA

$convergence
[1] 0

$message
NULL
```

Note that for `lambda2=0`, the loss is exactly what we need for the Lasso. Note that the first coefficient is very close to zero, so the first variable has dropped out of the regression. As already explained, direct numerical optimisation is problematic, especially for non-smooth objectives such as this. Fortunately there is an **R** package called `elasticnet` which has efficient procedures for optimising loss functions of this form. We can use it to solve this problem as follows.

```
> require(elasticnet)
> predict(enet(X,y,lambda=lambda2,normalize=FALSE),s=lambda1,mode="
   penalty",type="coefficients")$coefficients
      east.west     north.south           angle radial.position
    0.000000000    -2.836045617     0.007405521     1.104725175
```

Note how the value for the ridge loss term (here, 0) is passed into the call to `enet()`, but the coefficient of the l_1 penalty is only needed for the call to the generic function **`predict`**`()`. This is due to the way that the algorithm is implemented, in that solutions for all values of the l_1 penalty are computed simultaneously. We can exploit this in order to understand the effect of varying the l_1 penalty over a range of values. The command

```
> plot(enet(X,y,lambda=lambda2,normalize=FALSE))
```

gives the plot in Figure 7.1. The left hand side of this plot shows the results of using a very large shrinkage parameter which shrinks all of the regression coefficients to zero.

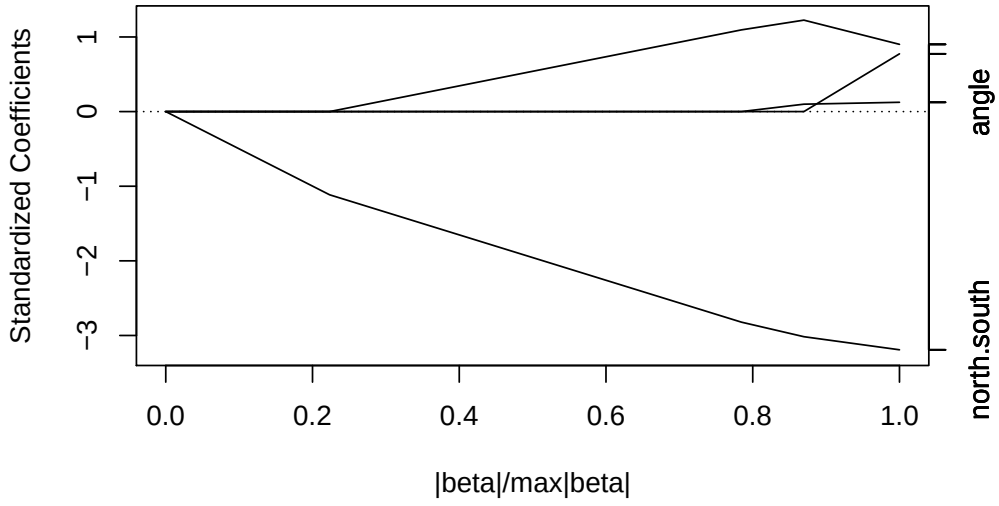


Figure 7.1: A graphical illustration of the effect of varying the LASSO shrinkage parameter on the coefficients of the optimal predictor.

The RHS shows the effect of using a zero shrinkage parameter, leading to the usual coefficients for ordinary least squares. Moving from left to right, we see additional variables being incorporated into the optimal predictor as the l_1 penalty is gradually relaxed.

7.1.4 The elastic net

The lasso turns out to be good for variable selection, but less good than ridge regression for regularisation and shrinkage, and also has limitations in the $p > n$ scenario. It is therefore natural to consider combining the two different penalties, and this is the idea behind the elastic net. So the elastic net uses the loss function

$$L^e(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \lambda_2 \|\boldsymbol{\beta}\|^2 + \lambda_1 \|\boldsymbol{\beta}\|_1.$$

Clearly in the special case $\lambda_1 = 0$ we get ridge regression, and in the case $\lambda_2 = 0$ we get the lasso. So the elastic net is a generalisation of both ridge regression and the lasso, and combines the best aspects of both.

Example: galaxy data

Again, we could attempt to optimise this loss function directly, as follows.

```
> lambda1=100000
> lambda2=100
> optim(rep(0,4),loss,control=list(maxit=10000,reltol=1e-12))$par
[1]  1.211979e-11 -2.834280e+00  7.376000e-03  1.104901e+00
```

We can also use the `enet()` function to compute this for us.

```
> predict(enet(X,y,lambda=lambda2,normalize=FALSE),s=lambda1,mode="
   penalty",type="coefficients")$coefficients/(1+lambda2)
      east.west     north.south           angle radial.position
       0.000000       -2.834275        0.007378        1.104903
```

Note the division by $(1 + \lambda_2)$ at the end. If we do not impose this correction, we get

```
> predict(enet(X,y,lambda=lambda2,normalize=FALSE),s=lambda1,mode="
   penalty",type="coefficients")$coefficients
      east.west     north.south           angle radial.position
       0.000000     -286.261800        0.745178      111.595172
```

These coefficients are a correction to those of the *naive* elastic net we have presented, which are thought to have better predictive performance in some scenarios.

7.1.5 $p >> n$

In some settings we have many more variables than observations, $p >> n$. We have seen that in this case there is not a unique solution to the ordinary least squares problem. In this case regularisation is used in order to make the problem well-defined. We have seen that ridge regression, or Tikhonov regularisation, is very effective at making the problem solvable, and shrinking the solution towards the origin. Some kind of shrinkage is vital in the $p >> n$ scenario, and loss functions containing an l_2 penalty are very often used for this purpose. Variable selection is very often desirable in high dimensional scenarios, but the Lasso does not perform especially well for $p >> n$ due to the lack of l_2 regularisation. In this case, the elastic net, which combines l_1 and l_2 regularisation, performs much better, allowing for a combination of shrinkage and variable selection which can lead to sparse predictors with good performance.

Example: microarray data

We can investigate the $p >> n$ case using the `nci` microarray data. Suppose that we want to regress the first gene on the rest. We can attempt to do ordinary least squares as follows.

```
y=t(nci)[,1]
X=t(nci)[,-1]
lm(y~X)$coefficients
```

but this fails. Trying to compute the solution directly doesn't help.

```
X0=cbind(rep(1,length(y)),X)
solve(t(X0)%*%X0,t(X0)%*%y)
```

We get an error, due to the fact that the system we are attempting to solve is singular. However, if we include an l_2 penalty, we can easily solve the ridge regression problem as follows.

```
solve(t(X0)%*%X0+100*diag(ncol(X0)),t(X0)%*%y)
```

This returns a vector of 6,830 regression coefficients, which we can use for prediction. However, this representation is not sparse, and so it makes sense to use some kind of variable selection method to obtain a more parsimonious predictor. We can use the elastic net for this purpose.

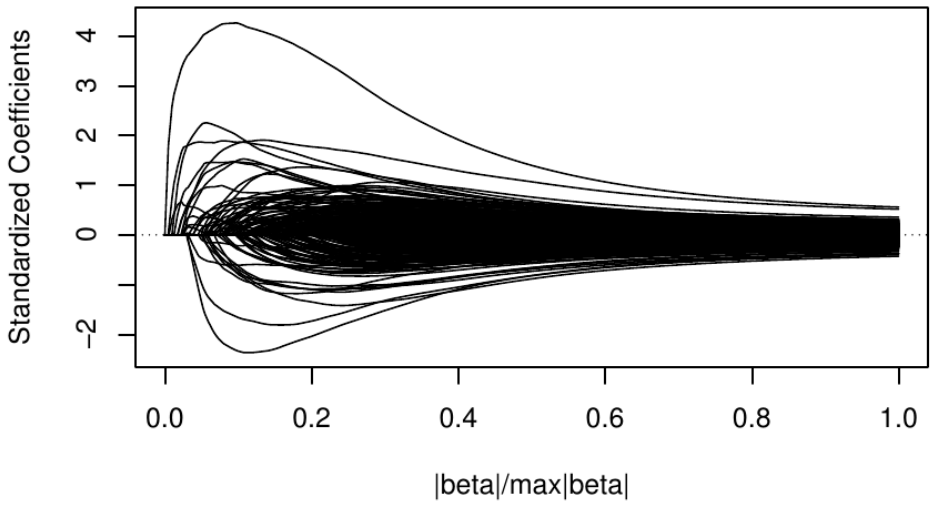


Figure 7.2: A graphical illustration of the effect of varying the l_1 shrinkage parameter on the coefficients of the optimal predictor for the `nci` microarray data.

```
lambda1=10
lambda2=100
nci.enet=enet(X,y,lambda=lambda2,normalize=FALSE)
predict(nci.enet,s=lambda1,mode="penalty",type="coefficients")$
   coefficients/(1+lambda2)
plot(nci.enet)
```

This leads to the plot shown in Figure 7.2. We see many predictors being brought in as the l_1 penalty is relaxed. In practice, we will most likely decide on a sensible number of predictors, and then choose λ_1 appropriately. Alternatively, we could keep back some test data and choose the parameters via cross-validation.

See section 3.3 (p.57) and 3.4 (p.61) of **[ESL]** for further details of regularisation and variable selection and Chapter 18 (p.649) of **[ESL]** for further details of methods for $p >> n$.

7.2 Multiple testing

7.2.1 Introduction

For this section, we will think in terms of an $n \times m$ data matrix X, rather than $n \times p$, as in this final section, p will be used to refer to p-values, in the usual statistical sense, and so it will help to avoid confusion if we let m denote the number of variables of interest. In the previous section we considered the problem of variable selection. Some methods of variable selection involve statistical testing of the null hypothesis that adding in the variable does not improve prediction. Indeed, there are many other contexts where it is natural to

apply a test to each variable separately in a multivariate problem. This is particularly common in genomics, where statistical tests are often applied to each variable in turn, in order to find genes which behave significantly differently in some appropriate sense. In this context, we often end up with a p-value associated with each variable (each gene in the genomic data context), and need to somehow threshold these p-values in order to produce a list of significant variables. The p-value is the probability (under repeated sampling) that a test statistic as extreme as that observed, will be observed, assuming that the null hypothesis is true.

7.2.2 The multiple testing problem

We typically consider some threshold significance level, α (often $\alpha = 0.05$ for a single test), and then flag as "significant" tests with a p-value smaller than α. There are many issues with this, but here we will focus on the problems which arise when many tests are considered together (here, m). In order to get a simple understanding of the problem, we will assume that all of the tests are independent, although in practice this is unlikely to be the case.

If we carry out m tests, even if the null hypothesis is true in every case, so that we would not wish any tests to be flagged as significant, we know that by construction, each of the m tests will be significant with probability α. In this case the number of false positives will be X, where $X \sim Bin(m, \alpha)$. Then we have $\mathrm{E}(X) = m\alpha$, so for large m we expect to get many false positive tests. For example, for $m = 7,000$ and $\alpha = 0.05$, we would expect to get 350 false positives in the case where there are no true positives. Indeed, if we anticipate the number of true positives to be small, will will *a priori* expect that most of the positive tests will in fact be false positives, and we will have no way to decide on which or how many of these we should consider. This is the multiple testing problem.

7.2.3 Bonferroni correction

The Bonferroni method is the classical solution to multiple testing. For a collection of (independent) tests, Bonferroni's method controls the family-wise error rate (FWER) — this is the probability of at least one false positive when there are no true positives. Therefore

$$\begin{aligned} \text{FWER} &= \mathrm{P}(\text{At least one false positive}) \\ &= 1 - \mathrm{P}(\text{No false positives}) \\ &= 1 - (1-\alpha)^m \end{aligned}$$

Now, for small α, $(1-\alpha)^m \simeq 1 - m\alpha$ (first 2 terms in binomial expansion), and so

$$\text{FWER} \simeq m\alpha.$$

Note that although we have derived this as an approximation, and assuming independent tests, it is possible to deduce directly using Boole's inequality that

$$\text{FWER} \leq m\alpha,$$

without assuming independence or making any approximation. Either way, if a FWER of α' is required, choosing a significance level for the individual tests of $\alpha = \alpha'/m$ will achieve

this. For example, for $m = 7,000$, if a FWER of $\alpha' = 0.1$ is required, then a significance level of around $\alpha = 0.000014$ is required. This incredibly stringent significance level is required in order to control the FWER, but will clearly come at the expense of many more false negatives. For large m, this leads to a procedure with poor power for identifying the true positives of primary interest.

7.2.4 False discovery rate (FDR)

In the context of a large number of tests, m, as is typically the case in "discovery science" such as genomics, Bonferroni's method is considered to be too conservative. This is because in that context, the FWER is really not of primary concern. Here it makes more sense to consider the false discovery rate (FDR) — this is the proportion of false positives among the collection of all rejected null hypotheses. For example, an FDR of 0.1 means that 10% of the tests flagged as significant are false. The converse of this is that 90% of the tests flagged as significant are true, and that will be fine in most discovery contexts — people are generally willing to tolerate a small proportion of false positives, as long as the majority of positives are true. So, we need to understand how to choose a significance threshold in order to give a desired FDR.

Suppose that we have m tests, with p-values $p_1, p_2, \ldots, p_m$. It will be helpful later to also have the p-values ordered in ascending order, $p_{(1)}, p_{(2)}, \ldots, p_{(m)}$, so that $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(m)}$, and $p_{(1)}$ is the most significant test, and $p_{(m)}$ is the least significant test. The choice of significance level will then be equivalent to deciding how many p-values to take, reading down the ordered list. Choosing a significance level α will lead to a particular cutoff, which we will denote $l(\alpha)$, giving a list of significant p-values, $p_{(1)}, p_{(2)}, \ldots, p_{(l(\alpha))}$. Clearly, decreasing α will shorten this list, and increasing α will lengthen the list. That is, the function $l(\alpha)$ is monotonic increasing in α.

Assume now that the number of true positives is small, then the number of false positives is X, where $X \sim Bin(m, \alpha)$, approximately. In that case the FDR is given by

$$\text{FDR} = \frac{X}{l(\alpha)}.$$

But then

$$\mathrm{E}(\text{FDR}) = \frac{\mathrm{E}(X)}{l(\alpha)} = \frac{m\alpha}{l(\alpha)}.$$

If we want to have $\mathrm{E}(\text{FDR}) < \alpha'$, we need to have

$$\frac{m\alpha}{l(\alpha)} < \alpha' \Rightarrow \alpha < \frac{\alpha' l(\alpha)}{m}.$$

That is, we need to choose α sufficiently small that this inequality is satisfied (but otherwise as large as possible). However, we need to be a little bit careful, since both sides of this inequality depend on α.

In order to see how to solve this, it is helpful to invert the relationship between l and α, where we now regard l as given, and consider α to be the value of α giving rise to a list of length l. Then we have

$$\alpha(l) < \frac{\alpha' l}{m},$$

but $p_{(l)}$ is the p-value giving rise to a list of length l, so

$$p_{(l)} < \frac{\alpha' l}{m}$$

is the inequality of interest. But then we can visualise the solution of this problem by plotting $p_{(l)}$ and $\frac{\alpha' l}{m}$ as functions of l, and take the crossing point as determining l and $p_{(l)}$, the threshold significance level.

Example: microarray data

Suppose that for the `nci` microarray data, we want to find genes specifically associated with melanoma. We could, for each gene, do a 2 sample t-test to see if the mean across the melanoma samples is significantly different to the mean across the other samples. On the basis of the computed t statistics, we can compute a p-value for each gene. We can do this in **R** as follows, though the details are not especially important for this course.

```
nci.mel=nci[,colnames(nci)=="MELANOMA"]
nci.rest=nci[,colnames(nci)!="MELANOMA"]
mel.var=apply(nci.mel,1,var)
rest.var=apply(nci.rest,1,var)
pool.var=((8-1)*mel.var+(56-1)*rest.var)/(8+56-2)
pool.se=sqrt(pool.var)*sqrt(1/8+1/56)
tstat=(apply(nci.mel,1,mean)-apply(nci.rest,1,mean))/pool.se
pval=2*(1-pt(abs(tstat),8+7-2))
```

What we end up with is a vector `pval` of length 6,830, containing a p-value for each gene. Under the assumption that the null hypothesis is true for every gene, we would expect $6,830 \times 0.05 = 341.5$ false positives. The command `length(pval[pval<0.05])` returns 1,377, and so it appears that there are likely to be many (around 1,000) true positives among the 1,377 genes that are significant at the 5% level. We could employ a Bonferroni correction, but the command `pval[pval<0.05/6830]` returns just six p-values meeting this very stringent criteria. The FDR approach is a compromise between taking just the six smallest p-values, which are very likely to be all true positives, but leads to vast numbers of false negatives, and the 1,377 p-values below the 5% threshold, which we know will contain large numbers of false positives.

We can sort the p-values and plot them as follows, leading to the plot shown in Figure 7.3.

```
pval.sort=sort(pval)
plot(1:6830,pval.sort,type="l",col=2)
abline(0.05,0,col=3)
abline(0.05/6830,0,col=5)
abline(0,0.05/6830,col=4)
```

The red line shows the ordered p-values. The green line represents the usual 0.05 cut-off, crossed by the p-values at 1,377. The cyan line represents the Bonferroni-corrected threshold, which just looks like it is at zero on the scale of this plot. The dark blue line is the FDR threshold. From this plot, all we can really tell is that the p-values cross this threshold sometime before 500. We can zoom in on the first 500 p-values as follows, giving the plot shown in Figure 7.4.

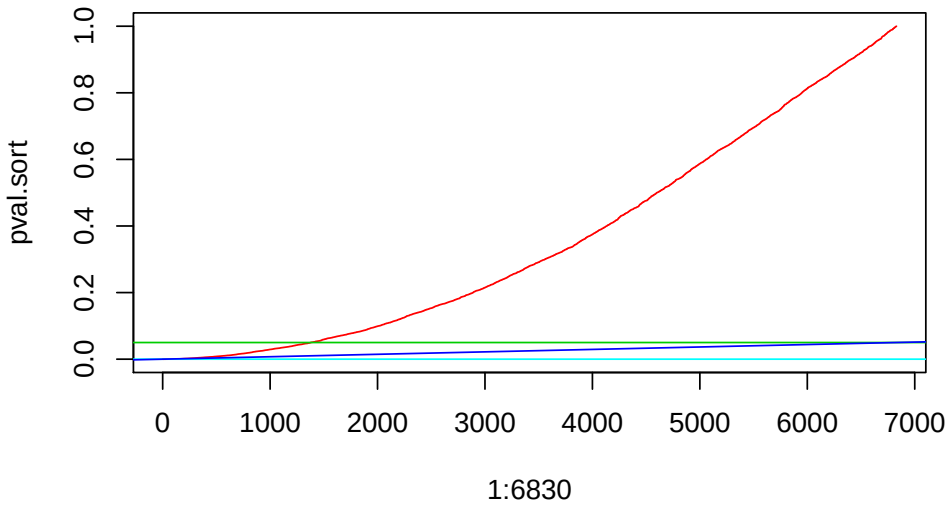


Figure 7.3: Ordered p-values for the `nci` microarray data.

```
plot(1:500,pval.sort[1:500],type="l",col=2)
abline(0.05/6830,0,col=5)
abline(0,0.05/6830,col=4)
```

We still can't see exactly where the p-values cross the Bonferroni threshold, but we know see that the p-values cross the FDR threshold at around 180 (in fact, it first exceeds at 187), and so we will choose to look at the smallest 186 p-values (corresponding to a significance threshold of around 0.0014), if we are only prepared to tolerate a FDR of 5%.

An alternative way to view the solution to the problem, which is also informative, is to rewrite the inequality as

$$\frac{p_{(l)}m}{l} < \alpha'.$$

Then defining

$$f_{(l)} = \frac{p_{(l)}m}{l},$$

we want to find the largest l such that

$$f_{(l)} < \alpha'.$$

So if we plot $f_{(l)}$ against l, we look for the (last) crossing of the α' threshold, from below. Consequently, we can think informally of $f_{(l)}$ as representing the expected FDR associated with the lth ordered p-value. This is closely related to (but not quite the same as) the concept of a q-value, which is also a kind of FDR-corrected p-value. Further examination of such concepts is beyond the scope of this course.

Example: microarray data

We can plot $f_{(l)}$ for 500 most significant microarray genes using the following commands, leading to the plot shown in Figure 7.5.

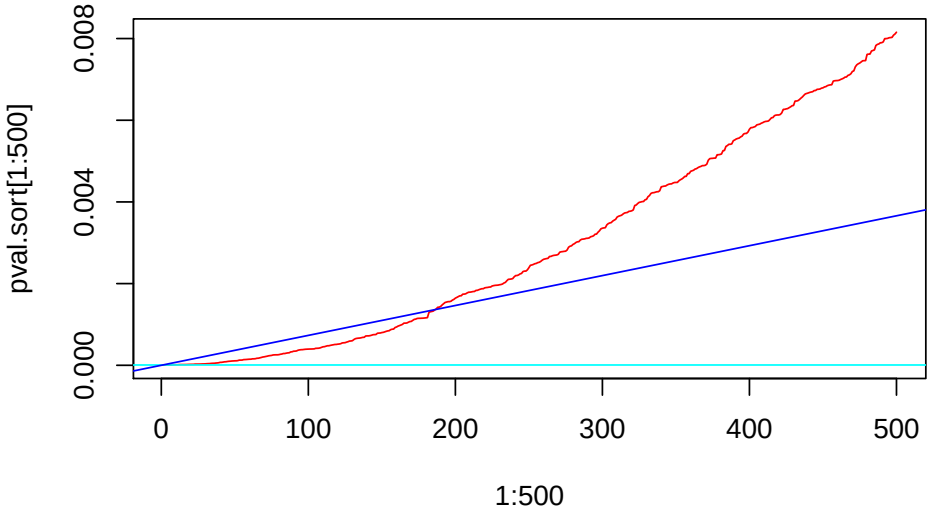


Figure 7.4: First 500 ordered p-values for the `nci` microarray data.

```
fdr=6830*pval.sort/1:6830
plot(1:500,fdr[1:500],type="l",col=2)
abline(0.05,0,col=3)
```

Notice that this function is not monotonic in l, and this why it is not quite right to interpret $f_{(l)}$ as an FDR-corrected p-value, but it is close enough for our purposes.

Before leaving this example, it is worth emphasising that when working with FDR, people often work with thresholds above the 0.05 often used in classical statistical testing. A threshold of 0.1 is very often used (tolerating 1 in 10 false positives), and thresholds of 0.15 are also used sometimes. We can see from Figure 7.5 that if we were to increase our FDR threshold to 0.1, we would get a list containing around 400 genes, and most scientists would consider that to be a more appropriate compromise.

See section 18.7 (p.683) of **[ESL]** for further details of multiple testing problems.

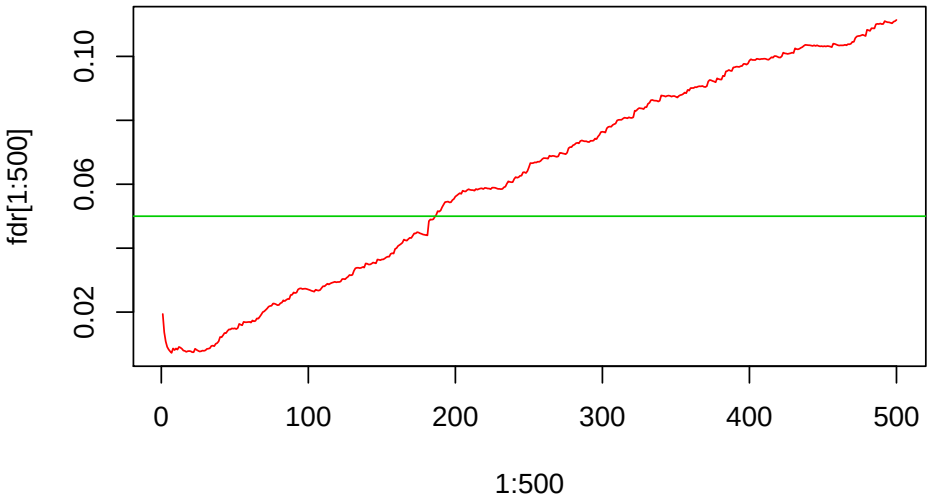


Figure 7.5: First 500 $f_{(l)}$ statistics for the `nci` microarray data.

Chapter 8

Linear Bayesian inference

8.1 Introduction

Much of this course has been concerned with methods for the analysis of multivariate data which are exploratory. When inference has been the primary objective, we have mainly adopted a frequentist perspective, considering the properties of estimators under repeated sampling. We now adopt a Bayesian viewpoint, combining the likelihood of our data with a prior distribution in order to obtain a posterior probability distribution representing our uncertainty about the object of interest conditional on having observed the data.

We adopt a standard convention in Bayesian statistics of using the notation $\pi(\cdot)$ to represent all probability densities of interest. The particular density will be clear from the arguments of the density function. So, a prior for a parameter θ may be represented by $\pi(\theta)$, the likelihood for some data y that is dependent on θ may be represented by $\pi(y|\theta)$ or $L(\theta; y)$, and the posterior for the parameter given the data by

$$\pi(\theta|y) \propto \pi(\theta)L(\theta; y).$$

8.2 Bayesian inference for the mean of an MVN

8.2.1 General solution

We now reconsider the problem of inference for the mean of a MVN distribution from an iid sample. We examined this problem in Chapter 3, finding that the sample mean was the MLE, in addition to being a consistent unbiased estimator. Here we examine the problem of inference for the mean from a Bayesian viewpoint. To keep things simple (and linear), we consider only the case where the variance matrix, Σ, is known.

We consider an $n \times p$ data matrix X with iid rows $\boldsymbol{X}_i \sim N(\boldsymbol{\mu}, \Sigma)$, and we assume a prior on $\boldsymbol{\mu}$ of the form

$$\boldsymbol{\mu} \sim N(\boldsymbol{\mu}_0, \Sigma_0),$$

so the prior density is

$$\pi(\boldsymbol{\mu}) \propto \exp\left\{-\frac{1}{2}(\boldsymbol{\mu} - \boldsymbol{\mu}_0)^\mathsf{T}{\Sigma_0}^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_0)\right\}.$$

We know from Section 3.3.3 that the likelihood of the observed sample is given by

$$L(\boldsymbol{\mu}, \Sigma; \mathsf{X}) = (2\pi)^{-np/2}|\Sigma|^{-n/2} \exp\left\{-\frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu})^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right\}.$$

We can combine the prior and the likelihood to obtain the kernel of the posterior as

$$\begin{aligned}
\pi(\boldsymbol{\mu}|\Sigma, \mathsf{X}) &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{\mu} - \boldsymbol{\mu}_0)^\mathsf{T}{\Sigma_0}^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_0)\right\} \exp\left\{-\frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu})^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left[(\boldsymbol{\mu} - \boldsymbol{\mu}_0)^\mathsf{T}{\Sigma_0}^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_0) + \sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu})^\mathsf{T}\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})\right]\right\} \\
&\propto \cdots \\
&\propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{\mu}^\mathsf{T}({\Sigma_0}^{-1} + n\Sigma^{-1})\boldsymbol{\mu} - 2\boldsymbol{\mu}^\mathsf{T}({\Sigma_0}^{-1}\boldsymbol{\mu}_0 + n\Sigma^{-1}\bar{\boldsymbol{x}})\right]\right\}.
\end{aligned}$$

By completing the square we conclude that

$$\boldsymbol{\mu}|\Sigma, \mathsf{X} \sim N\left(({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1}({\Sigma_0}^{-1}\boldsymbol{\mu}_0 + n\Sigma^{-1}\bar{\boldsymbol{x}}), ({\Sigma_0}^{-1} + n\Sigma^{-1})^{-1}\right).$$

As always, we should avoid explicit matrix inversion when computing solutions. Here there are various strategies which could be used, depending on the precise context of the problem. The most general strategy, which often works well when computing a Bayesian analysis involving MVN random quantities, is to work entirely with precision matrices. This avoids essentially all explicit inversion. However, other strategies are also possible in special cases.

8.2.2 Proportional prior

In the case of known Σ that we are concentrating on here, it is very natural to represent uncertainty about $\boldsymbol{\mu}$ using a variance matrix that is proportional to the variability of $\boldsymbol{X}$. That is, it is natural to put

$$\Sigma_0 = \lambda^{-1}\Sigma,$$

for some λ representing how uncertain we are about μ. Note that we have chosen to parametrise the problem so that our uncertainty increases as we decrease λ. Assuming a form for Σ in this way leads to a dramatic simplification of the posterior to

$$\boldsymbol{\mu}|\Sigma, \mathsf{X} \sim N\left(\frac{\lambda}{n+\lambda}\boldsymbol{\mu}_0 + \frac{n}{n+\lambda}\bar{\boldsymbol{x}}, \frac{1}{n+\lambda}\Sigma\right).$$

In this case the posterior mean is a simple linear combination of the prior mean and the sample mean, and in particular, is independent of Σ. Also note that in the vague prior limit of $\lambda \longrightarrow 0$, we get the posterior

$$\boldsymbol{\mu}|\Sigma, \mathsf{X} \sim N\left(\bar{\boldsymbol{x}}, \frac{1}{n}\Sigma\right),$$

which is exactly the same as the frequentist sampling distribution of $\bar{\boldsymbol{x}}$. Note that this is also the large n (big data) limit.

8.2.3 Spherical prior

Another natural prior on μ is a spherical (or *ridge*) prior which puts $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\Sigma_0 = \lambda^{-1}\,\mathrm{I}$. This leads to a posterior of the form

$$\boldsymbol{\mu}|\Sigma, \mathsf{X} \sim N\left(n(\lambda\Sigma + n\,\mathrm{I})^{-1}\bar{\boldsymbol{x}}, (n\Sigma^{-1} + \lambda\,\mathrm{I})^{-1}\right).$$

Note that the $\lambda \longrightarrow 0$ and large n limits are exactly as above. Although it may superficially look as though the above posterior will require explicit inversion for finite λ and n, this is not the case. Again, the best strategy will depend upon the context, but suppose that we can compute the spectral decomposition of Σ as

$$\Sigma = \mathsf{V}\mathsf{D}\mathsf{V}^{\mathsf{T}},$$

then the posterior becomes

$$\boldsymbol{\mu}|\Sigma, \mathsf{X} \sim N\left(n\mathsf{V}(\lambda\mathsf{D} + n\,\mathrm{I})^{-1}\mathsf{V}^{\mathsf{T}}\bar{\boldsymbol{x}}, \mathsf{V}(n\mathsf{D}^{-1} + \lambda\,\mathrm{I})^{-1}\mathsf{V}^{\mathsf{T}}\right),$$

requiring only inversion of diagonal matrices, which is trivial.

Note that since the main computational cost for high dimensional problems will be the symmetric eigendecompostion, this representation of the posterior shows that we can obtain the posterior for different values of λ for negligible additional cost. Also note that if Σ is actually a sample variance matrix, then it is better to derive the spectral decomposition indirectly via the SVD of the centred data matrix, as discussed in Chapter 2.

Galaxy data

Let's begin by computing the sample mean and variance of the galaxy data

```
> xbar=colMeans(galaxy)
> xbar
      east.west     north.south           angle radial.position
         velocity
     -0.3323685       1.5210889      80.8900929      -0.8427245
       1593.6253870
> sigma=var(galaxy)
```

We will assume that the true variance matrix, Σ is the sample variance matrix. We can calculate the posterior mean under the ridge prior for $\lambda = 0.001$ simply from the definition with

```
> n=dim(galaxy)[1]
> p=dim(galaxy)[2]
> lambda=0.001
> ridge=n*solve(lambda*sigma+n*diag(p),xbar)
> ridge
      east.west     north.south           angle radial.position
         velocity
      -2.485686       10.757489       80.345196       -8.666478
         1551.054721
> sqrt(sum(ridge*ridge))
[1] 1553.198
> sqrt(sum(xbar*xbar))
[1] 1595.678
```

Note that although the estimate is a slight shrinkage towards zero overall (as can be verified by looking at the lengths of the vectors), not every element is shrunk towards zero, due to the correlation structure. We now re-compute the ridge estimate using the eigendecomposition of the variance matrix as follows.

```
> e=eigen(sigma,symmetric=TRUE)
> ridge2=as.vector(n*e$vectors%*%diag(1/(lambda*e$values+n*rep(1,p)))
   %*%t(e$vectors)%*%xbar)
> ridge2
[1]   -2.485686   10.757489   80.345196   -8.666478 1551.054721
```

We see that this gives the same result, and could be re-computed for different values of λ without repeating the spectral decomposition.

8.3 Bayesian inference for the normal linear model

8.3.1 General solution

Let us now turn attention to a Bayesian analysis of the normal linear model. To keep the notation simple, we will concentrate on the case of a univariate output, but it is important to know that the analysis generalises to the general linear model in a straightforward manner. We therefore consider the model

$$\boldsymbol{y} = \mathsf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim N(\mathbf{0}, \sigma^2 \mathrm{I}).$$

Analogously with the previous section, we consider the case of known σ. We will adopt a MVN prior for $\boldsymbol{\beta}$ of the form

$$\boldsymbol{\beta} \sim N(\boldsymbol{\beta}_0, \Sigma).$$

Again, calculations are more straightforward when working with precisions rather than variances, so we will define $\mathsf{K} = \Sigma^{-1}$ and $\tau = \sigma^{-2}$, so that

$$\boldsymbol{\beta} \sim N(\boldsymbol{\beta}_0, \mathsf{K}^{-1}) \quad \text{and} \quad \boldsymbol{\varepsilon} \sim N(\mathbf{0}, \tau^{-1} \mathrm{I}).$$

Now from Chapter 3 we know that the likelihood for $\boldsymbol{\beta}$ takes the form

$$\begin{aligned} L(\boldsymbol{\beta}; \boldsymbol{y}) &= (2\pi)^{-n/2} \tau^{n/2} \exp\left\{-\frac{\tau}{2} \boldsymbol{\varepsilon}^{\mathsf{T}} \boldsymbol{\varepsilon}\right\} \\ &= (2\pi)^{-n/2} \tau^{n/2} \exp\left\{-\frac{\tau}{2} (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})^{\mathsf{T}} (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})\right\}. \end{aligned}$$

Combining this with the prior gives a posterior of the form

$$\begin{aligned} \pi(\boldsymbol{\beta}|\boldsymbol{y}) &\propto \exp\left\{-\frac{1}{2}\left[(\boldsymbol{\beta} - \boldsymbol{\beta}_0)^{\mathsf{T}} \mathsf{K} (\boldsymbol{\beta} - \boldsymbol{\beta}_0) + \tau (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})^{\mathsf{T}} (\boldsymbol{y} - \mathsf{X}\boldsymbol{\beta})\right]\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left[\boldsymbol{\beta}^{\mathsf{T}} (\mathsf{K} + \tau \mathsf{X}^{\mathsf{T}}\mathsf{X}) \boldsymbol{\beta} - 2\boldsymbol{\beta}^{\mathsf{T}} (\mathsf{K}\boldsymbol{\beta}_0 + \tau \mathsf{X}^{\mathsf{T}} \boldsymbol{y})\right]\right\}. \end{aligned} \tag{8.1}$$

Completing the square gives the posterior distribution

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N((\mathsf{K} + \tau \mathsf{X}^{\mathsf{T}}\mathsf{X})^{-1} (\mathsf{K}\boldsymbol{\beta}_0 + \tau \mathsf{X}^{\mathsf{T}} \boldsymbol{y}), (\mathsf{K} + \tau \mathsf{X}^{\mathsf{T}}\mathsf{X})^{-1}).$$

Again, explicit inversion should be avoided. This is simplest when K and $\mathsf{X}^\mathsf{T}\mathsf{X}$ are simultaneously diagonalisable, including when K is diagonal, and we will examine some such cases subsequently. In the most general case, there is still a useful trick, based on regarding the prior as pseudo-observations, which reduces the posterior to a least squares problem (cf. p.227 of Murphy, 2012). This, we have seen, can be solved using a QR decomposition of X.

First, begin by augmenting our data with p observations $\boldsymbol{z}$ with covariates in a $p \times p$ covariate matrix Λ. Our augmented data and covariate matrix then takes the form

$$\tilde{\mathsf{X}} = \begin{pmatrix} \mathsf{X} \\ \Lambda \end{pmatrix}, \quad \tilde{\boldsymbol{y}} = \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{pmatrix}.$$

Next note that

$$\begin{aligned}(\tilde{\boldsymbol{y}} - \tilde{\mathsf{X}}\boldsymbol{\beta})^\mathsf{T}(\tilde{\boldsymbol{y}} - \tilde{\mathsf{X}}\boldsymbol{\beta}) &= (\boldsymbol{y} - \mathsf{X}\beta)^\mathsf{T}(\boldsymbol{y} - \mathsf{X}\beta) + (\boldsymbol{z} - \Lambda\boldsymbol{\beta})^\mathsf{T}(\boldsymbol{z} - \Lambda\boldsymbol{\beta}) \\ &= (\boldsymbol{y} - \mathsf{X}\beta)^\mathsf{T}(\boldsymbol{y} - \mathsf{X}\beta) + (\boldsymbol{\beta} - \Lambda^{-1}\boldsymbol{z})^\mathsf{T}\Lambda^\mathsf{T}\Lambda(\boldsymbol{\beta} - \Lambda^{-1}\boldsymbol{z}) + \text{const},\end{aligned}$$

where const is independent of $\boldsymbol{\beta}$. Multiplying through by τ gives

$$\tau(\tilde{\boldsymbol{y}} - \tilde{\mathsf{X}}\boldsymbol{\beta})^\mathsf{T}(\tilde{\boldsymbol{y}} - \tilde{\mathsf{X}}\boldsymbol{\beta}) = \tau(\boldsymbol{y} - \mathsf{X}\beta)^\mathsf{T}(\boldsymbol{y} - \mathsf{X}\beta) + (\boldsymbol{\beta} - \Lambda^{-1}\boldsymbol{z})^\mathsf{T}\tau\Lambda^\mathsf{T}\Lambda(\boldsymbol{\beta} - \Lambda^{-1}\boldsymbol{z}) + \text{const}$$

Comparison with the posterior density for $\boldsymbol{\beta}$ (8.1) makes clear that by choosing $\mathsf{K} = \tau\Lambda^\mathsf{T}\Lambda$ and $\boldsymbol{\beta}_0 = \Lambda^{-1}\boldsymbol{z}$, the posterior for β is just the sampling distribution of the least squares estimate for the augmented data,

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N((\tilde{\mathsf{X}}^\mathsf{T}\tilde{\mathsf{X}})^{-1}\tilde{\mathsf{X}}^\mathsf{T}\tilde{\boldsymbol{y}}, \tau^{-1}(\tilde{\mathsf{X}}^\mathsf{T}\tilde{\mathsf{X}})^{-1}).$$

We already know how to compute this efficiently using the QR factorisation of $\tilde{\mathsf{X}}$. So, all we need to do is compute Λ as the *upper* Cholesky factor of K/τ, and then $\boldsymbol{z} = \Lambda\boldsymbol{\beta}_0$ in order to reduce the problem to the problem of ordinary least squares. Explicitly, if we let the QR factorisation of $\tilde{\mathsf{X}}$ be $\tilde{\mathsf{X}} = \mathsf{QR}$, then we have

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N(\mathsf{R}^{-1}\mathsf{Q}^\mathsf{T}\tilde{\boldsymbol{y}}, \tau^{-1}\mathsf{R}^{-1}\mathsf{R}^{\mathsf{T}-1}).$$

So, for example, we can compute the posterior mean of $\boldsymbol{\beta}$, $\boldsymbol{\beta}_1$, by back-solving the triangular linear system

$$\mathsf{R}\boldsymbol{\beta}_1 = \mathsf{Q}^\mathsf{T}\tilde{\boldsymbol{y}}.$$

Samples can also be generated from the posterior with one more triangular solve.

Galaxy data

In Chapter 3 we looked at regressing `velocity` on `angle` and `radial.position` in the `galaxy` data.

```
> m=lm(velocity~angle+radial.position,data=galaxy)
> summary(m)

Call:
lm(formula = velocity ~ angle + radial.position, data = galaxy)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-199.368  -50.481   -4.025   52.448  174.266

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     1586.9882     9.1264 173.891   <2e-16 ***
angle              0.1076     0.1021   1.054    0.293
radial.position    2.4515     0.1508  16.253   <2e-16 ***
---
Signif. codes:  0    ***    0.001    **    0.01    *    0.05    .
   0.1         1

Residual standard error: 69.98 on 320 degrees of freedom
Multiple R-squared:  0.4523,    Adjusted R-squared:  0.4488
F-statistic: 132.1 on 2 and 320 DF,  p-value: < 2.2e-16
```

Let us now suppose that we have the following prior on $\boldsymbol{\beta}$:

$$\boldsymbol{\beta} \sim N\left(\begin{pmatrix}1500\\0\\0\end{pmatrix}, \begin{pmatrix}10000 & 0 & 0\\0 & 9 & 5\\0 & 5 & 25\end{pmatrix}\right).$$

and assume that τ is known to be 70^{-2} (which is consistent with the residual standard error, above). We can construct the posterior mean of $\boldsymbol{\beta}$ directly as follows.

```
> tau=70^(-2)
> K=solve(matrix(c(10000,0,0,0,9,5,0,5,25),ncol=3,byrow=TRUE))
> beta0=c(1500,0,0)
> X=model.matrix(velocity~angle+radial.position,data=galaxy)
> y=galaxy$velocity
> beta1=solve(K+tau*t(X)%*%X,K%*%beta0+tau*t(X)%*%y)
> beta1
                         [,1]
(Intercept)     1586.2347082
angle              0.1152683
radial.position    2.4494407
```

We now recompute the solution using the augmented data approach.

```
> lambda=chol(K/tau)
> z=lambda%*%beta0
> Xtilde=rbind(X,lambda)
> ytilde=c(y,z)
> lm(ytilde~0+Xtilde)

Call:
lm(formula = ytilde ~ 0 + Xtilde)

Coefficients:
    Xtilde(Intercept)            Xtildeangle  Xtilderadial.position
            1586.2347                 0.1153                 2.4494
```

After augmenting the data appropriately we can just use the built-in `lm()` function to solve the least squares problem to obtain the correct posterior mean.

8.3.2 Noise invariance

It is possible to make the posterior mean invariant to the noise in the data by choosing Σ to be proportional to the noise variance, σ^2. That is, we can choose Σ to be of the form $\Sigma = \sigma^2\Sigma_0$ for fixed matrix Σ_0, or equivalently, $\mathsf{K} = \tau\mathsf{K}_0$ for fixed K_0 $(= {\Sigma_0}^{-1})$. This leads to the posterior

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N((\mathsf{K}_0 + \mathsf{X}^\mathsf{T}\mathsf{X})^{-1}(\mathsf{K}_0\boldsymbol{\beta}_0 + \mathsf{X}^\mathsf{T}\boldsymbol{y}), \tau^{-1}(\mathsf{K}_0 + \mathsf{X}^\mathsf{T}\mathsf{X})^{-1}).$$

Obviously we can always do this when τ is really known. However, often τ isn't really known, but it is nevertheless convenient to consider the posterior distribution of $\boldsymbol{\beta}$ *conditional* on a fixed value of τ. In this case the invariance of the posterior mean to τ is both a bug and a feature. It is a feature in that the posterior mean does not depend on τ, and is analytically convenient for various reasons. First, it lends itself to a conjugate analysis for unknown $\boldsymbol{\beta}$ and τ that we will not consider here. However, looking back to the previous section on pseudo-data, it is clear that a prior of this form makes the pseudo-data invariant to the choice of τ, which means that neither the pseudo-data nor the QR factorisation of the pseudo-covariate matrix will need to be re-computed should the value of τ change during an iterative algorithm. Clearly there are positives associated with parametrising uncertainty about $\boldsymbol{\beta}$ in this way. However, there are also very undesirable features of this prior. In particular, it implies that if the noise were to decrease to zero our *a priori* uncertainty regarding $\boldsymbol{\beta}$ would also tend to zero. This is not really plausible, in practice.

8.3.3 Spherical prior (Bayesian ridge regression)

If we use a noise invariant prior, as above, and further assume $\boldsymbol{\beta}_0 = \mathbf{0}$ and $\mathsf{K}_0 = \lambda\,\mathrm{I}$, then the posterior for $\boldsymbol{\beta}$ takes the form

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N((\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y}, \tau^{-1}(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})^{-1}).$$

The mean (and mode) of this posterior is precisely the ridge regression estimate. That is, the posterior mean for $\boldsymbol{\beta}$, $\boldsymbol{\beta}_1$, is the solution of the ridge equation

$$(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})\boldsymbol{\beta}_1 = \mathsf{X}^\mathsf{T}\boldsymbol{y},$$

which is the minimiser of the ridge penalty

$$(\boldsymbol{y} - \mathsf{X}\beta)^\mathsf{T}(\boldsymbol{y} - \mathsf{X}\beta) + \lambda\boldsymbol{\beta}^\mathsf{T}\boldsymbol{\beta}.$$

Obviously this leads to the sampling distribution of the least squares estimator, $\hat{\boldsymbol{\beta}}$, as $\lambda \longrightarrow 0$. Explicitly,

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N((\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y}, \tau^{-1}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}).$$

As previously discussed, this can be computed efficiently using the QR decomposition of X.

For finite λ we can solve this problem efficiently using the SVD of X. If $\mathsf{X} = \mathsf{UDV}^\mathsf{T}$ is the SVD of X, then

$$\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I} = \mathsf{V}(\mathsf{D}^2 + \lambda\,\mathrm{I})\mathsf{V}^\mathsf{T},$$

and hence

$$(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})^{-1} = \mathsf{V}(\mathsf{D}^2 + \lambda\,\mathrm{I})^{-1}\mathsf{V}^\mathsf{T}.$$

Consequently,

$$(\mathsf{X}^\mathsf{T}\mathsf{X} + \lambda\,\mathrm{I})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y} = \mathsf{V}(\mathsf{D}^2 + \lambda\,\mathrm{I})^{-1}\mathsf{D}\mathsf{U}^\mathsf{T}\boldsymbol{y}.$$

Together these relations allow the efficient solution of this problem without explicit inversion of anything other than diagonal matrices. Also note that if the SVD of X is the dominant computational cost, then the solution of the problem for many different values of λ can be obtained for negligible additional cost.

Galaxy data

We start by directly constructing the posterior mean for $\lambda = 0.1$.

```
> lambda=0.1
> beta1=solve(t(X)%*%X+lambda*diag(3),t(X)%*%y)
> beta1
                          [,1]
(Intercept)     1584.2940395
angle              0.1348488
radial.position    2.4528549
```

We can now re-compute this using the SVD of X as follows.

```
> s=svd(X)
> beta2=s$v%*%diag(s$d/(s$d^2 + lambda*rep(1,3)))%*%t(s$u)%*%y
> beta2
             [,1]
[1,] 1584.2940395
[2,]    0.1348488
[3,]    2.4528549
```

This gives the same results as before, and can be computed for different values of λ without re-doing the SVD of X.

The above example throws up an important issue regarding shrinkage for ridge regression. Here there is shrinkage of the regression coefficients, but this happens via shrinkage of the intercept. In many cases, we would like shrinkage of the regression coefficients associated with the predictors, but not shrinkage of the intercept. For this we can centre the data (both the response and the covariates), and then fit the regression model without an intercept. We do this first for basic least squares regression as follows.

```
> X=as.matrix(galaxy[c("angle","radial.position")])
> X=sweep(X,2,colMeans(X))
> y=galaxy$velocity-mean(galaxy$velocity)
> lm(y ~ 0+X)

Call:
lm(formula = y ~ 0 + X)
```

```
Coefficients:
          Xangle  Xradial.position
          0.1076            2.4515
```

This confirms that the least squares regression coefficients are unaffected. We can now re-compute the ridge estimate with

```
> beta3=solve(t(X)%*%X+lambda*diag(2),t(X)%*%y)
> beta3
                     [,1]
angle            0.107592
radial.position 2.451480
```

showing that we get very little shrinkage for $\lambda = 0.1$. We need to choose a fairly large value of λ to get any noticeable shrinkage in this example.

```
> lambda=10000
> beta4=solve(t(X)%*%X+lambda*diag(2),t(X)%*%y)
> beta4
                      [,1]
angle            0.1017795
radial.position 2.3425133
```

To see how the solution varies with λ, it is helpful to use the SVD approach.

```
> s=svd(X)
> beta=function(lambda) s$v%*%diag(s$d/(s$d^2 + lambda*rep(1,2)))%*%t(s
   $u)%*%y
> llambda=0:25
> betamat=sapply(exp(llambda),beta)
> plot(llambda,betamat[1,],type="l",col=2,ylim=c(0,3),xlab="log(lambda)
   ",ylab="beta")
> lines(llambda,betamat[2,],col=3)
```

Figure 8.1 shows how the ridge estimates of the regression coefficients gradually shrink to zero with increasing λ.

8.3.4 g-prior

Again we consider the noise invariant prior, but now choose $\boldsymbol{\beta}_0 = \mathbf{0}$ and $\mathsf{K}_0 = g^{-1}\mathsf{X}^\mathsf{T}\mathsf{X}$ for some g. Here, large g corresponds to large uncertainty. Clearly then we have

$$\boldsymbol{\beta}|\boldsymbol{y} \sim N\left(\frac{g}{1+g}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\mathsf{X}^\mathsf{T}\boldsymbol{y}, \tau^{-1}\frac{g}{1+g}(\mathsf{X}^\mathsf{T}\mathsf{X})^{-1}\right).$$

For values of g much bigger than one this corresponds to a slight shrinkage of the least squares sampling distribution, and gives the least squares distribution in the limit $g \longrightarrow \infty$. The g-prior has attractive scale-invariance properties associated with re-scaling covariates. In particular, covariates with a large variance correspond to elements of $\boldsymbol{\beta}$ with large prior uncertainty.

We already know how to compute solutions to this problem efficiently by using the QR decomposition of X.

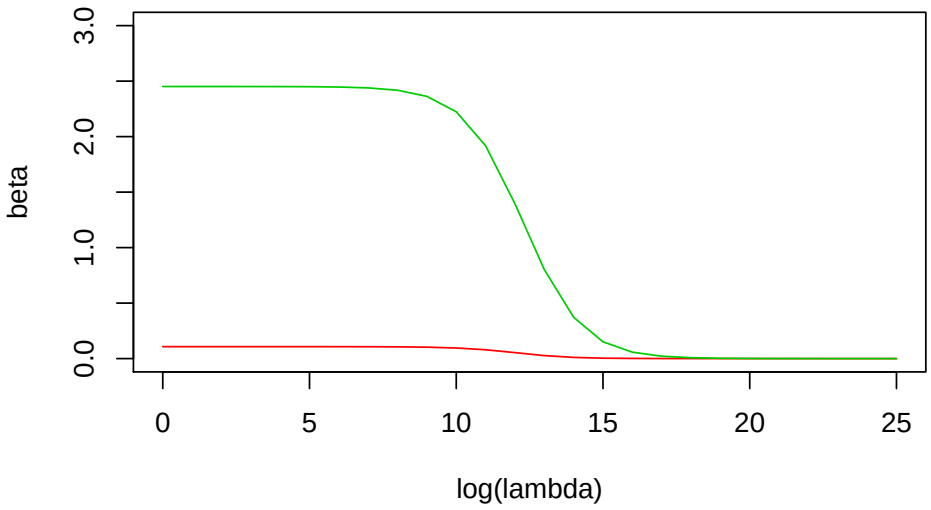


Figure 8.1: Ridge regression coefficients as a function of λ.