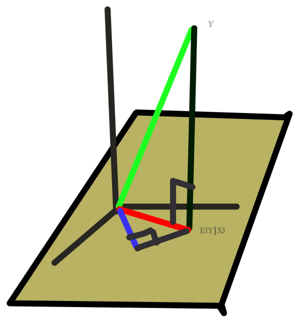


# Powered by Linear Algebra

**The central role of matrices and vector spaces in data science**

Norman Matloff

2025-08-17

# Table of contents

# 1

# Preface

Welcome to my magnum opus! :-) I've written a number of books, but consider this one to be the most important.

## Subtlety in the Title

Let's start with the title of this book, *Powered by Linear Algebra: The central role of matrices and vector spaces in Data Science*. It's important to understand why the title is NOT "Linear Algebra for Data Scientists." That latter would wrongly connote that people in Data Science (DS)will first learn linear algebra purely as a branch of math in this book, with no hint of connections to DS, then apply that knowledge in subsequent DS courses. Instead, the goal in the title is to emphasize the fact that:

> Linear algebra is absolutely fundamental to the Data Science field. For us data scientists, it is "our" branch of math. Almost every concept in this book is first motivated by a Data Science application. Mastering this branch of math, which is definitely within the reach of all, pays major dividends.

## Philosophy

*I learned very early the difference between knowing the name of something and knowing something* – physicist Richard Feynman

*...it felt random. "Follow these steps and you get the result you are looking for." But why does it work? What possessed you to follow this path as opposed to any other? How might I have*

*come up with this myself?* – comment by a reader of a famous linear algebra book

This book does not allow rote memorization, merely "knowing the name of something." The focus on How? and Why? is on every page.

A fundamental philosophy of my book here is to avoid reader frustration. It's easy to define, say the dimension of a vector subspace, but that's definitely not enough. What is the underlying intuition? Why is the concept important, especially in Data Science?

The presentation of each concept in this book begins with a problem to be solved, almost always from Data Science, then leading up to a linear algebra solution. Basically, the math sneaks up on the reader, who suddenly realizes they've just learned a new general concept! And the reader knows where the concept fits into the Big Picture, and can distill the abstraction into an intuitive summary.

Examples:

- In Chapter 1, we use Markov chain transition matrices and network graph models (e.g. social networks) to motivate the notion of a matrix and matrix multiplication. An interest in finding the stationary distribution of a Markov chain then leads to the concept of matrix inverses. (Linear models are presented later, after some groundwork is laid.)

- The chapter on matrix rank starts with a dataset right off the bat, and shows that R's linear model function **lm** fails if categorical variables are fully specified. This motivates the notion of rank, and the dataset dovetails with the theory throughout the chapter, which culminates in a proof that row rank equals column rank.

- The chapter on eigenanalysis begins with explaining the goals of PCA (with a real dataset). We derive the first PC as a constrained maximization of variance, and behold! – the solution turns out to have the form $Ax = \lambda x$!

So eigenanalysis comes from solving a Data Science problem. PCA is later covered in detail in the following chapter, but with this motivation we develop the properties of eigenvalues and eigenvectors in the current chapter.

## Who Is This Book For?

Of course the book should work very well as a classroom textbook. If a Data Science or Statistics program requires linear algebra offered by a Math Department, the mathematical content of this book should be similar to that math course, but with much better student motivation due to the Data Science emphasis of the book. The "applications first" approach is key to that motivational power.

The applications-centered nature of the book should make teaching the course more rewarding for instructors as well, and the use of Quarto enables easy conversion to Powerpoint by instructors.

I also hope the book's emphasis on the How? and Why? especially appeals to do-it-yourselfers, those whose engagement in self-study is motivated by intellectual curiosity rather than a course grade.

## Prerequisite Background

Basic data science:

- Calculus, e.g. derivatives as rates of change and slopes of tangent lines.
- Basics of random variables, e.g. density functiopns, expected value and variance.
- Some exposure to R is recommended, but the text can be read without it.

For a quick, painless introduction to R, see my fasteR tutorial, say the first 8 lessons.

## The Role of Math (and R)

Earlier in this Preface, I said, "If a Data Science or Statistics program requires linear algebra offered by a Math Department, the mathematical content of this book should be similar to that math course…" So, the mathematics is definitely here, but one might also say that this book is "mathematical but not overly theoretical."

*Theory:*

Theorems are mainly limited to results with practical importance. Among the Your Turn exercises at the end of each chapter, many of the ones requiring proofs are simple, of the "one liner" type. But the subject matter is indeed mathematical. Students are indeed expected to read and understand proofs, just as with the Mathematics Department course.

*Active reading:*

No sections are designated as "starred," i.e. reserved for the more mathematically adept readers. I believe all material in the book is within reach of any reader who did reasonably solid work in their calculus courses. But active participation on the part of the reader is key.

For instance, Chapter Chapter 4 is on very concrete topics and the reader hopefully will find the "3-dimensional bell shape" of the multivariate distribution family intriguing. Yet, due to an overloading for the term *covariance* in statistics, the reader must always actively keep in mind whether it is scalar or vector covariance being discussed.

As another example, Section Section 10.3 is a bit abstract and involves switching back and forth between random variables and members of a certain vector space, but it is presented with intuitive motivation and simple, concrete examples. Moreover, the material is motivated by culminating in an applied topic of major societal relevance. The patient, active reader who keeps in mind that certain entities here have dual roles will be rewarded with a plethora of insights into inner product spaces, and again, any successful "calculus graduate" should find this attainable, even eliciting an empowered feeling. For this reason, this section is *not* “starred,” nor are the other parts of the book so designated.

*Not a “how to do linear algebra in R” book:*

The goal is to develop in the reader mathematical skill and intuition into this powerful tool, rather than coding of linear algebra methods. Thus the many R examples are meant to make the mathematical concepts concrete, not as an “how to do linear algebra in R” book. In the software context, the Feynman quote above might be, “There is a difference between knowing how to use code libraries for something and knowing the core nature of that thing.” That said, the code examples do serve a vital role.

*Not intended as an “easier” version of your Math Dept. course:*

*The applied nature of the book is a double-edged sword.* It is has high value as a motivator, but understanding applications is actually *more* challenging than a purely mathematical treatment, not less so. Math is more crisply-defined, while applications can be “fuzzy.” Among the Your Turn exercises, many of the applied ones are somewhat open-ended, and they tend to be wordier than the theory ones. But often these are the ones that develop genuine understanding of the subject.

In other words, the book is intended to arm students with *usable practical insights*, rather than merely satisfying some curricular requirement that will be quickly forgotten. Hence the needs for (a) developing student intuition and (b) nonpassive learning are paramount (as they should be in any Data Science course).

## R Packages Used

```
dsld
ellipse
glmnet
igraph
imager
networkdata
```

```
pracma
qeML
regclass
WackyData
```

## Data Availabilty

The datasets used are included with the above packages.

## Web Site

*github.com/matloff/WackyLinear Algebra*

## Edition Number

Currently 1.0.0. Correction of typos etc. will usually increment the third digit.

## Permission to Copy

## Thanks

I deeply appreciate feedback from: Mike Hannon, Nick Knueppel, Joe Rickert and Noah Perry.

# 2 Matrices and Vectors

ℹ Goals of this chapter:

The two main structures in linear algebra are *matrices* and *vector spaces*. We begin the book with the former, introduced in this chapter, motivating matrix multiplication and presenting several applications.

In this chapter, we will take as our main application *Markov chains*, a statistical model having wide applications in medicine, economics and so on. The notion is very simple to explain, thus making it a good choice for introducing matrices.

## 2.1 A Random Walk Model

Let's consider a *random walk* on {1,2,3,4,5} in the number line. Time is numbered 1,2,3,... Our current position is termed our *state*. The notation $\mathrm{X_k} = \mathrm{i}$ means that at time k we are in state/position i.

Our rule will be that at any time k, we flip a coin. If we are currently at position i, we move to either i+1 or i-1, depending on whether the coin landed heads or tails. The exceptions are k = 1 and k = 5, in which case we stay put if tails or move to the adjacent position if heads.

We can summarize the probabilities with a *matrix*, a two-dimensional array:

$$P_1 = \left( \begin{array}{rrrrr} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0.5 \end{array} \right) \tag{2.1}$$

For instance, look at row 2. There are 0.5 values in columns 1 and 3, meaning there is a 0.5 chance of a move 2 → 1, and a 0.5 chance of a move 2 → 3.

We use a subscript 1 here in $P_1$, meaning "one step." We go from, say, state 2 to state 1 in one step with probability 0.5. $P_1$

is called the *one-step transition matrix* (or simply the *transition matrix*) for this process.

Note that each row in a transition matrix must sum to 1. After all, from state i we must go *somewhere.*

What about the two-step transition matrix $P_2$? For instance, what should be in the row 3, column 1 position in that matrix? In other words, if we start at position 3, what is the probability that we go to position 1 in two steps? This would happen via two tails flips of the coin. The probability of that is $0.5^2 = 0.25$. So the row 3, column 1 element in $P_2$ is 0.25. On the other hand, if from state 3 we flip tails then heads, or heads then tails, we are back to state 3. So, the row 3, column 3 element in $P_2$ is $0.25 + 0.25 = 0.5$.

The reader should verify the correctness here:

$$P_2 = \left( \begin{array}{rrrrr} 0.5 & 0.25 & 0.25 & 0 & 0 \\ 0.25 & 0.5 & 0 & 0.25 & 0 \\ 0.25 & 0 & 0.5 & 0 & 0.25 \\ 0 & 0.25 & 0 & 0.5 & 0.25 \\ 0 & 0 & 0.25 & 0.25 & 0.5 \end{array} \right)$$

Well, finding two-step transition probabilities would be tedious in general, but it turns out that is a wonderful shortcut: Matrix multiplication. We will cover this in the next section, but first a couple of preliminaries.

The above random walk is a *Markov chain.* The Markov Property says that the system "has no memory." If say we land at position 2, we will go to 1 or 3 with probability 1/2 *no matter what the previous history of the system was*; it doesn't matter *how* we got to state 3. That in turn comes in this example from the independence of the successive coin flips.

**Notation:** Individual elements of a matrix are usually written with double subscripts. For instance, $a_{25}$ will mean the row 2, column 5 element of the matrix $A$. If say $A$ has more than 9 rows, its row 11, column 5 element is denoted by $a_{11,5}$, using the comma to avoid ambiguity.

## 2.2 Vectors

Matrices are two-dimensional arrays. One-dimensional arrays are called *vectors*, either in row or column form, e.g.

$$u = (12, 5, 13)$$

and

$$u = \left( \begin{array}{r} 12 \\ 5 \\ 13 \end{array} \right)$$

Please note:

- Vectors may also be viewed as one-row or one-column matrices.
- When not otherwise stated, the term "vector" will mean column form.
- The term *scalar* simply means a number, rather than a matrix or vector. It will be used quite frequently in this book.

## 2.3 Addition and Scalar Multiplication

Vectors of the same length may be summed, in elementwise form, e.g.

$$\left( \begin{array}{r} 12 \\ 5 \\ 13 \end{array} \right) + \left( \begin{array}{r} -3 \\ 6 \\ 18.2 \end{array} \right) = \left( \begin{array}{r} 9 \\ 11 \\ 31.2 \end{array} \right)$$

Similarly, two matrices may be added, again in elementwise fashion, provided the number of rows is the same for both, as well as the same condition for number of columns.

Vectors and matrices can be multiplied by scalars, again elementwise, e.g.

$$0.3 \begin{pmatrix} 6 \\ 15 \end{pmatrix} = \begin{pmatrix} 1.8 \\ 4.5 \end{pmatrix}$$

## 2.4 Matrix-Matrix Multiplication

This is the most fundamental operation in linear algebra. It is defined as follows:

> Given matrix $A$ of $k$ rows and $m$ columns and matrix $B$ of $m$ rows and $r$ columns, the product $C = AB$ is a $k \times r$ matrix, whose row $i$, column $j$ element is
>
> $$a_{i1}b_{1j} + a_{i2}b_{2j} + ... + a_{im}b_{mj}$$
>
> *This is the "dot product" of row $i$ of $A$ and column $j$ of $B$: Find the products of the paired elements in the two vectors, then sum.*

The notion of a dot product will arise often in this book.

For example, set

$$A = \begin{pmatrix} 5 & 2 & 6 \\ 1 & 1 & 1 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 5 & -1 \\ 1 & 0 \\ 0 & 8 \end{pmatrix}$$

Let's find the row 2, column 2 element of $C = AB$. Again, that means taking the dot product of row 2 of $A$ and column 2 of $B$, which we've highlighted below.

$$A = \begin{pmatrix} 5 & 2 & 6 \\ \color{red}{1} & \color{red}{1} & \color{red}{1} \end{pmatrix}$$

and

$$B = \begin{pmatrix} 5 & -1 \\ 1 & 0 \\ 0 & 8 \end{pmatrix}$$

The value in question is then

1 (-1) + 1 (0) + 1 (8) = 7

Let's check it, with R:

```r
a <- rbind(c(5,2,6),c(1,1,1))
b <- cbind(c(5,1,0),c(-1,0,8))
a %*% b
```

```
     [,1] [,2]
[1,]   27   43
[2,]    6    7
```

The **rbind** and **cbind** functions ("row bind" and "column bind") are very handy tools for creating matrices. The reader should make sure to check the other elements by hand.

Tip 1

Always keep in mind that in the matrix product $AB$, the number of rows of $B$ must equal the number of columns of $A$. The two matrices are then said to be *conformable*.

## 2.5 The Identity Matrix

The *identity matrix* $I$ of size $n$ is the $n \times n$ matrix with 1s on the diagonal and 0s elsewhere. Here is the one for $n = 2$:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Identity matrices are *multiplicative* identities, i.e. they simply copy the companion factor when multiplied: $IB = B$ and $AI = A$ for any conformable $A$ and $B$.

## 2.6 Application to Markov Chain Transition Matrices

Now let's return to the question of how to easily compute $P_2$, the two-step transition matrix. It turns out that:

> Let P denote the transition matrix of a (finite-state) Markov chain. The k-step transition matrix is $P^k$.

At first, this may seem amazingly fortuitous, but it makes sense in light of the "and/or" nature of the probability computations involved. Recall our computation for the row 1, column 2 element of $P_2$ above. From state 1, we could either stay at 1 for one flip, then move to 2 on the second flip, or we could go to 2 then return to 1. Each of these has probability 0.5, so the total probability is

$$(0.5)(0.5) + (0.5)(0.5)$$

But this is exactly the form of our "dot product" computation in the definition of matrix multiplication,

$$a_{i1}b_{i1} + a_{i2}b_{i1} + ... + a_{m1}b_{m1}$$

Then $P^3$ stores the 3-step probabilities and so on.

Statisticians and computer scientists like to look at the *asymptotic* behavior of systems, meaning what happens to a quantity when time or size or some other value grows. Let's see where we might be after say, 6 steps:

```
matpow <- function(m,k) {
   nr <- nrow(m)
   tmp <- diag(nr)  # identity matrix
   for (i in 1:k) tmp <- tmp %*% m
   tmp
}

p1 <- rbind(c(0.5,0.5,0,0,0), c(0.5,0,0.5,0,0), c(0,0.5,0,0.5,0),
   c(0,0,0.5,0,0.5), c(0,0,0,0.5,0.5))
matpow(p1,6)
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.312500 0.234375 0.234375 0.109375 0.109375
[2,] 0.234375 0.312500 0.109375 0.234375 0.109375
[3,] 0.234375 0.109375 0.312500 0.109375 0.234375
[4,] 0.109375 0.234375 0.109375 0.312500 0.234375
[5,] 0.109375 0.109375 0.234375 0.234375 0.312500
```

So for instance if we start at position 2, there is about an 11% chance that we will be at position 3 at time 6. What about time 25?

```
matpow(p1,25)
```

```
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.2016179 0.2016179 0.1993820 0.1993820 0.1980001
[2,] 0.2016179 0.1993820 0.2016179 0.1980001 0.1993820
[3,] 0.1993820 0.2016179 0.1980001 0.2016179 0.1993820
[4,] 0.1993820 0.1980001 0.2016179 0.1993820 0.2016179
[5,] 0.1980001 0.1993820 0.1993820 0.2016179 0.2016179
```

So, no matter which state we start in, at time 25 we are about 20% likely to be at any of the states. In fact, as time $n$ goes to infinity, this probability vector becomes exactly (0.20,0.20,0.20,0.20,0.20). It will be shown below that the vector of long-run state probabilities $\nu$ is the solution of

$$P\nu = \nu \tag{2.2}$$

where $P$ is the transition matrix for the Markov chain.

## 2.7 Network Graph Models

There has always been lots of analysis of "Who is connected to whom," but activity soared after the advent of Facebook and the film, *A Social Network.* See for instance *Statistical Analysis of Network Data with R* by Eric Kolaczy and Gábor Csárdi. As the authors say,

> The oft-repeated statement that "we live in a connected world" perhaps best captures, in its simplicity why networks have come to hold such interest in recent years. From on-line social networks like Facebook to the World Wide Web and the Internet itself, we are surrounded by examples of ways in which we interact with each other. Similarly, we are connected as well at the level of various human institutions (e.g., governments), processes (e.g., economies), and infrastructures (e.g., the global airline network). And, of course, humans are surely not unique in being members of various complex, inter-connected systems. Looking at the natural world around us, we see a wealth of examples of such systems, from entire eco-systems, to biological food webs, to collections of inter-acting genes or communicating neurons.

And of course, at the center of it all is a matrix! Here is why:

## 2.8 Example: Karate Club

Let's consider the famous Karate Club dataset:

```r
# remotes::install_github("schochastics/networkdata")
library(networkdata)
data(karate)
library(igraph)
```

Calling

```r
plot(karate)
```

then yields

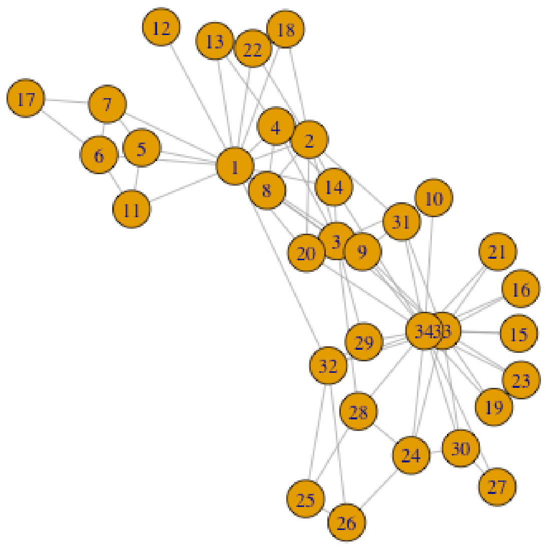


Figure 2.1: Karate Club network

There is a link between node 13 and node 4, meaning that club members 13 and 4 are friends.

This graph is *undirected*, as friendship is mutual. Many graphs are *directed*, but we will assume undirected here.

The *adjacency matrix* has row i, column j element as 1 or 0, according to whether a link exists between nodes i and j.

```r
adjK <- as_adjacency_matrix(karate)
adjK
```

```
34 x 34 sparse Matrix of class "dgCMatrix"
```

```
 [1,] . 1 1 1 1 1 1 1 1 . 1 1 1 1 . . . 1 . 1 . 1 . . . . . . . . . 1 . .
 [2,] 1 . 1 1 . . . 1 . . . . . 1 . . . 1 . 1 . 1 . . . . . . . . 1 . . .
 [3,] 1 1 . 1 . . . 1 1 1 . . . 1 . . . . . . . . . . . . . 1 1 . . . 1 .
 [4,] 1 1 1 . . . . 1 . . . . 1 1 . . . . . . . . . . . . . . . . . . . .
 [5,] 1 . . . . . 1 . . . 1 . . . . . . . . . . . . . . . . . . . . . . .
 [6,] 1 . . . . . 1 . . . 1 . . . . . 1 . . . . . . . . . . . . . . . . .
 [7,] 1 . . . 1 1 . . . . . . . . . . 1 . . . . . . . . . . . . . . . . .
 [8,] 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 [9,] 1 . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . 1 1
[10,] . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
[11,] 1 . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . .
[12,] 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
[13,] 1 . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
[14,] 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
[15,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 1
[16,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 1
[17,] . . . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . .
[18,] 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
[19,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 1
[20,] 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
[21,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 1
[22,] 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
[23,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 1
[24,] . . . . . . . . . . . . . . . . . . . . . . . . . 1 . 1 . 1 . . 1 1
[25,] . . . . . . . . . . . . . . . . . . . . . . . . . 1 . 1 . . . 1 . .
[26,] . . . . . . . . . . . . . . . . . . . . . . . 1 1 . . . . . . 1 . .
[27,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . 1
[28,] . . 1 . . . . . . . . . . . . . . . . . . . . 1 1 . . . . . . . . 1
[29,] . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . 1
[30,] . . . . . . . . . . . . . . . . . . . . . . . 1 . . 1 . . . . . 1 1
[31,] . 1 . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . 1 1
[32,] 1 . . . . . . . . . . . . . . . . . . . . . . . 1 1 . . 1 . . . 1 1
[33,] . . 1 . . . . . 1 . . . . . 1 1 . . 1 . 1 . 1 1 . . . . . 1 1 1 . 1
[34,] . . . . . . . . 1 1 . . . 1 1 1 . . 1 1 1 . 1 1 . . 1 1 1 1 1 1 1 .
```

```r
adjK[13,4]
```

```
[1] 1
```

Accordingly, row 13, column 4 does have a 1 entry, consistent with what we saw in the picture.

## 2.9 The Role of Matrix Multiplication in Network Graph Models

As is the case with Markov transition matrices, powers of an adjacency matrix can yield valuable information. In the Markov case, multiplication gives us sums of paired products, computing probabilities. What about the network graph case?

Here products are of the form $0 \times 0$, $0 \times 1$, $1 \times 1$, If there is a nonzero entry $m$ in row $i$, column $j$ of the square of the adjacency matrix, that means there were $m$ $1 \times 1$ products in that sum, which would correspond to $m$ paths. Let's look into this.

```
adjK2 <- adjK %*% adjK
```

We see that **adjK2[11,1]** is 2. Inspection of **adjK** shows that its row 11, columns 6 and 7 are 1s, and that rows 6 and 7, column 1 are 1s as well. So there are indeed two two-hop paths from node 11 to node 1, specifically $11 \rightarrow 6 \rightarrow 1$ and $11 \rightarrow 7 \rightarrow 1$. Thus the 2 we see in **adjK2[11,1]** was correct.

In other words, $A^k$ for a network adjacency matrix $A$ shows the number of paths from each node to each of the others.

Actually, what is typically of interest is *connectivity* rather than number of paths. For any given pair of nodes, is there a multihop path between them? Or does the graph break down to several "islands" of connected nodes?

Again consider the Karate Club data. Since there are 34 nodes in this graph, if the graph is connected, there should be a path that hits all of them with at most 33 hops. Let's see what paths of this length give us.

```
u <- matpow(adjK,33)
sum(u == 0)
```

```
[1] 0
```

So, in that graph representing paths of 33 links, there are no 0s. In this graph, every pair of nodes has a path between them. The graph is connected.

In general, determining whether a graph is connected requires paying attention to things such as cycles. The details are beyond the scope of this book.

## 2.10 Recommender Systems

If you inquire about some item at an online store, the software will also present you with some related items that it thinks would be of interest to you. How does the software make this guess?

Clearly, the full answer is quite complex. But we can begin to see the process by looking at some real data.

```
site <- 'http://files.grouplens.org/datasets/movielens/ml-100k/u.data'
q <- read.table(site)
names(q) <- c('user','movie','rating','userinfo')
head(q)
```

```
  user movie rating  userinfo
1  196   242      3 881250949
2  186   302      3 891717742
3   22   377      1 878887116
4  244    51      2 880606923
5  166   346      1 886397596
6  298   474      4 884182806
```

We see for instance that user 22 gave movie 377 a rating of 1. If we want to know some characteristics of this user, his/her ID is 878887116, which we can find in the file **u.user** at the above URL. Other files tell us more about this movie, e.g. its genre, and so on.

Let's explore the data a bit:

How many users and movies are in this dataset?

```r
length(unique(q$user))
```

```
[1] 943
```

```r
length(unique(q$movie))
```

```
[1] 1682
```

How many other users rated movie number 242?

```r
sum(q$movie == 242)
```

```
[1] 117
```

Did user 22 rate movie 234, for instance?

```r
which(q$user == 22 & q$movie == 234)
```

```
integer(0)
```

Now we can begin to see a solution to the recommender problem. Say we wish to guess whether user 22 would like movie 234. We could look for other users who have rated many of the same movies as user 22, and whose ratings of those movies were generally similar to those of user 22. We would then focus on the ones who rated movie 234. We could average those ratings to obtain a predicted rating for movie 234 by user 22.

We could also incorporate the characteristics of user 22 and the others, which may improve our prediction accuracy, but we will not pursue that here.

In order to assess interuser similarity of the nature described above, we might form a matrix $S$, as follows. There would be 943 rows, one for each user, and 1682 columns, one for each movie. The element in row $i$, column $j$ would be the rating user $i$ gave to movie $j$. Most of the matrix would be 0s.

The point of constructing $S$ is that determining the similarity of users becomes a matter of measuring similarity of rows of $S$. This paves the way to exploiting the wealth of matrix-centric methodology we will develop in this book.

## 2.11 Matrix Algebra

### 2.11.1 Other basic operations

Matrix multiplication may seem odd at first, but other operations are straightforward.

**Addition:** We just add corresponding elements. For instance,

$$A = \begin{pmatrix} 5 & 2 & 6 \\ 1 & 2.6 & -1.2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 20 & 6 \\ 3 & 5.8 & 1 \end{pmatrix}$$

$$A + B = \begin{pmatrix} 5 & 22 & 12 \\ 4 & 8.4 & -0.2 \end{pmatrix}$$

We do have to make sure the addends match in terms of numbers of rows and columns, 2 and 3 in the example here.

**Scalar multiplication:** Again, this is simply elementwise. E.g. with A as above,

$$1.5\ A = \begin{pmatrix} 7.5 & 3 & 9 \\ 1.5 & 3.9 & -1.8 \end{pmatrix}$$

**Distributive property:**

For matrices A, B and C of suitable conformability (A and B match in numbers of rows and columns, and their common number of columns matches the number of rows in C), we have

(A+B) C = AC + BC

### 2.11.2 Matrix transpose

This is a very simple but very important operation: We merely exchange rows and columns of the given matrix. For instance, with A as above, its transpose (signified with "”'), is

$$A' = \left( \begin{array}{rr} 5 & 1 \\ 2 & 2.6 \\ 6 & -1.2 \end{array} \right)$$

Some books use the notation $A^t$ or $A^T$ instead of $A'$. The R function for transpose is `t()`.

It can be shown that if $A$ and $B$ are conformable, then

$$(AB)' = B'A'$$

For some matrices $C$, we have $C' = C$. We then say that $C$ is *symmetric.*

We will often write a row vector in the form (a,b,c,...). So (5,1,88) means the 1x3 matrix with those elements. If we wish to write a column vector within some text, we use transpose, so that for instance (5,1,88)' means a 3x1 matrix.

### 2.11.3 Trace of a square matrix

The *trace* of a square matrix $A$ is the sum of its diagonal elements, $tr(A) = \sum_{i=1}^{n} A_{ii}$. This measure has various properties, some obvious (the trace of the sum of two matrices is the sum of their traces), and some less so, such as:

**Theorem 2.1.** *Suppose A and B are square matrices of the same size. Then*

$$tr(AB) = tr(BA) \tag{2.3}$$

*Proof.* See Your Turn problem below. □

And furthermore:

**Corollary 2.1.** *Trace is invariant under* circular shifts, *e.g. $UVW$, $VWU$ and $WUV$ all have the same trace.*

## 2.12 Partitioned Matrices: an Invaluable Visualization Tool

Here, "visualization" is not a reference to graphics but rather to highlighting certain submatrices.

Tip 2: A Crucial Tool

Matrix partitioning is amazingly powerful, given its utter simplicity. It compactifies matrix algebra, making complex expressions easier to visualize and discuss.
The techniques introduced in this section will be used repeatedly throughout the book. **Readers should spend extra time here,** making sure the compact representations make sense.

Specifics now follow:

### 2.12.1 How partitioning works

Consider a matrix-matrix product $MQ$. The use of partitioning works on "pretending":

Matrix Partitioning

1. Partition $M$ and $Q$ into groups of contiguous rows or columns, respectively. Let $g_m$ and $g_q$ denote the number of groups. (A group size of 1 is permissible.)
2. Enter Pretend Mode: Pretend that each group is a number. Now $M$ and $Q$ look like vectors, of lengths $g_m$ and $g_q$.
3. Go ahead and "multiply" the two "vectors."

4. Re-enter Reality Mode. Replace the elements of the "product" in step 3 by the groups' actual rows and columns.

5. Marvel at the fact that this bit of "alchemy" actually produces the correct matrix equation.

It will be easier to see how this works by considering the special case of $Q$ being a vector $v$. Of course, that means that $v$ is a column vector $(v_1, v_2, ..., v_n)'$. Say $M$ is of size $m \times n$.

Now, we ask for the reader's patience here, as we will move back and forth between a symbolic "pretend world" and reality. We promise, it will be worth it.

Let's denote column $j$ of $M$ by $c_j$. Write $M$ symbolically as

$$M = (c_1, c_2, ..., c_n)$$

so the group size in this case is 1, with $n$ groups of columns. Treating the $c_j$ as "numbers," again symbolically, $M$ looks like a $1 \times n$ matrix. We temporarily pretend it's $1 \times n$ even though it's actually $m \times n$, and that the $c_j$ are numbers, even though they are vectors. Then the "product"

$$Mv = (c_1, c_2, ..., c_n) \left( \begin{array}{c} v_1 \\ v_2 \\ ... \\ v_n \end{array} \right) = v_1 c_1 + v_2 c_2 + ... + v_n c_n$$

looks, once again symbolically like the product of $1 \times n$ and $n \times 1$ vectors, a dot product, resulting in a $1 \times 1$, i.e. a number.

The wonderful thing about all this is that if we now stop pretending, we get the right answer! If we now treat the $c_j$ for what they really are, column vectors, then the relation

$$Mv = v_1 c_1 + v_2 c_2 + ... + v_n c_n$$

$M$ is indeed equal to this sum of scalars times its columns. Let's check with a specific example.

For instance, take

$$A = \begin{pmatrix} 5 & 2 & 6 \\ 1 & 2.6 & -1.2 \end{pmatrix} \tag{2.4}$$

and

$$v = \begin{pmatrix} 10 \\ 2 \\ 1 \end{pmatrix} \tag{2.5}$$

The reader should check that

$$10 \begin{pmatrix} 5 \\ 1 \end{pmatrix} + 2 \begin{pmatrix} 2 \\ 2.6 \end{pmatrix} + 1 \begin{pmatrix} 6 \\ -1.2 \end{pmatrix} \tag{2.6}$$

so that e.g. $v_1 = 10$ and $c_1 = (5, 1)'$, does indeed work out to

$$\begin{pmatrix} 60 \\ 14 \end{pmatrix}$$

i.e. to $Av$. It works!

Note that *we choose the partitioning*; there is no inherent partition structure. In some settings, there is a structure that fits our needs, and we use that.

### 2.12.2 Linear combinations of rows and columns of a matrix

Note that the above expression Equation 2.6,

$$10 \begin{pmatrix} 5 \\ 1 \end{pmatrix} + 2 \begin{pmatrix} 2 \\ 2.6 \end{pmatrix} + 1 \begin{pmatrix} 6 \\ -1.2 \end{pmatrix},$$

is a sum of scalar products of vectors, which is called a *linear combination* of those vectors. The quantities 10, 2 and 1 are the *coefficients* in that linear combination.

In view of the fact that that linear combination worked out to be $Av$, we have that:

**Theorem 2.2.** *The product Av of a matrix times a column vector is equal to a linear combination of the columns of the matrix, with coefficients equal to the column vector.*

Similarly,

**Theorem 2.3.** *The product wA of a row vector and a matrix is equal to a linear combination of the rows of the matrix, with the coefficients coming from the row vector.*

To further illustrate all this, write the above matrix in Equation 2.4 as

$$A = \begin{pmatrix} A_{11} & A_{12} \end{pmatrix} \tag{2.7}$$

where

$$A_{11} = \begin{pmatrix} 5 & 2 \\ 1 & 2.6 \end{pmatrix}$$

and

$$A_{12} = \begin{pmatrix} 6 \\ -1.2 \end{pmatrix}$$

Symbolically, in Equation 2.7, $A$ now looks like a 1x2 "matrix." Similarly, rewriting Equation 2.5, we have

$$v = \begin{pmatrix} v_{11} \\ v_{21} \end{pmatrix}$$

where

$$v_{11} = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$$

and $v_{21} = 1$ (a 1x1 matrix); $v$ looks to be 2x1, though it is actually $3 \times 1$.

So, again pretending, treat the product $Av$ as the multiplication of a 1 x 2 "matrix" and a 2 x 1 "vector", yielding a 1 x 1 result, another "dot product,"

$$A_{11}v_{11} + A_{12}v_{21}$$

But all that pretending actually does give the correct answer!

$$A_{11}v_{11}+A_{12}v_{21} = \begin{pmatrix} 5 & 2 \\ 1 & 2.6 \end{pmatrix} \begin{pmatrix} 10 \\ 2 \end{pmatrix} + \begin{pmatrix} 6 \\ -1.2 \end{pmatrix} 1 = \begin{pmatrix} 60 \\ 14 \end{pmatrix}$$

which is the true value of $Av$.

We can extend that reasoning further. Say $A$ and $B$ are matrices of sizes $m \times n$ and $n \times k$, and consider the product $AB$. Partition $B$ by its columns,

$$B = (B^{(1)}|B^{(2)}|...|B^{(k)})$$

Now pretending that $A$ is a $1 \times 1$ "matrix" and $B$ is a $1 \times k$ "matrix", we have

$$AB = (AB^{(1)}|AB^{(2)}|...|AB^{(k)})$$

In other words, making use of Theorem 2.2,

**Theorem 2.4.** *In the product $AB$, column $j$ is a linear combination of the columns of $A$, and the coefficients in that linear combination are the elements of column $j$ of $B$.*

*A similar result holds for the rows of the product.*

## 2.13 Your Turn

**Your Turn:** Fill in the blank with a term from this chapter: The adjacency matrix of an undirected graph is necessarily ____________.

**Your Turn:** Consider the Karate Club dataset.

- Which members is member 6 linked with?
- Which member is linked to the most number of members? How about the least number?
- Find an example of a *triad*, i.e. a set of 3 members who are all linked to each other.

**Your Turn:** We say a square matrix is *upper-triangular* if its below-diagonal elements are all 0s. (There is a similar concept of *lower-triangular*.) Explain why, given two upper-triangular matrices $A$ and $B$ of the same size, the product $AB$ is also upper-triangular.

**Your Turn:** Write an R function with call form

```
outLinks(adj)
```

where **adj** is the adjacency matrix of some network graph, possibly directed. The function will return an R list, whose $i^{th}$ element is a vector of all the nodes that have exactly $i$ outgoing links.

**Your Turn:** Consider the matrix

$$A = \begin{pmatrix} 3 & -1 \\ 0 & 8 \\ 4 & 5 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Say we decide to partition it as

$$A = \begin{pmatrix} B \\ I \end{pmatrix}$$

Now consider the product $AA'$. Using partitioning, we would treat $B$ and $I$ numbers and the product as having factors of size $2 \times 1$ and $1 \times 2$. That would give us a $2 \times 2$ "matrix"

$$AA' = \begin{pmatrix} B \\ I \end{pmatrix} (B', I) = \begin{pmatrix} BB' & B \\ B' & I \end{pmatrix} \tag{2.8}$$

Evaluate $AA'$ and the far-right side of Equation 2.8 to verify that the partitioning did indeed give us the right answer.

**Your Turn:** The long-run probabilities in Section 2.6 turned out to be uniform, with value 0.20 for all five states. In fact, that is usually not the case. Make a small change to $P_1$ – remember to keep the row sums to 1 – and compute a high power to check whether the long-run distribution seems nonuniform.

**Your Turn:** Not every Markov chain, even ones with finitely many states, have long-run distributions. Some chains have *periodic* states. It may be, for instance, that after leaving state $i$, once can return only after an even number of hops. Modify our example chain here so that states 1 and 5 (and all the others) have that property. Then compute $P^n$ for various large values of $n$ and observe oscillatory behavior, rather than long-run convergence.

**Your Turn:** Consider the following Markov model of a discrete-time, single-server queue:

- Model parameters are $p$ (probability of job completion in a particular time epoch), $q$ (probability of new job arriving) and $m$ (size of the customer waiting area).
- Jobs arrive, are served (possibly after queuing) and leave.
- Only one job can be in service at a time.
- At each time epoch:
  - The job currently in service, if any, will complete with probability $p$.
  - After a job completion, a job in the queue, if any, will start service.

  - A new job will arrive with probability $q$. If the server is free, this new job will start service, rather than going to the waiting room. If the queue is not full when this job arrives, it will join the queue; otherwise, the job is discarded.

- The system is memoryless in the Markov sense. If a job has been in service for many epochs now, the probability that it finishes in the next epoch is still $p$.

- The current state is the number of jobs in the system, taking on the values 0,1,2,..,m+1; that last state means m jobs in the queue and 1 in service.

For instance, say p = 0.4, q = 0.2, m = 5, Suppose the current state is 3, so there is a job in service and two jobs in the queue. Our next state will be 2 with probability (0.4) (0.8); it will be 3 with probability (0.4) (0.2), and so on.

Analyze this system for the case given above. Find the approximate long-run distribution, and also the proportion of jobs that get discarded.

**Your Turn:** Prove Equation 2.3. Hint: Write out the left-hand side as a double sum. Reverse the order of summation, and work toward the right-hand side.

**Your Turn:** Write out the details of the "similar result" in the statement of Theorem 2.4.

# 3 Matrix Inverse

Goals of this chapter:

Central to matrix operations is the matrix *inverse*, which is somewhat analogous to reciprocals in arithmetic. This is a fundamental operation in linear algebra (though ironically, related quantities are often used instead of using the inverse directly).

To motivate our discussion of matrix inverse, we first revisit the topic of Markov chains.

## 3.1 A Further Look at Markov Chains

Suppose $X_0$, our state at time 0, is random. Let $f$ denote its distribution, i.e. its list of probabilities: $f_i = P(X_0 = i)$, i = 1,...,k, where k is the number of states in the chain. What about the distribution of $X_1$, the state at time 1? Let's find an expression for $g$, the distribution of $X_1$, in terms of $f$.

$$g_j = P(X_1 = j) = \sum_{i=1}^{k} P(X_0 = i) P(X_1 = j | X_0 = i) = \sum_{i=1}^{k} f_i a_{ij}$$

where $a_{ij}$ is the row $i$, column $j$ element of the chain's transition matrix $P$.

Putting this is more explicit matrix terms,

$$g' = (g_1, ..., g_k)' = (f_1 a_{11} + ... + f_k a_{k1}, ..., f_1 a_{1k} + ... + f_k a_{kk})$$

Setting $b_j$ to column $j$ of $P$ and using matrix partitioning, we see that that last expression is

$$(f'b_1, ..., f'b_k) = f'(b_1, ..., b_k) = f'P$$

So we have a nice compact relation for the distribution of $X_1$ in terms of the distribution of $X_0$.

$$g' = f'P$$

And setting $h$ to the distribution of $X_2$, the same reasoning gives us

$$h' = g'P$$

Let $d_r$ denote the distribution of $X_r$. Generalizing the above reasoning gives us

Note that we used the Markov property, "memorylessness." Once we reach time 1, "time starts over," regradless of the previous history, i.e. regardless of where we were at time 0.

$$d'_r = d'_{r-1}P$$

For convenience, let's take transposes (recalling that $(AB)' = B'A'$):

$$d_r = P'd_{r-1} \tag{3.1}$$

Now suppose our chain has a long-run distribution $\nu$, as in {Section 2.6}, so that

$$\lim_{r\to\infty} d_r = \nu$$

Applying this to Equation 3.1, we have

$$\nu = P'\nu \tag{3.2}$$

Since P is known, this provides us with a way to compute $\nu$. All we need to do is solve Equation 3.2. Well, how do we do that? It turns out that use of matrix inverses will solve our problem.

## 3.2 Definition

For any square matrix $A$, its *inverse* $B$ (if it exists) is a square matrix of the same size such that

$$AB = BA = I$$

where $I$ is the identity matrix of that size.

As hinted, many matrices do not have inverses. For instance, if $A$ consists of all 0s, there is no way to get $I$ from $AB$.

In very rough terms, it sometimes helps the intuition to think of an inverse as the "reciprocal" of the matrix.

We will often speak of *the* inverse of $A$. In fact, if $A$ is invertible, its inverse is unique.

## 3.3 Example: Computing Long-Run Markov Distribution

Now let us return to Equation 3.2, which expresses the vector of long-run state probabilities for a Markov chain with transition matrix P and stationary distribution $\nu$, How can we use matrix inverses to solve this equation?

Starting with

$$\nu = P'\nu \tag{3.3}$$

rewrite using the identity matrix:

$$(I - P')\nu = 0$$

For the random walk chain in Chapter 1, we had

In that particular model, P' = P, but for most chains this is not the case.

$$P = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0.5 \end{pmatrix} \tag{3.4}$$

With $\nu = (\nu_1, \nu_2, \nu_3, \nu_4, \nu_5)'$, the equation to be solved, $(I - P')\nu = \nu$, is

$$\begin{pmatrix} 0.5 & -0.5 & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 & 0 \\ 0 & -0.5 & 1 & -0.5 & 0 \\ 0 & 0 & -0.5 & 1 & -0.5 \\ 0 & 0 & 0 & -0.5 & 0.5 \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \\ \nu_4 \\ \nu_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

If we perform the matrix multiplication, we have an ordinary system of linear equations:

$$\begin{aligned} 0.5\nu_1 - 0.5\nu_2 &= 0 \\ -0.5\nu_1 + \nu_2 - 0.5\nu_3 &= 0 \\ -0.5\nu_2 + \nu_3 - 0.5\nu_4 &= 0 \\ -0.5\nu_3 + \nu_4 - 0.5\nu_5 &= 0 \\ -0.5\nu_4 + 0.5\nu_5 &= 0 \end{aligned} \tag{3.5}$$

This is high school math, and we could solve the equations that way. But this is literally what linear algebra was invented for, solving systems of equations! We will use matrix inverse.

But first, we have a problem to solve: The only solution to the above system is with all $\nu_i = 0$. We need an equation involving a nonzero quantity.

But we do have such an equation. The vector $\nu$ is a stationary distribution for a Markov chain, i.e. the set of long-run probabilities, and thus it must sum to 1.0. Let's replace the last row by that relation:

$$\begin{pmatrix} 0.5 & -0.5 & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 & 0 \\ 0 & -0.5 & 1 & -0.5 & 0 \\ 0 & 0 & -0.5 & 1 & -0.5 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \\ \nu_4 \\ \nu_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.6}$$

More compactly,

$$G\nu = q$$

If our matrix $G$ is invertible, we can premultiply both sides of our equation above, yielding

$I - P'$ might not be invertible, as there may not be a long-run distribution.

$$G^{-1}q = G^{-1}G\nu = \nu$$

So, we have obtained our solution for the stationary distribution $\nu$,

$$\nu = G^{-1}q$$

We can evaluate it numerically via the R **solve** function, which finds matrix inverse:

```
G <-
rbind(c(0.5,-0.5,0,0,0), c(-0.5,1,-0.5,0,0), c(0,-0.5,1,-0.5,0),
   c(0,0,-0.5,1,-0.5), c(1,1,1,1,1))
G
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  0.5 -0.5  0.0  0.0  0.0
[2,] -0.5  1.0 -0.5  0.0  0.0
[3,]  0.0 -0.5  1.0 -0.5  0.0
[4,]  0.0  0.0 -0.5  1.0 -0.5
[5,]  1.0  1.0  1.0  1.0  1.0
```

```
Ginv <- solve(G)
# check the inverse
Ginv %*% G  # yes, get I (of course with some roundoff error)
```

```
              [,1]         [,2]          [,3]          [,4]          [,5]
[1,]  1.000000e+00 1.942890e-16 -1.387779e-16  1.942890e-16  8.326673e-17
[2,]  1.942890e-16 1.000000e+00  3.053113e-16 -2.775558e-17  8.326673e-17
[3,] -2.775558e-17 0.000000e+00  1.000000e+00  0.000000e+00  2.775558e-17
[4,] -1.665335e-16 4.996004e-16 -3.608225e-16  1.000000e+00 -2.775558e-17
[5,] -1.665335e-16 7.216450e-16 -4.996004e-16  5.551115e-17  1.000000e+00
```

```
nu <- Ginv %*% c(0,0,0,0,1)  # recall that q = c(0,0,0,0,1)
nu
```

```
     [,1]
[1,]  0.2
[2,]  0.2
[3,]  0.2
[4,]  0.2
[5,]  0.2
```

This confirms our earlier speculation in Section 2.6 based on powers of $P$.

## 3.4 Matrix Algebra

Several properties to note:

- If the inverses of $A$ and $B$ exist, and $A$ and $B$ are conformable, then $(AB)^{-1}$ exists and is equal to $B^{-1}A^{-1}$.

  *Proof:* Consider the product $(AB)(B^{-1}A^{-1})$. The $B$ factors give us $I$, leaving $AA^{-1}$, which too is $I$.

- $(A')^{-1}$ exists and is equal to $(A^{-1})'$.

  *Proof:* Follows immediately from $AA^{-1} = I$ and the fact that $(UV)' = V'U'$.

- If $A$ is invertible and symmetric, then $(A^{-1})'$ is also symmetric.

  *Proof:* For convenience, let $B$ denote $A^{-1}$. Then

$$AB = I$$

  Again using the fact that the transpose of a product is the reverse product of the transposes, we have

$$I = I' = (AB)' = B'A'$$

  But since $A' = A$, we have

$$I = B'A$$

  In other words, not only is $B$ the inverse of $A$, $B'$ is too! So, $B = B'$.

## 3.5 Computation of the Matrix Inverse

Finding the inverse of a large matrix – in data science applications, the number of rows and columns $n$ can easily be hundreds or more – can be computationally challenging. The run time is proportional to $n^3$, and roundoff error can be an issue. Sophisticated algorithms have been developed, such as the QR decompositions. So in R, we should use, say, **qr.solve** rather than **solve** if we are working with sizable matrices, or even use methods that do not directly compute the inverse.

The classic "pencil and paper" method for matrix inversion is instructive, and will be presented here.

### 3.5.1 Pencil-and-paper computation

Note: Some readers will notice some similarity here with elementary methods they learned in high school, but actually the treatment here is much more sophisticated. It will play an important practical and theoretical role in Chapter 7.

The basic idea follows the pattern the reader learned for solving systems of linear equations, but with the added twist of involving some matrix multiplication.

Let's take as our example

$$A = \begin{pmatrix} 4 & 7 \\ -8 & 15 \end{pmatrix}$$

We aim to transform this to the $2 \times 2$ identity matrix, via a sequence of row operations.

### 3.5.2 Use of elementary matrices

Let's multiply row 1 by 1/4, to put 1 in the first element:

$$\begin{pmatrix} 1 & \frac{7}{4} \\ -8 & 15 \end{pmatrix}$$

In matrix terms, that operation is equivalent to premultiplying $A$ by

$$E_1 = \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{pmatrix}$$

(Reader: Make SURE to verify this!)

We then add 8 times row 1 to row 2, yielding

$$\begin{pmatrix} 1 & \frac{7}{4} \\ 0 & 29 \end{pmatrix}$$

The premultiplier for this operation is

$$E_2 = \begin{pmatrix} 1 & 0 \\ 8 & 1 \end{pmatrix}$$

Multiply row 2 by 1/29:

$$\begin{pmatrix} 1 & \frac{7}{4} \\ 0 & 1 \end{pmatrix}$$

corresponding to premultiplication by

$$E_3 = \begin{pmatrix} 1 & 0 \\ 8 & 1/29 \end{pmatrix}$$

And finally, add -7/4 row 2 to row 1:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The premultiplier is

$$E_4 = \begin{pmatrix} 1 & \frac{7}{4} \\ 0 & 1 \end{pmatrix}$$

Now, how does that give us $A^{-1}$? The method your were taught probably set up the partioned matrix $(A, I)$. The row operations that transformed $A$ to $I$ also transformed $I$ to $A^{-1}$. Here's why:

As noted, the row operations are such that

$$E_4 E_3 E_2 E_1 A$$

give us the final transformed result, i.e. the matrix $I$:

$$(E_4 E_3 E_2 E_1) A = I$$

Aha! We have found the inverse of $A$ – it's $E_4 E_3 E_2 E_1$.

Note too that

Recall that the inverse of a product (if it exists is the reverse product of the inverses.

$$A = (E_4 E_3 E_2 E_1)^{-1} I = E_4^{-1} E_3^{-1} E_2^{-1} E_1^{-1}$$

So apparently the inverses of the elementary matrices $E_i$ also exist, and in fact we can obtain them easily:

Each $E_i^{-1}$ simply "undoes" its partner. $E_1$, for instance, multiplies the row 1, column 1 element by 1/4, which is "undone" by multiplying that element by 4,

$$E_1^{-1} = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}$$

Also, to undo the operation of adding 8 times row 1 to row 2, we add -8 times row 1 to row 2:

$$E_2^{-1} = \begin{pmatrix} 1 & 0 \\ -8 & 1 \end{pmatrix}$$

## 3.6 Nonexistent Inverse

Suppose our matrix $A$ had been slightly different:

$$A = \begin{pmatrix} 4 & 7 \\ -8 & -14 \end{pmatrix}$$

This would have led to

$$\begin{pmatrix} 1 & \frac{7}{4} \\ 0 & 0 \end{pmatrix}$$

This cannot lead to $I$, indicating that $A^{-1}$ does not exist, and the matrix is said to be *singular*. And it's no coincidence that row 2 of $A$ is double row 1. This has many implications, as will be seen in our chapter on vector spaces.

## 3.7 Determinants

This is a topic that is quite straightforward and traditional, even old-fashioned – in fact, *too* old-fashioned, according to mathematician Sheldon Axler. The theme of his book, *Linear Algebra Done Right*, is that determinants are overemphasized. He relegates the topic to the very end of the book. Yet determinants do appear often in applied linear algebra settings. Moreover, they will be convenient to use in explaining concepts in this book on linear algebra in *Data Science*.

But why place the topic in this particular chapter? The answer lies in the fact that earlier in this chapter we had the proviso "If $(A'A)^{-1}$ exists." The following property of determinants is then relevant:

> A square matrix $G$ is invertible if and only if $\det(G) \neq 0$.

There are better ways to ascertain invertibility than this, but it is conceptually helpful. Determinants play a similar role in the topic of eigenvectors in Chapter 13.

### 3.7.1 Definition

The standard definition is one of the ugliest in all of mathematics. Instead we will define the term using one of the methods for calculating determinants.

> Consider an $r \times r$ matrix $G$. For $r = 2$, write $G$ as
>
> $$G = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$
>
> and define $\det(G)$ to be $ad - bc$. For $r > 2$, define submatrices as follows.
>
> Define $G_j$ to be the $(r-1) \times (r-1)$ submatrix obtained by removing row 1 and column $j$ from $G$. Then $\det(G)$ is defined recursively as

$$\sum_{i=1}^{r}(-1)^{i+1}\det(G_i)$$

> Say $r = 4$. Applying this formula, we find our original determinant in terms of the determinants of some $3 \times 3$ matrices, and then we apply the formula to *those* matrices. That gives us a sum of determinants of $2 \times 2$ matrices, for which we have an explicit formula.
>
> Actually, we can alternatively remove row $i$ instead of row 1. If $i$ is an odd number, the same recursive formula holds, but for even $i$, replace $(-1)^{i+1}$ by $(-1)^i$.
>
> The same rules apply if 'row' is replaced by 'column' above.

For instance, consider

$$M = \begin{pmatrix} 5 & 1 & 0 \\ 3 & -1 & 7 \\ 0 & 1 & 1 \end{pmatrix}$$

Then $\det(M) = 5(-1-7) - 1(3-0) + 0 = -43$.

*A glimpse at the classical definition:*

Using the same approach as in the last computation, we would find that the determinant of a general $3 \times 3$ matrix

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

is

$$aei + bfg + cdh - afh - bdi - ceg$$

Each term here is involves a product of 3 of the elements of the matrix. In general, the determinant involves sums and differences of permuted products of distinct elements of the matrix, as we see above. The formation of the terms in general, and the determination of + and - signs, is done in complex but precise manner that we will not present here. But the reader should at least keep in mind that each term is a product of $n$ elements of the matrix, a fact that will be relevant in the sequel.

### 3.7.2 Properties

We state these without proof:

- $G^{-1}$ exists if and only if $\det(G) \neq 0$
- $\det(GH) = \det(G)\det(H)$

## 3.8 Your Turn

**Your Turn:** In Section 3.5.2, find the inverses of $E_3$ and $E_4$ using similar reasoning, and thus find $A^{-1}$.

**Your Turn:** If the matrix $A$ has a 0 row, then $\det(A)$ must be 0. Explain why.

**Your Turn:** We say a square matrix $A = (a_{ij})$ is *upper-triangular* if its below-diagonal elements are all 0s. Give a closed-form formula for $\det(A)$ in terms of the $a_{ij}$.

**Your Turn:** In Equation 2.1, consider the variant

$$P = \left( \begin{array}{rrrrr} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 1.0 & 0.0 \end{array} \right)$$

Here, the walker at state 5 immediately "bounces back" to state 4, rather than remaining at state 5 for one or more epochs.

In the original chain, we found that $\nu = (0.2, 0.2, 0.2, 0.2, 0.2)'$. Speculate as to the effect on $\nu_5$ of the above change in model. Then investigate to determine if our speculation was correct.

**Your Turn:** Consider a Markov chain with transition probability matrix

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

This chain is termed *periodic*, with period 2. It alternates between states 1 and 2, and thus a long-run distribution $\nu$ does not exist. That would suggest that $I - P'$ is noninvertible. Confirm this.

**Your Turn:** If you are familiar with recursive calls, write a function `dt(a)` to compute the determinant of a square matrix $A$.

**Your Turn:** Prove the assertions in Section 3.4. Note that for the identity matrix $I$, $I' = I$.

**Your Turn:** The determinant of a 3 x 3 matrix

$$M = \begin{pmatrix} a & b & d \\ d & e & f \\ g & h & i \end{pmatrix}$$

is

$$aei + bfg + cdh - ceg - bdi - afh$$

Suppose the elements of $M$ are independent random variables with uniform distributions on (0,1). Argue that P(M is invertible) = 1.

# 4 Covariance Matrices, MV Normal Distribution

Goals of this chapter:

A central entity in multivariate analysis is that of the *covariance matrix*. In this chapter we define the term, list its many properties, and show its role in the multivariate analog of the normal distribution family.

## 4.1 Random Vectors

You are probably familiar with the concept of a random variable, but of even greater importance is random *vectors*.

Say we are jointly modeling height, weight, age, systolic blood pressure and cholesterol, and are especially interested in relations between these quantities. We then have the random vector

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{pmatrix} = \begin{pmatrix} \text{height} \\ \text{weight} \\ \text{age} \\ \text{bp} \\ \text{chol} \end{pmatrix}$$

### 4.1.1 Sample vs. population

We may observe $n$ realizations of $X$ in the form of sample data, say on $n = 100$ people. In the statistics world, we treat this data as a random sample from some population, say all Americans. Usually, we are just given the data rather then having actual random sampling, but this view recognizes that there are a lot more people out there than our data.

We speak of estimating population quantities. For instance, we can estimate the population value $E(X_1)$, i.e. mean of $X_1$ throughout the population, by the sample analog,

$$\frac{1}{n} \sum_{i=1}^{n} X_{1j}$$

where $X_{ij}$ denotes the value of $X_i$ for the $j^{th}$ person in our sample.

By contrast, this view is rarely taken in the machine learning community. The data is the data, and the fact that it is a small subset of a much larger group is irrelevant. They will often allude to the randomness of the data by mentioning the "data generating mechanism," but go no further.

## 4.2 Covariance

### 4.2.1 Scalar covariance

Recall first the notion in statistics of *covariance*: Given a pair of random variables $U$ and $V$, their covariance is defined by

$$Cov(U,V) = E[(U - EU)(V - EV)]$$

Loosely speaking, this measures the degree to which the two random variables vary together. Consider for instance human height $H$ and $W$. Taller people tend to also be heavier. Say we sample many people from a population. Most of those who are taller than average, i.e. $H > EH$, will also be heavier than average, $W > EW$, making $(H-EH)(W-EW) > 0$. Similarly, shorter people tend to be lighter, i.e. we often have $H < EH$ and $W < EW$, but then we still have $(H-EH)(W-EW) > 0$. So, one way or the other, usually $(H - EH)(W - EW) > 0$, and though there will be a number of exceptions, they will be rare enough so that

$E[(H - EH)(W - EW)] > 0.$

Of course, the *magnitude* of $(H - EH)(W - EW)$ plays a role too.

In other words, $Cov(H,W) > 0$. Similarly if $U$ is often large when $V$ is small, and vice versa, we will likely have $Cov(H,W) < 0$.

If this sounds like correlation to you, then your hunch is correct. Covariance will indeed later lead to the concept of correlation, but that intuition will serve us now.

Note some properties of scalar covariance.

- Symmetry:

$$Cov(U,V) = Cov(V,U)$$

- $Cov$ is bilinear:

$$Cov(aU,bV) = ab\ Cov(U,V)$$

- Variance as a special case:

$$Cov(U,U) = Var(U)$$

- Cross-product term:

$$Var(U+V) = Var(U) + Var(V) + 2Cov(U,V) \tag{4.1}$$

- "Short cut" formula:

$$Cov(U,V) = E(UV) - (EU)(EV) \tag{4.2}$$

### 4.2.2 Covariance matrices

The above was a review of the notion of covariance between two scalar random variables. We now turn to defining covariance for a random vector, which will turn out to be a matrix.

The relations between the various components of a random vector $X$ are often characterized by the *covariance matrix* of $X$, whose entries consist of scalar covariances between pairs of components of a random vector.

It is defined as follows for a $k$-component random vector $X$. The covariance matrix, denoted by $Cov(X)$, is a $k \times k$ matrix, and for $1 \leq i,j \leq k$, its row $i$, column $j$ element is

$$Cov(X)_{ij} = Cov(X_i, X_j)$$

**Warning:** The notation is somewhat overloaded. "Cov" refers both to the covariance between two random variables, say height and weight, and to the covariance matrix of a random vector. But it will always be clear from context which one is being discussed.

The matrix expression for $Cov(X)$ is

$$E[(X - EX)(X - EX)'] \tag{4.3}$$

where $EX$ is the mean vector for $X$, whose $i^{th}$ component is $E(X_i)$.

As an example, here is data on major league baseball players:

```
library(qeML)
data(mlb1)
head(mlb1)
```

```
        Position Height Weight   Age
1        Catcher     74    180 22.99
2        Catcher     74    215 34.69
3        Catcher     72    210 30.78
4  First_Baseman     72    210 35.43
5  First_Baseman     73    188 35.71
6 Second_Baseman     69    176 29.39
```

```
hwa <- mlb1[,-1]
cov(hwa)
```

```
           Height    Weight        Age
Height  5.3542814  25.61130 -0.8239233
Weight 25.6113038 433.60211 12.9110576
Age    -0.8239233  12.91106 18.6145019
```

```
cor(hwa)
```

```
            Height    Weight         Age
Height  1.00000000 0.5315393 -0.08252974
Weight  0.53153932 1.0000000  0.14371113
Age    -0.08252974 0.1437111  1.00000000
```

Again, this is sample data. We find that the *sample estimate* of the covariance between height and weight is 25.61130.

We'll be discussing more of this later, but what about that negative correlation between height and age? It's near 0, and this could be a sampling artifact, but another possibility is that in this sport, shorter players do not survive as well.

Properties of the matrix version of covariance:

- Matrix form of definition:

$$Cov(X) = E[(X - X)(X - EX)'] \tag{4.4}$$

  (Note the dimensions: $X$ is a column vector, say $k \times 1$, so $(X - X)(X - EX)'$ is $k \times k$. The expected value is then of that size as well.)

- For statistically independent random vectors $Q$ and $W$ of the same length,

$$Cov(Q + W) = Cov(Q) + Cov(W) \tag{4.5}$$

- For any nonrandom scalar $c$, and $Q$ a random vector, we have

$$Cov(cQ) = c^2 Cov(Q)$$

- Say we have a random vector $X$, of length $k$, and a nonrandom matrix $A$ of size $m \times k$. Then $AX$ is a new random vector $Y$ of $m$ components. It turns out that

$$Cov(Y) = ACov(X)A' \tag{4.6}$$

  The proof is straightforward but tedious, and will be omittted.

- $Cov(X)$ is a symmetric matrix. This follows from the symmetry of the definition.

- The diagonal elements of $Cov(X)$ are the variances of the random variables $X_i$. This follows from $Cov(U, U) = Var(U)$ for scalar $U$.
- If $X$ is a vector of length 1, i.e. a number, then

$$Cov(X) = Var(X)$$

- For any length-$k$ column vector $a$,

$$Var(a'X) = a'\ Cov(X)\ a \tag{4.7}$$

- Since variance is nonnegative, we thus see that $Cov(X)$ is *nonnegative definite*, meaning that for any length-$k$ column vector $a$

$$a'Cov(X)a \geq 0 \tag{4.8}$$

  Similarly, a matrix $Q$ is termed positive definite if

$$a'Qa > 0 \tag{4.9}$$

  unless $a = 0$.

- More generally, for any compatible constant vectors $a$ and $b$,

$$Cov(a'X, b'X) = a'Cov(X)b$$

- For any constant (i.e. nonrandom) $m \times k$ matrix $A$,

$$Cov(AX) = ACov(X)A' \tag{4.10}$$

## 4.3 The Multivariate Normal Distribution Family

The familiar "bell-shaped curve" refers to the *normal* (or *Gaussian*) family, whose densities have the form

$$\frac{1}{\sigma\sqrt{2\pi}}e^{-0.5(\frac{t-\mu}{\sigma})^2}$$

The values of $\mu$ and $\sigma$ are the mean and standard deviation. But what if we have a random vector, say of length $k$? Is there a generalized normal family?

### 4.3.1 Example: k = 2

The answer is yes. Here is an example for $k = 2$:

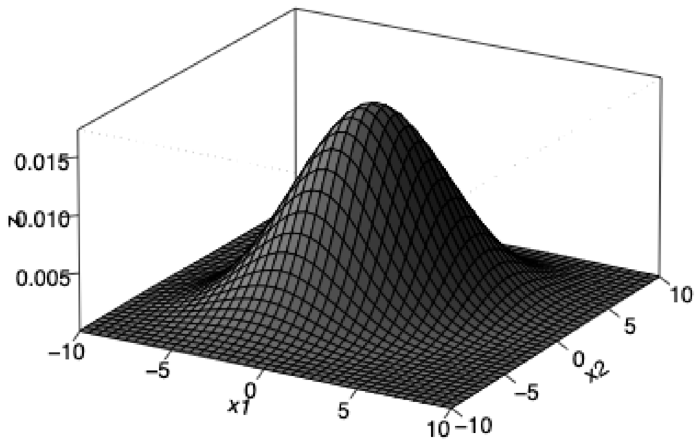


Figure 4.1: 3D bell density

### 4.3.2 General form

Well, then, what is the form of the $k$-dimensional density function? Just as the univariate normal family is parameterized by mean and variance, the multivariate one is parameterized via mean *vector* $\mu$ and covariance *matrix* $\Sigma$. The form is

$$(2\pi)^{-k/2} \det(\Sigma)^{-k/2} e^{-0.5(t-\mu)'(\Sigma)^{-1}(t-\mu)} \tag{4.11}$$

Note the intuition:

- Instead of $1/\sigma^2$, i.e. instead of dividing by variance, we "divide by $\Sigma$," intuitively viewing matrix inverse as a "reciprocal" of a matrix.

- In other words, covariance matrices operate roughly like generalized variances.
- Instead of squaring the scalar $t - \mu$, we "square" it in the vector case by peforming a $w'w$ operation, albeit with $\Sigma^{-1}$ in the middle.

Clearly, we should not stretch these analogies very far, but they do help our intuition here.

### 4.3.3 Properties

**Theorem 4.1.** *If a random vector is multivariate (MV) normally distributed, then the conditional distribution of any one of its components $Y$, given the others $X_{others} = t$ (note that $t$ is a vector if $k > 2$) has the following properties:*

- *It has a (univariate) normal distribution.*
- *Its mean $E(Y|X_{others} = t)$ is linear in $t$.*
- *Its variance $Var(Y|X_{others} = t)$ does not involve $t$.*

*These of course are the classical assumptions of linear regression models: normality, linearity and homoskedasticity. They actually come from the MV normal model.*

*More generally: Denote the mean vector and covariance matrix a random vector $X$ by $\mu$ and $\Sigma$. Partition $X$ as*

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

*and partition $\mu$ and $\Sigma$ similarly. The conditional distribution of $X_1$ given $X_2 = t$ is multivariate normal with these paramters:*

$$E(X_1|X_2 = t) = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(t - \mu_2) \tag{4.12}$$

$$Cov(X_1|X_2 = t) = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \tag{4.13}$$

*Proof.* This comes out of writing down the conditional density (overall density divided by marginal), and then doing some algebra.

□ □

Again, note the absence of $t$ in Equation 4.13.

Note too that the conditional covariance matrix, $\Sigma_{11}$ of $X_1$ given $X_2$,

$$\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

is "smaller" than the unconditional one, $\Sigma_{11}$. $X_1$ varies less when we know something about $X_2$.

Of course, this is just a rough intuitive view. One matrix is not "smaller" than another. One way to make this mathematically rigorous is to say that the matrix $B$ is smaller than (or equal to) the matrix $A$ if $A - B$ is nonnegative definite, i.e. $u'(A-B)u \geq 0$ for all vectors $u$. Since in this case $A - B$ is a covariance matrix, thus nonnegative definite (see Equation 4.8), our intuitive statement would become rigorous under this definition.

**Theorem 4.2.** *If $X$ is a multivariate-normal random vector, then so is $AX$ for any conformable nonrandom matrix $A$.*

*Proof.* Again, perform direct evaluation of the density.

□ □

**Theorem 4.3** (Cramer-Wold Theorem)**.** *A random vector $X$ has a multivariate normal distribution if and only if $w'X$ has a univariate normal distribution for all conformable nonrandom vectors $w$.*

*Proof.* "Only if" follows from above. "If" part too complex to present here.

□ □

**Theorem 4.4** (The Multivariate Central Limit Theorem)**.** *Let $X_1, X_2, ...$ be a sequence on statistically independent random vectors, with common distribution having mean vector $\mu$ and covariance matrix $\Sigma$, but not necessarily MV normal. Write*

$$\bar{X} = \frac{X_1 + ... + X_n}{n}$$

*Then the distribution of the random vector*

$$W_n = \sqrt{n}(\bar{X} - \mu)$$

*goes to multivariate normal with the 0 vector as mean and covariance matrix* $\Sigma$.

*The usual form would involve the "square root" of a matrix, but we will not discuss that concept until a later chapter.*

*Proof.* Theorem 4.3 reduces the problem to the univariate case, where we know the Central Limit Theorem holds.

□ □

## 4.4 Multinomial Random Vectors Have Approximate Multivariate Normal Distributions

Recall that a *multinomial* random vector is the mathematial analog of an R factor variable – a categorical variable with $k$ levels/categories. Just as a binomial random variable represents the number of "successes" in $n$ "trials," a multinomial random vector represents the numbers of successes in each of the $k$ categories.

To be consistent, view the binomial case as tabulating both successes and failures, e.g. both heads and tails in coin flips. Then from that viewpoint, the binomial case is multinomial with $k = 2$.

Let's write such a random vector as

$$X = \left( \begin{array}{c} N_1 \\ ... \\ N_k \end{array} \right)$$

Let $p_i$ be the probability of a trial having outcome $i = 1, ..., k$. Note the following:

- $N_1 + ... + N_k = n$
- The marginal distribution of $N_i$ is binomial, with success probability $p_i$ and $n$ trials.

So for instance if we roll a fair die 10 times, then $N_i$ is the number of trials in which the roll's outcome was $i$ dots and $p_i = 1/6$ , $i = 1, 2, 3, 4, 5, 6$.

Let's find $Cov(X)$. Define the *indicator* vector

$$I_i = \begin{pmatrix} I_{i1} \\ ... \\ I_{ik} \end{pmatrix}$$

Here $I_{ij}$ is 1 or 0, depending on whether trial $i$ resulted in category $j$. (A 1 "indicates" that caregory $j$ occurred.)

The key is that

$$X = \sum_{i=1}^{n} I_i \tag{4.14}$$

For instance, in the die-rolling example, the first component on the right-hand side is the number of rolls in which we got 1 dot, and that is by definition the same as $N_1$, the first component of $X$.

So there we have it – $X$ is a sum of independent, identically distributed random vectors, so by the Multivariate Central Limit Theorem, $X$ has an approximate multivariate normal distribution. Now, what are the mean vector and covariance matrix in that distribution?

From our discussion above, we know that

$$EX = \begin{pmatrix} np_1 \\ ... \\ np_k \end{pmatrix}$$

What about $Cov(X)$? Again, recognizing that Equation 4.14 is a sum of independent, identically distributed terms, Equation 4.5 tells us that

$$Cov(X) = Cov(\sum_{i=1}^{n} I_i) = \sum_{i=1}^{n} Cov(I_i) = nCov(I_1),$$

that last equality reflecting that the $I_i$ are identically distributed (the trials all have the same probabilistic behavior).

Now to evaluate that covariance matrix, consider two specific elements $I_{1j}$ and $I_{1m}$ of $I_1$. Recall, those elements are equal to 1 or 0, depending on whether the first trial results in Categories j and m, respectively. Then what is $Cov(I_{1j}, I_{1m})$? From Equation 4.2, we have

$$Cov(I_{1j}, I_{1m}) = E(I_{1j} I_{1m}) - (EI_{1j})(EI_{1m}) \tag{4.15}$$

Consider the two cases:

- $i = j$: Here $E(I_{1j}^2) = E(I_{1j}) = p_j$. Thus

$$Cov(I_{1j}, I_{1m}) = p_j(1 - p_j)$$

- $i \neq j$: Each $I_r$ consists of one 1 and $k - 1$ 0s. Thus $E(I_{1j} I_{1m}) = 0$ and

$$Cov(I_{1j}, I_{1m}) = -p_j p_m$$

## 4.5 Your Turn

**Your Turn:** In Equation 4.11 with $k = 2$, write $t = (t_1, t_2)$, and consider the quantity in the exponent,

$$(t - \mu)'(\Sigma)^{-1}(t - \mu)$$

Say we set this quantity to some constant $c$, then graph the locus of points $t$ in the $t_1, t_2$ plane. What geometric figure would we get?

**Your Turn:** In Theorem 4.1, suppose

$$\mu = \begin{pmatrix} 1.5 \\ 8.0 \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} 5.2 & 6.2 \\ 6.2 & 20.1 \end{pmatrix}$$

Find the regression line

$$\text{mean Y } = a + bX$$

**Your Turn:** Show that in the scalar context,

$$Cov(X,Y) = E(XY) - EX\ EY$$

**Your Turn:** Show that in the vector context,

$$Cov(X) = E(XX') - (EX)(EX)'$$

**Your Turn:** Suppose

$$W = X'\beta + \alpha S$$

for a random vector $X$, a scalar random variable $S$, a nonrandom vector $\beta$ and a nonrandom scalar $\alpha$. Show that

$$Cov(W,S) = \beta' Cov(X,S) + \alpha Var(S)$$

# 5 Linear Statistical Models

Goals of this chapter:

Here we bring together the concepts of previous chapters by presenting the *linear model*, one of the most fundamental techniques in statistics. It relies heavily on linear algebra, notably matrix inverse, but the use of linear algebra extends far beyond that.

This chapter could have been titled, say, "Optimization, Part I," since many applications of linear algebra involve minimization or maximization of some kind, and this chapter will involve calculus derivatives. But the statistical applications of linear algebra are equally important.

## 5.1 Linear Regression through the Origin

Let's consider the **Nile** dataset built-in to R on the height of the Nile River. It is a time series, one measurement per year. We will predict each year's value from the previous one.

```
head(Nile)
```

```
[1] 1120 1160  963 1210 1160 1160
```

```
n1 <- Nile[-(length(Nile))]
head(n1)
```

```
[1] 1120 1160  963 1210 1160 1160
```

```
# we will need a lag-1 version of the data
n2 <- Nile[-1]
head(n2)
```

```
[1] 1160  963 1210 1160 1160  813
```

Calling

```r
plot(n1,n2,cex=0.4,xlim=c(0,1400),yli=c(0,1400))
```

gives us

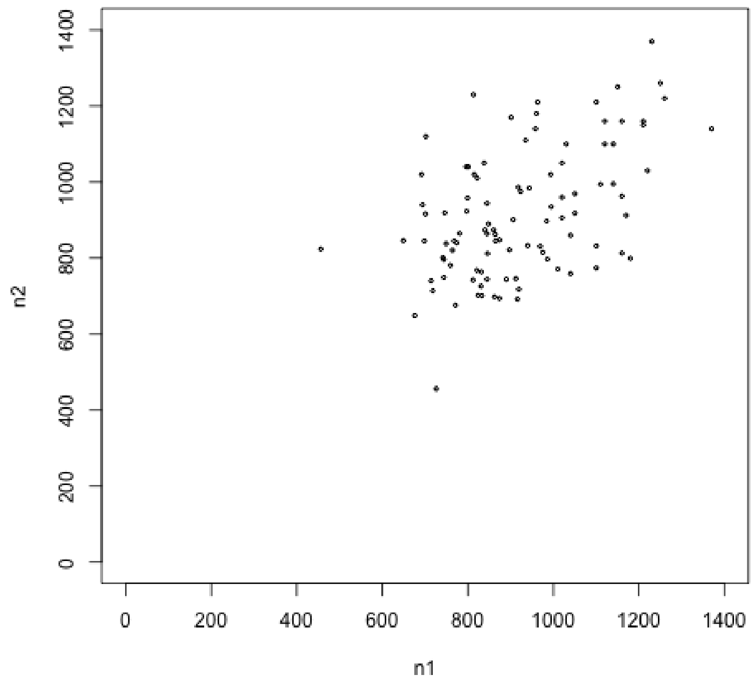


Figure 5.1: lagged Nile

We would like to fit a straight line through that data point cloud. We might have two motivations for doing this:

- The line might serve as nice summary of the data.
- More formally, let $C$ and $V$ denote the current and previous year's measurements. Then the model

$$E(C|V) = \beta V$$

  may be useful. Here the slope $\beta$ is an unknown value to be estimated from the data.

Readers with some background in linear regression models should note that this assumption of a linear trend through the origin is the only assumption we are making here. Nothing on normal distributions etc.

❗ Model Validity

The great statistician George Box once said, “All models are wrong but some are useful.” All data scientists should keep this at the forefronts of their minds.

### 5.1.1 Least squares approach

We wish to estimate $\beta$ from our data, which we regard as a sample from the population/data generating process. Denote our data by $(C_i, V_i), i = 1, ..., 100$. Let $\widehat{\beta}$ denote our estimate. How should we obtain it?

As noted in Section 4.1.1, this is something all data scientists should keep at the forefronts of their minds. We are always working with sample data, subject to intersample variations. The quantity $\widehat{\beta}$ is a random variable.

The “hat” notation ‘ˆ’ is statistical convention for “estimate of”; $\widehat{\beta}$ is our estimate of the unknown parameter $\beta$. Make sure to distinguish between the two!

Pretend for a moment that we don’t know, say, $C_{28}$. Using our estimated $\beta$, our predicted value would be $\hat{\beta} V_{28}$. Our squared prediction error would then be $(C_{28} - \hat{\beta} W_{28})^2$.

Well, we actually do know $C_{28}$ (and the others in our data), so we can answer the question:

In our search for a good value of $\hat{\beta}$, we can ask how well we would predict our known data, using that candidate value of $\hat{\beta}$ in our data. Our total squared prediction error would be

$$\sum_{i=1}^{100} [C_i - \hat{\beta} V_i]^2$$

A natural choice for $\hat{\beta}$ would be the value that minimizes this quantity.

Why not look at the absolute value instead of the square? The latter makes the math flow well, as will be seen shortly.

### 5.1.2 Calculation

As noted, our choice for $\hat{\beta}$ will be the minimizer of

$$\sum_{i=1}^{100} (C_i - bV_i)^2$$

over all possible values of $b$. We then set $\hat{\beta}$ to that minimizing value of $b$.

This is a straightforward calculus problem. Setting

$$0 = \frac{d}{db} \sum_{i=1}^{100} (C_i - bV_i)^2 = -2 \sum_{i=1}^{100} (C_i - bV_i) V_i$$

and solving $b$, we find that

$$b = \frac{\sum_{i=1}^n C_i V_i}{\sum_{i=1}^n V_i^2}$$

### 5.1.3 R code

```r
lm(n2 ~ n1-1)
```

```

Call:
lm(formula = n2 ~ n1 - 1)

Coefficients:
  n1
0.98
```

This says, "Fit the model $E(C|V) = \beta V$ to the data, with the line constrained to pass through the origin." The constraint is specified by the -1 term.

We see that the estimated regression line is

$$\widehat{E}(C|V) = 0.98V$$

## 5.2 Linear Regression Model with Intercept Term

Say we do not want to constrain the model to pass the line through the origin. Our model is then

In most applications, we do not assume such a constraint, and thus do include a $\beta_0$ term in our model.

$$E(C|V) = \beta_0 + \beta_1 V$$

where we now have two unknown parameters to be estimated.

### 5.2.1 Least-squares estimation, single predictor

Our sum of squared prediction errors is now

$$\sum_{i=1}^{100} [C_i - (b_0 + b_1 V_i)]^2$$

This means setting two derivatives to 0 and solving. Since the derivatives involve two different quantities to be optimized, $b_0$ and $b_1$, the derivatives are termed *partial*, and the $\partial$ symbol is used instead of ‘d’.

$$\begin{aligned} 0 &= \frac{\partial}{\partial b_0} \sum_{i=1}^{100} [C_i - (b_0 + b_1 V_i)]^2 && (5.1) \\ &= -2 \sum_{i=1}^{100} [C_i - (b_0 + b_1 V_i)] && (5.2) \end{aligned}$$

and

$$\begin{aligned} 0 &= \frac{\partial}{\partial b_1} \sum_{i=1}^{100} [C_i - (b_0 + b_1 V_i)]^2 && (5.3) \\ &= -2 \sum_{i=1}^{100} [C_i - (b_0 + b_1 V_i)] V_i && (5.4) \end{aligned}$$

We could then solve for the $b_i$, but let’s go straight to the general case.

## 5.3 Least-Squares Estimation, General Number of Predictors

### 5.3.1 Nile example

As we have seen, systems of linear equations are natural applications of linear algebra. The equations setting the derivatives to 0 can be written in matrix terms as

$$\begin{pmatrix} \sum_{i=1}^n C_i \\ \sum_{i=1}^n C_i V_i \end{pmatrix} = \begin{pmatrix} 100 & \sum_{i=1}^n V_i \\ \sum_{i=1}^n V_i & \sum_{i=1}^n V_i^2 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \tag{5.5}$$

Actually, that matrix equation can be derived more easily by using matrices to begin with:

Define $S$ and $T$:

$$S = \begin{pmatrix} C_1 \\ C_2 \\ ... \\ C_{100} \end{pmatrix}$$

and

$$T = \begin{pmatrix} V_1 \\ V_2 \\ ... \\ V_{100} \end{pmatrix}$$

Then our linear assumption, $E(C|V) = \beta_0 + \beta_1 V$, applied to $S$ and $T$, is

$$E(S|T) = A\beta$$

where

$$A = \begin{pmatrix} 1 & V_1 \\ 1 & V_2 \\ ... & ... \\ 1 & V_{100} \end{pmatrix} \tag{5.6}$$

and $\beta = (\beta_0, \beta_1)'$.

Our predicted error vector, using our candidate estimate $b$ of $\beta$, is very simply expressed:

$$S - Ab$$

And since for any column vector $u$, the sum of its squared elements is

$$u'u$$

our sum of squared prediction errors is

$$(S - Ab)'(S - Ab) \tag{5.7}$$

Now how we will minimize that matrix expression with respect to the vector $b$? That is the subject of the next sections.

Note: Our use of the letters 'A' and 'S' here is nonstandard; 'X' and 'Y' are the typical choices. However, a major problem is that 'X' and 'Y' are then used for other separate quantities, causing confusion, hence our use of 'A' and 'S'. We will sometimes use X and Y with quotation marks, e.g. "X" and "Y" in informal comments, to indicate our predictor variables and the quantity to be predicted. For instance, 'Say we have "X" and "Y" as human height and weight' means 'Say we wish to predict human weight from height.'

### 5.3.2 General setting

We consider the general linear regression setting in which we have $p$ predictor variables $V^{(1)}, ..., V^{(p)}$ with

$$E(C|V^{(1)} = t_1, ..., V^{(p)} = t_p) = \beta_0 + \beta_1 t_1 + ... + \beta_p t_p$$

Say we have $n$ data points. Denote the value of $V^{(j)}$ in the $i^{th}$ data point by $a_{i,j}$, and write

$$A = \begin{pmatrix} 1 & a_{11} & ... & a_{1p} \\ ... & ... & ... & ... \\ 1 & a_{n1} & ... & a_{np} \end{pmatrix}$$

and

$$E(S|A) = A\beta$$

Note the column of 1s, needed to pick up the $\beta_0$ term.

As before $S$ is the vector of associated "Y" values.

### 5.3.3 Matrix derivatives

The (column) vector of partial derivatives of a scalar quantity is called the *gradient* of that quantity. For instance, with

$$u = 2x + 3y^2 + xy$$

we have that its gradient is

$$\left( \begin{array}{c} 2+y \\ 6y+x \end{array} \right)$$

With care, we can compute gradients entirely at the matrix level, using easily derivable properties, without ever resorting to returning to the scalar expressions. Let's apply them to the case at hand in the last section,

$$(S - Ab)'(S - Ab) \tag{5.8}$$

### 5.3.4 Differentiation purely in matrix terms

It can be shown that for a column vector $a$,

$$\frac{d}{da} a'a = 2a \tag{5.9}$$

Equation 10.2 is indeed of the form $a'a$, with $a = S - Ab$, but the problem here is that $a$ in turn is a function of $b$, This calls for the Chain Rule, which does exist at the matrix level:

For example if $u = Mv + w$, with $M$ and $w$ constants (i.e. not functions of $v$, then

$$\frac{d}{dv} u'u = 2M'u$$

We must keep in mind that we are working with vectors and matrices, so that $M'u$, say $r \times s$ times $s \times 1$, is conformable matrix multiplication.

In our case at hand, we have $M = -A$ and $w = S$, so that

$$\frac{d}{db}[(S - Ab)'(S - Ab)] = -2A'(S - Ab)$$

So, set

$$0 = A'(S - Ab) = A'S - A'Ab$$

yield our minimizing $b$:

$$\hat{\beta} = (A'A)^{-1}A'S \tag{5.10}$$

providing the inverse exists (more on this in later chapters).

Let's check this with the Nile example:

```
A <- cbind(1,n1)
S <- n2
Ap <- t(A)  # R matrix transpose
solve(Ap %*% A) %*% Ap %*% S  # R matrix inverse
```

```
          [,1]
   452.7667508
n1   0.5043159
```

```
# check via R
lm(n2 ~ n1)
```

```
Call:
lm(formula = n2 ~ n1)

Coefficients:
(Intercept)           n1
   452.7668       0.5043
```

Also,

```
det(Ap %*% A)
```

```
[1] 277463876
```

Nonzero! So $(A'A)^{-1}$ does exist, as we saw.

### 5.3.5 The general case

Say our data consists of $n$ points, each of which is of length $p$. Write the $j^{th}$ element of the $i^{th}$ data point as $X_{ij}$. Then set

$$A = \begin{pmatrix} 1 & X_{11} & ... & X_{p1} \\ 1 & X_{21} & ... & X_{p2} \\ ... & ... & ... & ... \\ 1 & X_{n1} & ... & X_{np} \end{pmatrix} \tag{5.11}$$

Continue to set $S$ to the length-$n$ column vector of our response variable. Our model is

$$E(S|A) = A\beta$$

for an unknown vector $\beta$ of length $p+1$. Our estimated of that vector based on our data will be denoted

$$b = (b_0, b_1, ..., b_p)'$$

Then, using the same reasoning as before, we have the minimizing value of $b$:

$$\hat{\beta} = (A'A)^{-1}A'S \qquad (5.12)$$

again providing that the inverse exists.

### 5.3.6 Example: mlb1 data

As an example, let's take the **mlb1** from my qeML ('Quick and Easy Machine Learning package. The data is on major league baseball players. We will predict weight from height and age.

Dataset kindly provided by the UCLA Dept. of Statistics

```
library(qeML)
data(mlb1)
head(mlb1)
```

```
        Position Height Weight   Age
1        Catcher     74    180 22.99
2        Catcher     74    215 34.69
3        Catcher     72    210 30.78
4   First_Baseman     72    210 35.43
5   First_Baseman     73    188 35.71
6 Second_Baseman     69    176 29.39
```

```
ourData <- as.matrix(mlb1[,-1]) # must have matrix to enable %*%
head(ourData)
```

```
  Height Weight   Age
1     74    180 22.99
2     74    215 34.69
3     72    210 30.78
4     72    210 35.43
5     73    188 35.71
6     69    176 29.39
```

```
A <- cbind(1,ourData[,c(1,3)])
Ap <- t(A)
S <- as.vector(mlb1[,3])
solve(Ap %*% A) %*% Ap %*% S
```

```
                 [,1]
       -187.6381754
Height    4.9235994
Age       0.9115326
```

```r
# check via R
lm(Weight ~ .,data=mlb1[,-1])
```

```
Call:
lm(formula = Weight ~ ., data = mlb1[, -1])

Coefficients:
(Intercept)       Height          Age
  -187.6382       4.9236       0.9115
```

So, if we have a player of known height and age, we would predict the weight to be

-187.6382 + 4.9236 x height + 0.9115 x age

### 5.3.7 Homogeneous variance case

In addition to

$$E(S|A) = A\beta$$

$$Cov(S|A) = \sigma^2 I$$

where $\sigma$ is an unknown constant to be estimated from the data. This is called the *homoskedasticity* assumption.

So $Var(S_i) = \sigma^2$ for all $i$. It is usually not a realistic assumption. Say for instance we are predicting human weight from height. There should be more variation in weight among taller people than above shorter people. But it's a simplifying assumption, so it is commonly used.

A variation called the *sandwich* estimator allows for nonhomogeneous and unknown $Cov(S|A)$. See the package **sandwich**.

And in that setting, our formulas from Chapter 4 come in handy, as follows.

Recall that $\widehat{\beta}$ is our estimate of the unknown population parameter $\beta$, based on our random sample data. But that means that $\widehat{\beta}$ is a random vector, and thus has a covariance matrix. Using Equation 4.6 with $R = (A'A)^{-1}$, and recalling the properties of transpose, we have

Fixed-X vs. random-X settings: In some applications, $A$ is actually chosen by an experimenter, so that it is not random, the so-called *fixed-X* setting. For instance, say we are predicting children's height from age. Under the random-X approach, we would simply sample $n$ children at random from the population of interest. That would result in both age and height data being random. Under the fixed-X paradigm, we would preset ages of interest, say 2,4,...,16, and choose kids at random from those specific age groups. Here the heights would be random but the ages would be nonrandom. But even in the random case (*random-X* setting), it is standard to condition on $A$. When we derive confidence intervals in Chapter 6, the distinction won't matter. By Equation 10.5, a conditional confidence interval, say, at the 95% level also has that level unconditionally.

$$
\begin{aligned}
Cov(\widehat{\beta}) &= Cov(RA'S) && (5.13)\\
&= RA'\sigma^2 IAR' && (5.14)\\
&= \sigma^2 (A'A)^{-1} && (5.15)
\end{aligned}
$$

Classical statistical formulas use this relation to find standard errors for $\widehat{\beta}_i$ etc., to be presented in Chapter 6.

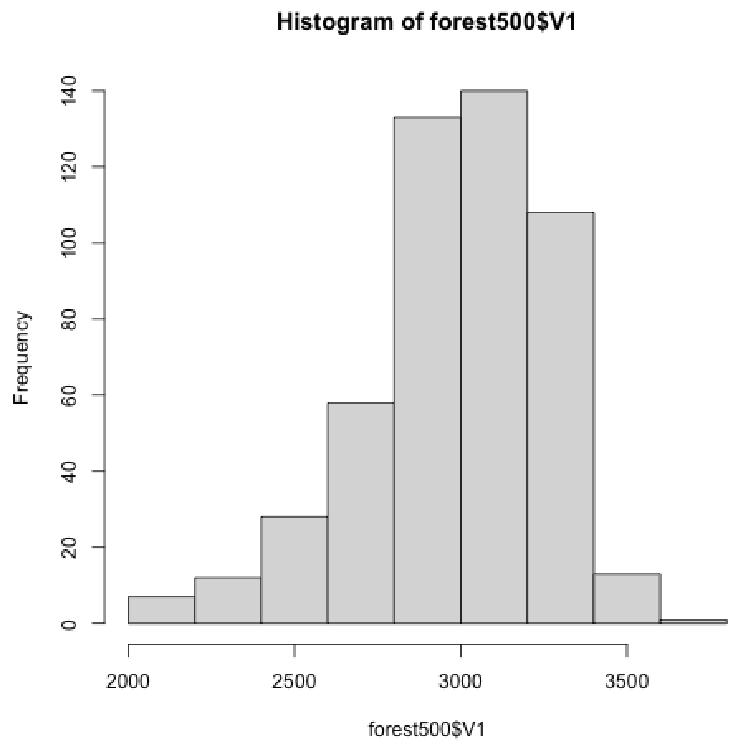
## 5.4 Update Formulas

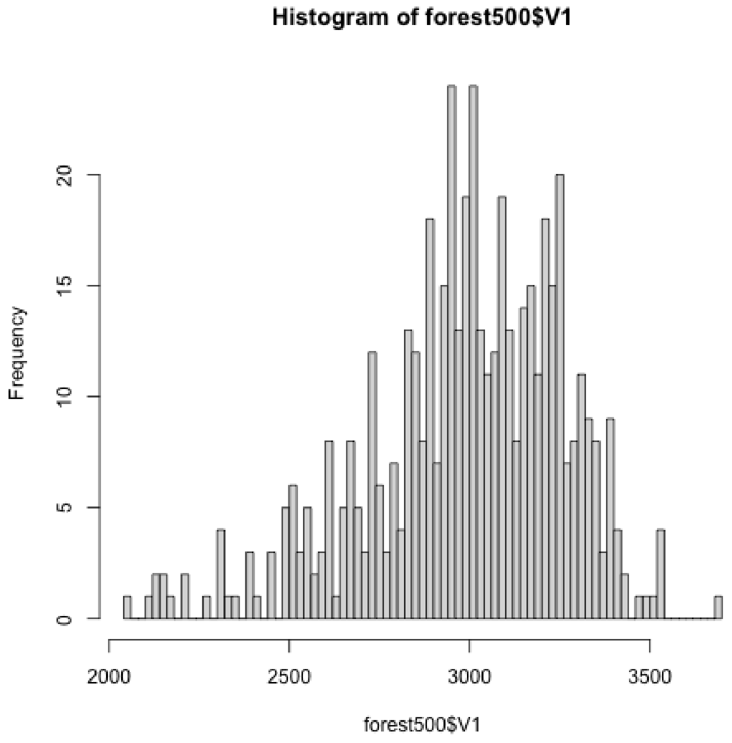
One important theme in developing prediction models (linear regression, neural networks etc.) is the avoidance of *overfitting*, meaning that we fit an overly elaborate model to our data. We simply are estimating too many things for the amount of data we have, “spreading our data too thin.”

A common example is using too many predictor variables, so that, e.g. in the linear model case, we are estimating a large number of coefficients $\beta_i$.

Or we may draw a histogram with too many bins:

```r
library(qeML)
data(forest500)  # data on forest ground cover
hist(forest500$V1,breaks=10)
hist(forest500$V1,breaks=100)
```


Histogram of forest500$V1
Frequency
0
20
40
60
80
100
120
140
2000
2500
3000
3500
forest500$V1



In a histogram, we are estimating the heights of the bins. With 10 bins we obtained a smooth graph, but with 100 bins it became choppy. So again, we are estimating too many things, given the capacity of the data.

A histogram is an estimate of the probability density function of the observed random variable. Having more, thus narrower, bins reduces estimation bias but increases estimation variance.

With larger datasets, we can use more predictor variables, more histogram bins, and so on. The question then arises is, for instance, *How many* predictors, *how many* bins and so on, can we afford to use with our given data?

The typical solution is to fit several models of different complexity, then choose the one that predicts the best. But we must do this evaluation on "fresh" data; we should not predict on the same data on which we fitted our model.

We thus rely on partitioning our data, into a *training set* to which we fit our model, and a *test set*, on which we predict using the fitted model. We may wish to do this several times.

A special case is the Leaving One Out method, in which the test set size is 1. It might go like this, for a dataset d:

```
sumErrs = 0
for i = 1,...,n  # dataset has n datapoints
   fit lm to d[-i,]
   use result to predict d[i,]
   add prediction error to sumErrs
return sumErrs
```

This can become computationally challenging, as we would need to refit the model each time. Each call to **lm** involves a matrix inversion (or equivalent), and we must do this $n$ times.

It would be nice if we could have an "update" formula that would quickly recalculate the model found on the full dataset. Then we would need to perform matrix inversion just once. In the case of linear models, such a formula exists, in the Sherman-Morrison-Woodbury relation:

Given an invertible matrix $B$ and row vectors $u$ and $v$ having lengths equal to the number of columns of $B$. form the matrix

$$C = B + uv'$$

Then $C^{-1}$ exists and is equal to

Note that the quantity $uv'$ is a square matrix the size of $B$, so the sum and product make sense.

$$B^{-1} - \frac{1}{1 + v'B^{-1}u} B^{-1}(uv')B^{-1}$$

Now, how can we apply this to the Leave One Out method? In the matrix $A$ in Equation Equation 5.11, we wish to remove row $i$; call the result $A_{-i}$. Our new version of $A'A$ is then

$$A'_{-i}A_{-i}$$

So our main task is to obtain

$$(A'_{-i}A_{-i})^{-1}$$

by updating $(A'A)^{-1}$, which we already have from our computation in the full dataset.

We can do this as follows. Denote row $i$ of $A$ by $a_i$, and set

$$u = -a_i, v = a_i$$

To show why these choices for $u$ and $v$ work, consider the case in which we delete the last row of $A$. (The analysis would be similar for other cases.) Write the latter as a partitioned matrix,

$$\begin{pmatrix} A_{(-n)} \\ a_n \end{pmatrix}$$

We pretend it is a $2 \times 1$ "matrix," and $A'$ is then "$1 \times 2$":

$$A' = (A'_{(-n)} | a'_n)$$

Thus

$$A'A = A'_{-i} A_{-i} + a_n a'_n$$

yielding

$$A'_{-i} A_{-i} = A'A - a_n a'_n$$

just what we need for Sherman-Morrison-Woodbury: With $B = A'A$, we have

$$[A'_{-i} A_{-i}]^{-1} = B^{-1} + \frac{1}{1 - a'_i B^{-1} a_i} B^{-1}(a_i a'_i) B^{-1}$$

Let's check it:

```
a <- rbind(c(1,3,2),c(1,0,5),c(1,1,1),c(1,9,-3))
apa <- t(a) %*% a
apai <- solve(apa)
a2 <- a[-2,]
apa2 <- t(a2) %*% a2
apa2i <- solve(apa2)
# prepare for S-M-W
adel <- matrix(a[2,],ncol=1)
w1 <- 1/(1 - t(adel) %*% apai %*% adel)
w1 <- as.numeric(w1)
uvt <- adel %*% t(adel)
w2 <- apai %*% uvt %*% apai
# S-M-W says this will be apa2i
apai + w1 * w2
```

```
           [,1]       [,2]      [,3]
[1,]  3.4140625 -0.7109375 -1.015625
[2,] -0.7109375  0.1640625  0.234375
[3,] -1.0156250  0.2343750  0.406250
```

```
apa2i
```

```
           [,1]       [,2]      [,3]
[1,]  3.4140625 -0.7109375 -1.015625
[2,] -0.7109375  0.1640625  0.234375
[3,] -1.0156250  0.2343750  0.406250
```

## 5.5 Generalized Linear Models

These are nonlinear models that have a “linear” component to them. The best known is the *logistic* (or *logit*) model. Here our “Y” is a binary variable, $Y = 0, 1$.

Note first that now

$$\begin{aligned} E(Y|X=t) &= 1 \cdot P(Y=1|X=t) + 0 \cdot P(Y=0|X=t) && (5.16) \\ &= P(Y=1|X=t) && (5.17) \\ & && (5.18) \end{aligned}$$

So the conditional mean now reduces to a conditional probability. This is very useful, e.g. finding the probability that a patient has a certain disease, given test results and symptoms.

So, what about the "linear component"? Here is the model:

$$P(Y=1|x=t) = \frac{1}{1+\exp\left[-\beta'\tilde{t}\right]}$$

where as usual, $\beta = (\beta_0, \beta_1, ..., \beta_p)'$. The quantity $\tilde{t}$ is

$$\begin{pmatrix} 1 \\ t \end{pmatrix}$$

and the 1 is there to pick up the $\beta_0$ term. So there is the linear component, $\beta'\tilde{t}$.

Least-squares computation doesn't work here, though some modifications do, e.g. *Iteratively Reweighted Least Squares*. This topic is beyond the scope of this book.

### 5.5.1 Example: census data

This is data for Silicon Valley programmers and engineers in the 2000 Census.

Let's predict female gender:

```
data(svcensus)
head(svcensus)
```

```
       age     educ occ wageinc wkswrkd gender
1 50.30082 zzzOther 102   75000      52 female
2 41.10139 zzzOther 101   12300      20   male
3 24.67374 zzzOther 102   15400      52 female
4 50.19951 zzzOther 100       0      52   male
5 51.18112 zzzOther 100     160       1 female
6 57.70413 zzzOther 100       0       0   male
```

```
svcensus$gender <- as.numeric(svcensus$gender == 'female')  # Y=0,1
head(svcensus)
```

```
       age     educ occ wageinc wkswrkd gender
1 50.30082 zzzOther 102   75000      52      1
2 41.10139 zzzOther 101   12300      20      0
3 24.67374 zzzOther 102   15400      52      1
4 50.19951 zzzOther 100       0      52      0
5 51.18112 zzzOther 100     160       1      1
6 57.70413 zzzOther 100       0       0      0
```

```
# for simplicity, omit a couple of the columns
glmOut <- glm(gender ~ .,data=svcensus[,-c(2,5)],family=binomial)
coef(glmOut)
```

```
  (Intercept)           age        occ101        occ102        occ106
-6.039075e-01  4.992302e-03 -3.593655e-01 -3.677021e-01  3.938857e-01
       occ140        occ141       wageinc
-8.875065e-01 -1.447660e+00 -6.063449e-06
```

The ‘binomial’ value here specifies the logistic model. Others are available, e.g. *Poisson regression.*

The direction of influence on “Y” is the same as in the linear case: A positive coefficient means, holding other predictor variables fixed, larger values of this variable produce larger probabilities for $Y = 1$. The most impactful predictors seem to be occupations 140 and 141, each of which reduces the probability of female.

We of course must form confidence intervals for a careful analysis, the topic of Chapter 6.

It should be noted, though, that the amount of impact depends on the values of the other pedictors, unlike the linear case. A popular interpretation of the logit model is the *log-odds* view, which forces focus on the linear component:

$$\log \left[ \frac{P(Y=1|X=t)}{P(Y=0|X=t)} \right] = \beta' \widetilde{t}$$

However, I have never felt this is helpful. Why on Earth should we be interested in the log-odds value? And it detracts from the focus of logit modeling a probability, of central importance.

## 5.6 Centering and Scaling One's Data

You often hear data analysts speak of *centering and scaling* their data. What does it mean, and why is useful?

- Centering a variable means to subtract its mean. The new version now has mean 0.
- Scaling a variable means to divide by its standard deviation. The new version now has standard deviation 1.0.

Both operations make variables more comparable, especially scaling. If say two predictor variables are of different sizes, say Height and Age in our baseball data above, it's hard to compare their $\widehat{\beta}_i$ values. Scaling them makes things more comparable.

As to centering, this can reduce roundoff error in computations, and anyway it seems questionable to scale but not center.

Note that if we center not only the "X" variables but also "Y," it can be shown this is equivalent to setting a model without an intercept term $\beta_0$. This often simplifies matters in various ways. Note that in this setting, we do not include a 1s column in the $A$ matrix.

The R function **scale** does centering and/or scaling.

## 5.7 Your Turn

**Your Turn:** Show Equation 5.9. Write $a = (a_1, ..., a_k)'$ and find the gradient "by hand." Compare to $2a$.

**Your Turn:** Show that

$$\frac{d}{du} u'Qu = 2Qu$$

for a constant symmetric matrix $Q$ and a vector $u$. ($u'Qu$ is called a *quadratic form.*)

# 6 Confidence Sets

Goals of this chapter:

The reader is probably already familiar with the concept of a *confidence interval* (CI). But that is just in one dimension. Confidence intervals are even more useful in the multidimensional setting. In this chapter, we introduce a fairly general method of forming a CI from multivariate data, and then extend the notion to *confidence sets* for unknown multivariate quantities themselves. As the reader may have guessed, the word "multivariate" here is a sign that linear algebra is involved, which will indeed be the case.

## 6.1 Review: Confidence Intervals, Standard Errors

To set the stage, let's review the statistical concepts of *confidence interval* and *standard error*. Say we have an estimator $\hat{\theta}$ of some population parameter $\theta$, e.g. $\bar{X}$ for a population mean $\mu$.

Loosely speaking, the term *standard error* of is our estimate of $\sqrt{Var(\hat{\theta})}$. More precisely, suppose that $\hat{\theta}$ is asymptotically normal. The standard error is an estimate of the standard deviation of that normal distribution. For this reason, it is customary to write $AVar(\hat{\theta})$ rather than $Var(\hat{\theta})$. Similarly we use $ACov$ rather than $Cov$.

Recall the Warning in Section 4.2.2. Just as Cov can mean either the covariance between two random variables or the covariance matrix of a random vector, the analogous statement holds for ACov.

This can be used to form a confidence interval (see below), but also stands on its own as an indication of the accuracy of $\hat{\theta}$.

A, say 95%, confidence interval (CI) for $\mu$ is then

$$\hat{\theta} \pm 1.96 \text{ SE}(\hat{\theta})$$

where we denote the standard error of $\hat{\theta}$ by $\text{SE}(\hat{\theta})$.

The 95% figure means that of all possible samples of the given size from the population, 95% of the resulting confidence intervals will contain $\theta$. In many cases, the 95% figure is only approximate, stemming from a derivation that uses the Central Limit Theorem.

In general, for confidence level $1 - \alpha$ replace 1.96 by $z_\alpha$, the $1 - \alpha/2$ quantile of the N(0,1) distribution, Then our CI is

$$\hat{\theta} \pm z_\alpha \mathrm{SE}(\hat{\theta}) \tag{6.1}$$

❗ Note Regarding Sensitivity of Phrasing

There is a bit of drama in this word *contain* in the phrase "will contain $\theta$." Instead of saying the intervals *contain* $\theta$, why not simply say $\theta$ is *in* the intervals? Aren't these two descriptions equivalent in terms of English? Of course they are.
But many instructors of statistics classes worry that students will take the description based on "in" to mean that $\theta$ is the random quantity, when in fact the CI is random (random center, random radius) and $\theta$ is fixed (though unknown). The instructors thus insist on the more awkward phrasing "contain," so as to avoid students misunderstanding. Indeed some instructors would contend that use of the word *in* is itself just plain incorrect.
My own view is that in some cases the word *in* is clearer (and certainly correct in any case), and that it is better to add a warning about what is random/nonrandom than engage in awkward phrasing.

## 6.2 The R vcov Function

So, how does one obtain standard errors? In R, in many cases, they will be provided by the **summary** function, but we may need an entire estimated covariance matrix rather than just a standard error.

Due to the Multivariate Central Limit Theorem, many common statistical estimators have approximately normal distributions. In R, functions such as **lm**, **glm**, **lme** and **coxph** come with an associated function **vcov**. This gives the approximate covariance matrix of the computed estimator, e.g. for the estimated beta coefficients vector in a linear model. This enables formation of approximate confidence intervals for not only individual model parameters but also linear combinations of them, as well as computing other quantities related to confidence sets.

The **vcov** function is an R *generic* function, playing a similar role to **print**, **plot**, **summary** and so on. Many R statistical functions have this for their output, including say **glm**. When we make the call **vcov(glmOut)**, the R interpreter sees that **glmOut** is of class "glm" and thus transfers the call to the class-specific function, **vcov.glm(glmOut)**. The function **coef**, also used here, is also generic.

### 6.2.1 Example: Iranian churn data

Here we use a logistic mode to predict whether a telecom customer will move to another provider. We obtain $AVar(\hat{\beta})$ using the **vcov** function:

```
data(IranianChurn)
glmOut <- glm(Exited ~ ., data = iranChurn, family = binomial)
acov <- vcov(glmOut)
acov
```

```
                  (Intercept)   CreditScore GeographyGermany GeographySpain
(Intercept)       5.993100e-02 -5.051370e-05    -1.589878e-04  -1.388436e-03
CreditScore      -5.051370e-05  7.859301e-08    -2.683572e-07  -2.454056e-07
GeographyGermany -1.589878e-04 -2.683572e-07     4.579770e-03   1.678887e-03
GeographySpain   -1.388436e-03 -2.454056e-07     1.678887e-03   4.989715e-03
GenderMale       -1.350817e-03  2.156023e-07     2.008595e-05  -2.200094e-05
Age              -2.685656e-04 -7.969688e-09     6.042109e-06  -2.107579e-06
Tenure           -4.049072e-04  1.614151e-09    -6.645863e-06   3.828636e-06
Balance          -2.936332e-08  1.033099e-13    -1.358788e-08  -2.156656e-11
NumOfProducts    -3.775016e-03 -8.271320e-08    -3.655500e-04  -3.357316e-05
HasCrCard1       -2.595873e-03  2.066409e-07    -7.611739e-05   3.603876e-05
IsActiveMember1   5.010389e-04 -3.102862e-07    -1.008754e-04  -3.954646e-05
EstimatedSalary  -2.301952e-08  1.070775e-12    -7.124867e-11  -8.778366e-11
                   GenderMale           Age        Tenure       Balance
(Intercept)      -1.350817e-03 -2.685656e-04 -4.049072e-04 -2.936332e-08
CreditScore       2.156023e-07 -7.969688e-09  1.614151e-09  1.033099e-13
GeographyGermany  2.008595e-05  6.042109e-06 -6.645863e-06 -1.358788e-08
GeographySpain   -2.200094e-05 -2.107579e-06  3.828636e-06 -2.156656e-11
GenderMale        2.968982e-03 -4.697135e-06 -7.714322e-06 -9.609423e-10
Age              -4.697135e-06  6.633235e-06 -2.330915e-07  4.769766e-11
Tenure           -7.714322e-06 -2.330915e-07  8.751351e-05 -1.419504e-11
Balance          -9.609423e-10  4.769766e-11 -1.419504e-11  2.644146e-13
```

```
NumOfProducts     5.853316e-05  2.887014e-06 -2.575910e-06  6.486034e-09
HasCrCard1       -8.288460e-06  1.405830e-07 -1.356629e-05  6.276417e-10
IsActiveMember1   2.637202e-05 -3.924088e-05  1.978922e-05 -5.129968e-10
EstimatedSalary   1.976874e-10  1.824674e-11 -7.850909e-11 -3.430272e-15
                 NumOfProducts    HasCrCard1 IsActiveMember1 EstimatedSalary
(Intercept)      -3.775016e-03 -2.595873e-03    5.010389e-04    -2.301952e-08
CreditScore      -8.271320e-08  2.066409e-07   -3.102862e-07     1.070775e-12
GeographyGermany -3.655500e-04 -7.611739e-05   -1.008754e-04    -7.124867e-11
GeographySpain   -3.357316e-05  3.603876e-05   -3.954646e-05    -8.778366e-11
GenderMale        5.853316e-05 -8.288460e-06    2.637202e-05     1.976874e-10
Age               2.887014e-06  1.405830e-07   -3.924088e-05     1.824674e-11
Tenure           -2.575910e-06 -1.356629e-05    1.978922e-05    -7.850909e-11
Balance           6.486034e-09  6.276417e-10   -5.129968e-10    -3.430272e-15
NumOfProducts     2.221635e-03 -3.379445e-06   -3.685755e-05    -4.030670e-10
HasCrCard1       -3.379445e-06  3.521179e-03    6.930480e-05     3.401694e-11
IsActiveMember1  -3.685755e-05  6.930480e-05    3.327629e-03     2.676211e-10
EstimatedSalary  -4.030670e-10  3.401694e-11    2.676211e-10     2.243567e-13
```

There are several categorical variables here, so after expansion to dummies, $AVar(\hat{\beta})$ is $12 \times 12$. This is the covariance matrix for the vector of estimated logistic regression coefficients $\hat{\beta}$.

Say we wish to compare Germany and Spain. The difference will be of the form $a'\beta$. What should we take for $a$?

```
> row.names(acov)
 [1] "(Intercept)"      "CreditScore"      "GeographyGermany" "GeographySpain"
 [5] "GenderMale"       "Age"              "Tenure"           "Balance"
 [9] "NumOfProducts"    "HasCrCard1"       "IsActiveMember1"  "EstimatedSalary"
```

Ah, we set $a$ to (0,0,1,-1,0,0,0,0,0,0,0,0)'. Using Equation 4.7, we compute the standard error and the CI:

```
a <- c(0,0,1,-1,0,0,0,0,0,0,0,0)
avar <- t(a) %*% acov %*% a
se <- sqrt(avar)
estdiff <- t(a) %*% coef(glmOut)
c(estdiff-1.96*se,estdiff+1.96*se)
```

```
[1] 0.5850204 0.8939729
```

## 6.3 The Delta Method

This is one of the most useful simple tools in statistics.

### 6.3.1 Motivating example

Now as a first example, say we are estimating a population mean $\mu$ and are also interested in estimating $\log(\mu)$.

We will probably use the sample mean $\bar{X}$ to estimate $\mu$, and thus use $W = \log \bar{X}$ to estimate $\log(\mu)$. But how do we obtain a standard error for $W$?

If we just need to form a confidence interval for $\log(\mu)$, we can form a CI for $\mu$ and then take the log of both endpoints. But again, standard errors are of interest in their own right.

### 6.3.2 Use of the Central Limit Theorem

The Central Limit Theorem tells us that $\bar{X}$ is asymptotically normally distributed. But what about $\log \bar{X}$?

From calculus, we know that a smooth function $f$ can be written as a Taylor series,

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)(x - x_0)^2/2 + ...$$

where ' denotes derivative rather than matrix transpose.

In our case here, setting $f(t) = \log t$, $x_0 = \mu$ and $x = \bar{X}$, we have

$$W = \log \mu + \log'(\mu)(\bar{X} - \mu) + \log''(\mu)(\bar{X} - \mu)^2/2 + ...$$

and $\log'(t) = 1/t$ and so on.

The key point is that as n grows, $\bar{X} - \mu$ goes to 0, and $(\bar{X} - \mu)^2$ goes to 0 even faster. Using theorems from probability theory, one can show that, *in the sense of distribution*,

$$W \approx log(\mu) + log'(\mu)(\bar{X} - \mu)$$

The right-hand side is a linear function of $\overline{X}$. The latter is asymptotically normal by the Central Limit Theorem, and thus the linear function $W$ is also asymptotically normal.

In other words, $W$ has an approximate normal distribution that has mean $log(\mu)$ and variance

$$\frac{1}{\mu^2}\sigma^2/n \tag{6.2}$$

Note the wording. The distribution of $W$ is close to a normal distribution, but mean and variance formulas here apply to that latter distribution, *not* that of $W$. For example, $E(W)$ could be far from Equation 6.2.

where $\sigma^2$ is the population variance $Var(X)$ . We estimate the latter by the usual $S^2$ quantity, and thus have our standard error,

$$\text{s.e.}(W) = \frac{S}{\bar{X}\sqrt{n}}$$

## 6.3.3 Use of the Multivariate Central Limit Theorem

Now, what if the function $f$ has two arguments instead of one? The above linear approximation is now

$$f(v,w) \approx f(v_0,w_0) + f_1(v_0,w_0)(v-v_0) + f_2(v_0,w_0)(w-w_0) \tag{6.3}$$

where $f_1$ and $f_2$ are partial derivatives,

A *partial derivative* of a function of more than one variable is the derivative with respect to one of those variables. E.g. $\partial/\partial v \ vw^2 = w^2$ and $\partial/\partial w \ vw^2 = 2vw$.

$$f_1(v,w) = \frac{\partial}{\partial v} f(v,w)$$

$$f_2(v,w) = \frac{\partial}{\partial w} f(v,w)$$

So if we are estimating, for instance, a population quantity $(\alpha,\beta)'$ by $(Q,R)'$, standard error of the latter is the square root of

$$f_1^2(Q,R)AVar(Q) + f_2^2(Q,R)AVar(R) \tag{6.4}$$
$$+ 2f_1(Q,R)f_2(Q,R)ACov(Q,R) \tag{6.5}$$

where we have made use of Equation 4.1.

As usual, use of matrix notation can help clean up messy expressions like this. The *gradient* of $f$, say in the two-argument case as above, is the vector

$$\nabla f = \left( \begin{array}{c} f_1(v_0,w_0) \\ f_2(v_0,w_0) \end{array} \right)$$

so that our above Taylor series Equation 6.3 can be written as

$$f(v,w) \approx f(v_0,w_0) + (\nabla f)' \left( \begin{array}{c} v - v_0 \\ w - w_0 \end{array} \right)$$

(Here ' is matrix transpose, not a derivative.)

Then from Equation 4.7,

$$AVar[f(Q,R)] = (\nabla f)' ACov(Q,R)(\nabla f) \tag{6.6}$$

Here 'ACov' means the asymptotic covariance matrix of the vector

$$W = \left( \begin{array}{c} Q \\ R \end{array} \right) \tag{6.7}$$

Let's call that asymptotic covariance matrix $\widehat{\Sigma}$.

### 6.3.4 Example: ratio of two means

Often $\widehat{\Sigma}$ will be provided by our application software, such as with R's **vcov** function, but we will need to derive it in this case, using properties of sample means.

Say our sample data consists of mother-daughter pairs,

$$\begin{pmatrix} M \\ D \end{pmatrix}$$

representing the heights of mother and daughter. Denote the population mean vector by

$$\nu = \begin{pmatrix} \mu_M \\ \mu_D \end{pmatrix}$$

We might be interested in the ratio $\omega = \mu_D / \mu_M$. Our estimator will be $\hat{\omega} = \bar{D}/\bar{M}$, the ratio of the sample means.

So take $Q = \bar{D}$ and $R = \bar{M}$:

$$\begin{pmatrix} Q \\ R \end{pmatrix} = \begin{pmatrix} \bar{D} \\ \bar{M} \end{pmatrix} \tag{6.8}$$

Then in Equation 6.6, with $f(q, r) = q/r$

$$\nabla f = \begin{pmatrix} 1/r \\ -q/r^2 \end{pmatrix}$$

which in our application here is

$$\nabla f = \begin{pmatrix} 1/\bar{M} \\ -\bar{D}/\bar{M}^2 \end{pmatrix}$$

Thus we will need $AVar(\bar{D})$, $AVar(\bar{M})$ and $ACov(\bar{D}, \bar{M})$. These quantities are exact, not asymptotic, so we can simplify our notation, e.g. changing $AVar(\bar{M})$ to $Var(\bar{M})$.

Be sure to distinguish between population and sample quantities. For example:

- $Var(M)$ measures how $M$ varies across all individuals in the population,

  $$E[(M - EM)^2]$$

  We can estimate it via its sample analog:

  $$\widehat{Var}(M) = \frac{1}{n}\sum_{i=1}^{n}(M_i - \bar{M})^2$$

  which measures how much $M$ varies in our sample. Similarly,

  $$E[(D - ED)^2]$$

  measures how much $D$ varies in our sample, and so on.

- $Var(\bar{M})$ measures how $\bar{M}$ varies across all size-$n$ samples in the population.

- We know from statistics that

  $$Var(\bar{M}) = \frac{1}{n}Var(M)$$

  Similar properties hold for $Var(\bar{D})$ and $Cov(\bar{D}, \bar{M})$:

In other words, we obtain the needed covariance matrix of Equation 6.8 via the sample analog of Equation 4.3 as

$$\frac{1}{n^2}\sum_{i=1}^{n}(W_i - \bar{W})(W_i - \bar{W})' \tag{6.9}$$

where

$$W_i = \begin{pmatrix} D_i \\ M_i \end{pmatrix}$$

is the value of $W$ for the $i^{th}$ daughter-mother pair in our data.

## 6.4 Example: Mother/Daughter Height Data

```r
library(WackyData)
data(Heights)
head(heights)
```

```
  Mheight Dheight
1    59.7    55.1
2    58.2    56.5
3    60.6    56.0
4    60.7    56.8
5    61.8    56.0
6    55.5    57.9
```

```r
m <- heights[,1]
d <- heights[,2]
meanm <- mean(m)
meand <- mean(d)
fDel <- matrix(c(1/meanm,-meand/meanm^2),ncol=1)
n <- length(m)
sigma <- (1/n) * cov(cbind(m,d))
se <- sqrt(t(fDel) %*% sigma %*% fDel)
se
```

```
            [,1]
[1,] 0.001097201
```

```r
meanmd <- meanm / meand
meanmd
```

```
[1] 0.9796356
```

```r
c(meanmd - 1.96*se, meanmd + 1.96*se)
```

```
[1] 0.9774850 0.9817861
```

## 6.5 Regarding Those Pesky Derivatives

Though finding expressions for the derivatives in the above example was not onerous, the function $f$ can be rather complex, with the expressions for its derivatives even more complicated. Typically such tedious and error-prone operations can be avoided, by having the software calculate approximate derivatives.

Recall the definition of derivative:

$$f'(x) = \lim_{w \to 0} \frac{f(x+w) - f(x)}{w}$$

So an approximate value of $f'(x)$ is obtained by choosing some small value of $w$ and evaluating

$$\frac{f(x+w) - f(x)}{w}$$

Though of course there is an issue with one's choice of $w$, the point is that one can code the software to find approximate derivatives automatically using this device. *This is very common in Data Science libraries.*

For example, the R package **numDeriv** will compute numerical derivatives.

## 6.6 Scheffe's Method

In analyzing the Iranian Churn data above, we might form many CIs, each at the 0.95 level. However, we may wish to set an *overall* level to at least 0.95, meaning that the probability that at least one of the CIs fails to contain the desired population value is at most 0.05. This concept is variously known as *multiple inference*, *simultaneous inference* or *multiple comparisons.*

Whether to do this is a philosophical question, and the answer will depend on one's goals and personal preferences, or may possibly be due to requirements of a research journal or employer.

If we do wish to pursue the matter, then how? One of the most well-known approaches makes good use of linear algebra.

### 6.6.1 A confidence set for $\widehat{\beta}$ in the linear model

It can be shown that if a random vector $W$ has a $k$-dimensional normal distribution random vector with mean vector $\mu$ and covariance matrix $\Sigma$, then

$$Q = (W - \mu)' \Sigma^{-1} (W - \mu)$$

has a *chi-square* distribution with $k$ *degrees of freedom*. Say $d_\alpha$ is the upper-$\alpha$ quantile of that distribution, i.e.

$$P[(W - \mu)' \Sigma^{-1} (W - \mu) \leq d_\alpha] = 1 - \alpha$$

The chi-square distribution with $k$ degrees of freedom is defined to be the distribution of the sum of the squares of $k$ independent N(0,1) random variables. After applying properties such as Equation 4.6, one can show that the quadratic form here also has that distribution.

In the case $k = 2$, the set of all $t = (t_1, ..., t_k)'$ such that

$$(t - \mu)' \Sigma^{-1} (t - \mu) = d_\alpha$$

is an ellipse in the $(t_1, t_2)$ plane. (If $\Sigma = I$, we have a circle.) For $k = 3$ we have an ellipsoid, a "football," and so on in higher dimensions..

Now, in the context of the linear model, take $W$ to be $\widehat{\beta}$, $\mu = \beta$ etc. Here $\Sigma$ must be estimated by $\widehat{\Sigma}$ (given to us via **vcov**), so we should replace the chi-square distribution by the *F-distribution*. But for simplicity, let's stick with chi-square, which is a good approximation for large $n$ anyway.

We will use the linear model here for concretenss, but the same analysis holds for any asymptotically normal statistical estimator, e.g. Maximum Likelihood estimates.

So, the event

$$(\widehat{\beta} - \beta)' \widehat{\Sigma}^{-1} (\widehat{\beta} - \beta) \leq d_\alpha$$

has probability $1 - \alpha$. Equivalently, the event

$$(\beta - \widehat{\beta})' \widehat{\Sigma}^{-1} (\beta - \widehat{\beta}) \leq d_\alpha \tag{6.10}$$

has that probability as well.

Therefore, the set of all $\beta$ satisfying Equation 6.10 is a $100(1-\alpha)$ confidence set for the true population $\beta$.

This gives us a confidence ellipse for $\beta$ in two dimensions, a confidence ellipsoid in three dimensions and so on.

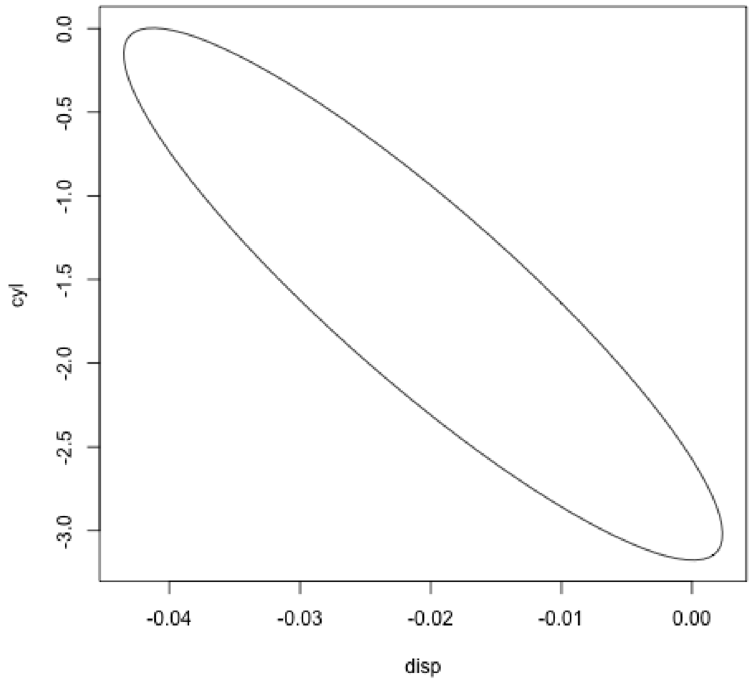
The **ellipse** library can draw this for us:

```
library(ellipse)
data(mtcars)
fit <- lm(mpg ~ disp + cyl , mtcars)
```

Calling

```
plot(ellipse(fit, which = c('disp', 'cyl'), level = 0.90), type = 'l')
```

then gives us



Here the axes are $\beta_{disp}$ and $\beta_{cyl}$.

By the way, note that the smaller $\alpha$ is, the larger will be the value of $d_\alpha$, thus the larger the ellipse. The reader should pause to confirm that this makes sense.

However, this is but an intermediate step toward our goal of a multiple inference method. Our next step will be to set up a math tool.

### 6.6.2 Lagrange multipliers

In Chapter 5, we saw the power of matrix derivatives, in our case to minimize a sum of squares. Here we go one step further, again doing optimization, but in this case under a constraint. To this end, we first need to introduce the concept of *Lagrange multipliers.*

The context is that we wish to minimize/maximize a quantity $f(w)$, subject to $g(w) = 0$, where $w$ is a vector argument. We set up the expression $f(w) + \lambda g(w)$, and find its extreme values.

For instance, say we wish to minimize

$$f(x, y) = x^2 + 2y^2$$

subject to the constraint.

$$3x + y = 8$$

We form the expression

$$x^2 + 2y^2 + \lambda(3x + y - 8)$$

and take partial derivatives with respect to $x$, $y$ and $\lambda$:

$$0 = 2x + 3\lambda$$

$$0 = 4y + \lambda$$

$$0 = 3x + y - 8$$

From the first two equations, we have

$$0 = 2x - 12y$$

Substituting, we find that

$$y = \frac{8}{19}$$

and so on.

The Lagrange multiplier here is $\lambda$. If we have several constraints, we have several multipliers.

### 6.6.3 Simultaneous confidence intervals for quantities $a'\beta$

In Section 6.2, we saw how to form confidence intervals for a quantity $a'\beta$. We may wish to form several, or even many, such intervals. Here is how the Scheffe' method can make the intervals simultaneous.

To find a CI for $a'\beta$, we find its maximum and minimum values subject to $\beta$ being in the ellipsoid, i.e. subject to the constraint

$$(\beta - \widehat{\beta})'\widehat{\Sigma}^{-1}(\beta - \widehat{\beta}) = d_\alpha \tag{6.11}$$

In other words, we solve the Lagrange multiplier problem

$$min, max_{\beta,\lambda} \;\; a'\beta + \lambda[(\beta - \widehat{\beta})'\widehat{\Sigma}^{-1}(\beta - \widehat{\beta}) - d_\alpha] \tag{6.12}$$

Those max and min values then define our CI.

Let $C$ be a symmetric matrix. The derivative of a quadratic form $u'Cu$ with respect to $u$ can be shown to be $2Cu$. Differentiating with respect to $\beta$, we thus have

There is an easily-missed subtlety here. Our phrasing "subject to $\beta$ being *in* the ellipsoid" (i.e. either in the interior or on the boundary) is at odds with the Langrange formulation, which stipulates that the min or max values occur on the boundary. But the latter property is implied by the linearity of $a'\beta$: Consider any point $q$ that is strictly interior to the ellipsoid, so our objective function has value $a'q$. Then there is room for us to move away from $q$ yet still be inside the ellipsoid, say to the point $u$, yet with $a'u$ either larger or smaller that $a'q$, depending on the direction we move in. So the min or max cannot be in the interior.

$$0 = a + 2\lambda[\widehat{\Sigma}^{-1}(\beta - \widehat{\beta})]$$

so that

$$\beta - \widehat{\beta} = -\frac{1}{2\lambda}\widehat{\Sigma}a \tag{6.13}$$

Now substitute for $\beta - \widehat{\beta}$ in the constraint Equation 6.11:

$$d_\alpha = (-\frac{1}{2\lambda}\widehat{\Sigma}a)'\widehat{\Sigma}^{-1}(-\frac{1}{2\lambda}\widehat{\Sigma}a) = \frac{1}{4\lambda^2}(a'\widehat{\Sigma}a)$$

Solve for $\lambda$:

$$2\lambda = \pm\sqrt{\frac{1}{d_\alpha}a'\widehat{\Sigma}a}$$

Finally, recall that the quantity whose min and max interest us is $a'\beta$. Using Equation 6.13, we have our CI for $a'\beta$,

$$a'\widehat{\beta} \pm \sqrt{d_\alpha a'\widehat{\Sigma}a} \tag{6.14}$$

Now, why does this interval hold *simultaneously* over *all* $a$? The point is that once we know $\beta$ is in the ellipsoid, then the above algebraic computations show that for *any* $a$, the quantity $a'\beta$ will be between $a'\widehat{\beta} - \sqrt{d_\alpha a'\widehat{\Sigma}a}$ and $a'\widehat{\beta} + \sqrt{d_\alpha a'\widehat{\Sigma}a}$. Again, this is solely an algebraic property, not a probabilistic one; the only probabilistic action occurred in $\beta$ being in the ellipsoid. Thus Equation 6.14 will hold for all $a$ simultaneously.

However, note that we pay a price for the simultaneity, in that Equation 6.14 will be wider than the ordinary CI for $a'\beta$, using Equation 6.1,

$$a'\widehat{\beta} \pm z_\alpha\sqrt{a'\widehat{\Sigma}a}$$

Consider our Iranian Churn data analysis above, with $k = 12$. Let's find $\sqrt{d_\alpha}$, for $\alpha = 0.05$:

```r
sqrt(qchisq(0.95,12))
```

```
[1] 4.585419
```

By contrast, $z_\alpha = 1.96$. The Scheffe' interval is more than twice as wide. As the saying goes, there is no "free lunch."

## 6.7 Your Turn

**Your Turn:** In the mother/daughter data, find the estimated covariance between the two heights.

**Your Turn:**

Write an R function with call form

```r
regFtnCI(lmOut,t,alpha)
```

that returns an approximate $(1-\alpha)$ confidence interval for the conditional mean $E(Y|X=t)$. Here **lmOut** is the object returned by a call to **lm**.

**Your Turn:**

The *geometric mean* of a set of $n$ numbers is the $n^{th}$ root of their product. Write an R function with call form

```r
geoGMxbarybar(x,y,alpha)
```

that returns an approximate $100(1-\alpha)$ percent CI for the population geometric mean of the two numbers $E(X)$ and $E(Y)$, using the sample analog

$$\sqrt{\bar{x}\ \bar{y}}$$

# 7 Matrix Rank

Goals of this chapter:

Many matrices are noninvertible. The subject of this chapter. matrix *rank* is closely tied to the invertibility issue, and is central to understanding matrix applications. We will see in later chapters that the notion of rank is particularly useful in modern Data Science.

In our computations in the latter part of Chapter 5, we added a proviso that $(A'A)^{-1}$ exists. In this chapter, we'll present a counterexample, which will naturally lead into our covering matrix rank, and in the next chapter, the basics of vector spaces.

## 7.1 Example: Census Data

```r
data(svcensus)
svc <- svcensus[,c(1,4:6)]
head(svc)
```

```
       age wageinc wkswrkd gender
1 50.30082   75000      52 female
2 41.10139   12300      20   male
3 24.67374   15400      52 female
4 50.19951       0      52   male
5 51.18112     160       1 female
6 57.70413       0       0   male
```

```r
lm(wageinc ~ .,data=svc)
```

```

Call:
lm(formula = wageinc ~ ., data = svc)

Coefficients:
(Intercept)          age      wkswrkd   gendermale
   -29384.1        496.7       1372.8      10700.8
```

So, we estimate that, other factors being equal, men about paid close to $11,000 more than women. This is a complex issue, but for our purposes here, how did **gender** become **gendermale**, no explicit mention of women?

Let's try to force the issue:

```r
svc$man <- as.numeric(svc$gender == 'male')
svc$woman <- as.numeric(svc$gender == 'female')
svc$gender <- NULL
head(svc)
```

```
       age wageinc wkswrkd man woman
1 50.30082   75000      52   0     1
2 41.10139   12300      20   1     0
3 24.67374   15400      52   0     1
4 50.19951       0      52   1     0
5 51.18112     160       1   0     1
6 57.70413       0       0   1     0
```

```r
lm(wageinc ~ .,data=svc)
```

```
Call:
lm(formula = wageinc ~ ., data = svc)

Coefficients:
(Intercept)          age      wkswrkd          man        woman
   -29384.1        496.7       1372.8      10700.8           NA
```

Well, we couldn't force the issue after all. Why not? We hinted above that $A'A$ may not be invertible. Let's take a look.

```r
A <- cbind(1,svc[,-2])
A <- as.matrix(A)
ApA <- t(A) %*% A
ApA
```

```
                1         age  wkswrkd      man    woman
1         20090.0    794580.7   907240  15182.0   4908.0
age      794580.7  33956543.6 35869770 600860.8 193719.9
wkswrkd  907240.0  35869770.5 45252608 692076.0 215164.0
man       15182.0    600860.8   692076  15182.0      0.0
woman      4908.0    193719.9   215164      0.0   4908.0
```

Is this matrix invertible? Let's apply the elementary row operations introduced in Section 3.5.1:

```r
library(pracma)
rref(ApA)
```

```
        1 age wkswrkd man woman
1       1   0       0   0     1
age     0   1       0   0     0
wkswrkd 0   0       1   0     0
man     0   0       0   1    -1
woman   0   0       0   0     0
```

Aha! Look at that row of 0s! The row operations process ended prematurely. This matrix will be seen to have no inverse. We say that the matrix has *rank* 4 – meaning 4 nonzero rows – when it needs to be 5. We also say that the matrix is of *nonfull rank*.

Though we have motivated the concept of matrix rank here with a linear model example, and will do so below, the notion pervades all of linear algebra, as will be seen in the succeeding chapters.

We still have not formally defined rank, just building intuition, but toward that end, let us first formalize the row operations process.

## 7.2 Reduced Row Echelon Form (RREF) of a Matrix

We formalize and extend Section 3.5.1.

### 7.2.1 Elementary row operations

These are:

- Multiply a row by a nonzero constant.
- Add a multiple of one row to another.
- Swap two rows.

Again, each operation can be implemented via pre-multiplying the given matrix by a corresponding elementary matrix. The latter is the result of applying the given operation to the identity matrix $I$. For example, here is the matrix corresponding to swapping rows 2 and 3:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

For example:

```r
e <- rbind(c(1,0,0),c(0,0,1),c(0,1,0))
e
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    0    1
[3,]    0    1    0
```

```r
a <- matrix(runif(12),nrow=3)
a
```

```
          [,1]      [,2]      [,3]       [,4]
[1,] 0.9494635 0.9084006 0.8968606 0.40691779
[2,] 0.3808983 0.9716125 0.2386077 0.06532082
[3,] 0.7758433 0.9581783 0.5169864 0.31278347
```

```r
e %*% a  # matrix mult. is NOT e * a!
```

```
          [,1]      [,2]      [,3]       [,4]
[1,] 0.9494635 0.9084006 0.8968606 0.40691779
[2,] 0.7758433 0.9581783 0.5169864 0.31278347
[3,] 0.3808983 0.9716125 0.2386077 0.06532082
```

Yes, rows 2 and 3 were swapped.

### 7.2.2 The RREF

By applying these operations to a matrix $A$, we can compute its *reduced row echelon form* $A_{rref}$, which is defined by the following properties:

- Each row, if any, that consists of all 0s is at the bottom of the matrix.
- The first nonzero entry in a row, called the *pivot*, is a 1.
- Each pivot will be to the right of the pivots in the rows above it.
- Each pivot is the only nonzero entry in its column.

The reader should verify that the matrix at the end of Section 7.1 has these properties.

### 7.2.3 Recap of Section 3.5.1

- Each row operation can be performed via premultiplication by an elementary matrix. Each such matrix is invertible, and its inverse is an elementary matrix.
- Thus

  $$B_{rref} = E_k ... E_2 E_1 B \tag{7.1}$$

  for a sequence of invertible elementary matrices.
- And

  $B = (E_1)^{-1} (E_2)^{-1} ... (E_k)^{-1} B_{rref}$,

  where each $(E_i)^{-1}$ is itself an elementary row operation.

### 7.2.4 Partitioned-matrix view of Reduced Echelon Forms

It is much easier to gain insight from RREF by viewing it in partitioned-matrix form:

> Say $A$ is of size $m \times n$. Then $A_{rref}$ has the form
>
> $$\begin{pmatrix} I_s & U \\ 0 & 0 \end{pmatrix} \tag{7.2}$$
>
> where $I_s$ is an identity matrix of some size $s$ and $U$ has dimension $s \times (n-s)$.
>
> Note that the second partitioned row here, (0,0) may consist of several actual 0 rows, or none at all.

The Column-Reduced Echelon Form is similar:

$$A_{cref} = \begin{pmatrix} I_t & 0 \\ V & 0 \end{pmatrix} \tag{7.3}$$

where $I_t$ is an identity matrix of some size $t$ and $V$ has dimension $(m-t) \times t$.

Clearly, the row rank of $A_{rref}$ is $s$, and its column rank is $t$. We will prove shortly that $s = t$.

## 7.3 Formal Definitions

> *Definition:* A *linear combination* of vectors $v_1, v_2, ...v_k$ is a sum of scalar multiples of the vectors, i.e.
>
> $$a_1 v_1 + ... + a_k v_k$$
>
> where the $a_i$ are scalars.

(The reader may wish to review Section 2.3.)

*Definition:* We say a set of vectors is *linearly dependent* if some linear combination of them (excluding the trivial case in which all the coefficients are 0) equals the 0 vector. If no nontrival linear combination of the vectors is 0, we say the vectors are *linearly independent*.

For instance, consider the matrix

$$\begin{pmatrix} 1 & 3 & 1 \\ 1 & 9 & 4 \\ 0 & 8 & 0 \end{pmatrix}$$

Denote its columns by $c_1$, $c_2$ and $c_3$. Observe that

$$(-1)c_1 + (1)c_2 + (-2)c_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Then $c_1$, $c_2$ and $c_3$ are not linearly independent.

So, here is the formal definition of rank:

*Definition* The *rank* of a matrix $B$ is its maximal number of linearly independent rows.

As an example involving real data, let's look again at the census example,

```r
head(A)
```

```
  1      age wkswrkd man woman
1 1 50.30082      52   0     1
2 1 41.10139      20   1     0
3 1 24.67374      52   0     1
4 1 50.19951      52   1     0
5 1 51.18112       1   0     1
6 1 57.70413       0   1     0
```

the sum of the last two columns of $A$ is equal to the first column, so

column1 - column4 - column5 = 0

Write this more fully as an explicit linear combination of the columns of this matrix:

1 column1 + 0 column2 + 0 column3 + (-1) column4 + (-1) column5 = 0

So we have found a linear combination of the columns of this matrix, with coefficients (1,0,0,-1,-1), that evaluates to 0. Though we have defined rank in terms of rows, one can do so in terms of columns as well:

## 7.4 Row and Column Rank

We have defined the rank of a matrix to be the number of maximally linearly independent rows. We'll now call that the *row rank*, and define the *column rank* to be the number of maximally linearly independent columns. So for instance the matrix analyzed at the end of the last section is seen to be of nonfull column rank.

As will be seen below, the row and column ranks will turn out to be equal. Thus in the sequel, we will use the term *rank* to refer to their common value. For now, though, the unmodified term "rank" will mean row rank.

## 7.5 Some Properties of Ranks

In Section 7.1, we found the matrix $A'A$ to be noninvertible, as its RREF had a row of 0s. We mentioned a relation to rank, but didn't formalize it, which we will do now.

**Theorem 7.1.** *Let $A$ be any matrix, and let $V$ and $W$ be square, invertible matrices with sizes conformable with the products below. Then the column rank of $VA$ is equal to that of $A$, and the row rank of $AW$ is equal to that of $A$.*

*Proof.* Say $A$ has column rank $r$. Then there exist $r$ (and no more than $r$) linearly independent columns of $A$. For convenience of notation, say these are the first $r$ columns, and call them $c_1, ..., c_r$. Then the first $r$ columns of $VA$ are $Vc_1, ..., Vc_r$, by matrix partitioning; call these columns $\tilde{c}_1, ..., \tilde{c}_r$.

Are these vectors linearly independent? Suppose not. Then there would exist nontrivial $\lambda_i$ such that

$$\lambda_1 \tilde{c}_1 + ... + \lambda_r \tilde{c}_r = 0$$

Multiplying both sides by $V^{-1}$ then gives us

$$\lambda_1 c_1 + ... + \lambda_r c_r = 0$$

contradicting the linear independence of the $c_i$.

Thus $VA$ has at least $r$ independent columns, and thus has column rank at least $r$. But a similar argument shows that if $VA$ were to have $sr$ independent columns, the same would have to be true for $A$, again a contradiction. The $VA$ has the same column rank as $A$.

Note that for a vector $x$, $Vx = 0$ if and only if $x = 0$. (Just multiply $Vx = 0$ on the left by $V^{-1}$.) So a linear combination $\lambda_1 c_1 + ... + \lambda_r c_r$ is 0 if and only if the corresponding linear combination $\lambda_1 V c_1 + ... + \lambda_r V c_r$ is 0. Thus the vectors $Vc_i$ are linearly independent, and $VA$ has column rank $r$.

The argument for the case of $AW$ is identical, this time involving rows of $A$.

□ □

**Theorem 7.2.** *The column rank of a matrix $B$ is equal to the column rank of its RREF, $B_{rref}$, which in turn is $s$ in Equation 7.2.*

*Proof.* Theorem 7.1 says that pre-postmultiplying $B$ will not change its column rank. Then invoke Equation 7.1.

It is clear that any column in $U$ in Equation 7.2 can be written as a linear combination of the columns of $I_s$. Thus the column rank of $B$ is $s$.

□ □

**Lemma 7.1.** *An elementary row operation on a matrix leaves the row rank unchanged.*

*Proof.* Let $r_i, i = 1, ..., m$ denote the rows of the matrix. Consider a linear combination

$$a_1 r_1 + ... + a_m r_m \neq 0$$

For any of the three elementary operations, a slightly modified set of the $a_i$ works (i.e. will be nonzero), using the modified elementary matrix:

- Swap rows $i$ and $j$: Swap $a_i$ and $a_j$.
- Multiplying row $i$ by a constant $c$. Since $r_i \to cr_i$, set $a_i \to (1/c)a_i$.
- Add $c$ times row $j$ to row $i$: Set $a_j \to a_j - ca_j$.

□ □

**Theorem 7.3** (The row rank and column rank of a matrix are equal.)**.**

*Proof.* The lemma shows that every nonzero linear combination of the rows of $A$ corresponds to one for the rows of $A_{rref}$, and vice versa. Thus the row rank of $A$ is the same as that of $A_{rref}$. But that is $s$ in Equation 7.2, which we found earlier was equal to the column rank of $A$.

□ □

In Section 7.1, we found the matrix $A'A$ to not be of full rank. It is surprising that so was $A$:

```
qr(ApA)$rank  # qr is a built-in R function
```

```
[1] 4
```

```
qr(A)$rank
```

```
[1] 4
```

This is rather startling. $A$ has over 20,000 rows — yet only 4 linearly independent ones? But it follows from this fact:

There are many subsets of 4 rows that are linearly independent. But no sets of 5 or more are linearly independent.

**Theorem 7.4.** *The (common value of) row and column ranks of an $m \times n$ matrix $B$ must be less than or equal to the minimum of the number of rows and columns of $B$ .*

*Proof.* The row rank of $b$ equals its column rank. The former is bounded above by $m$ while the latter's bound is $n$.

□ □

**Theorem 7.5.** *The matrix $A'A$ has the same rank as $A$.*

*Proof.* Using the column analog of Equation 7.1, we can write

$$A_{cref} = AF$$

where $F$ is an invertible product of matrices for elementary column operations. Then

$$(A_{rref})'A_{rref}) = F'A'AF$$

Theorem 7.1 tells us that the rank of $F'A'A$ has the same rank as $A'A$, and that $F'A'AF$ has the same rank as $F'A'A$. The latter in turn has the same rank as $A'A$. So we can concentrate on $(A_{rref})'A_{rref})$.

Using Equation 7.2, we have

$$(A_{rref})' A_{rref}) = \begin{pmatrix} I_s & U \\ U' & U'U \end{pmatrix}$$

The presence of $I_s$ tells us there are at least $s$ linearly independent rows, thus rank at least $s$. So the rank of $A'A$ is at least that of $A$.

On the other hand, again apply our knowlege of partitioning.

- Say $A$ is $m \times n$. Write

  $$A'A = \begin{pmatrix} c_1 \\ ... \\ c_n \end{pmatrix} A = \begin{pmatrix} c_1 A \\ ... \\ c_n A \end{pmatrix}$$

  where the $c_i$ are the columns of $A$, thus the rows of $A'$.

  Thus each row $c_i A$ of $A'A$ is a linear combination of the rows of $A$.

- Recall that the rank of $A'A$ is its maximal number of linearly independent rows. We saw above that each row of $A'A$ is a linear combination of the rows of $A$. Thus in turn, each linear combination of the rows of $A'A$ is a linear combination of linear combinations of rows of $A$ – still a linear combination of the rows of $A$!

- At most $s$ members of that latter linear combination are linearly independent. In other words, the rank of $A'A$ is at most $s$.

- Thus the rank of $A'A$ is also $s$.

□ □

## 7.6 Your Turn

**Your Turn:** Consider the matrix **A** in Section 7.3. Using R, show that Theorem 7.5 does indeed hold for this matrix.

**Your Turn:** Consider the full Census dataset **svcensus** , with several categorical variables. Use **factorsToDummies**

(via the **qeML** package) so that there is a different column for each level of a categorical variable. E.g. there should be six columns coming from the original **occ**. Reason out what the rank should be of the resulting data matrix, and use R to verify.

**Your Turn:** Consider an $m \times n$ matrix $A$ with $m \geq n$. Then consider the partitioned matrix

$$B = \begin{pmatrix} A \\ I \end{pmatrix}$$

where $I$ is the $n \times n$ identity matrix. Explain why $B$ is of full rank.

**Your Turn:** Show that a set of vectors is linearly dependent if and only if one of the vectors is a linear combination of the others.

**Your Turn:** Consider the three basic elementary matrices discussed here: Swap rows $i$ and $j$; multiply row $i$ by a constant $b$; adding $c$ times row $i$ to row $j$, for a constant $c$. Find general formulas for the determinants of the three matrices.

**Your Turn:** Consider a unidirectional graph $G$ of $n$ people, with an edge from $x$ to $y$ meaning that $x$ has worked for $y$ at some time. Suppose there are $r$ people whose only work experience is to work for person 3. What can we then say about the rank of $G$WW?

# 8 Vector Spaces

$x^2$

Goals of this chapter:

As noted earlier, the two main structures in linear algebra are matrices and vector spaces. We now introduce the latter, a bit more abstract than matrices but even more powerful. We lay the crucial foundation in this chapter. There won't be any direct practical applications in this chapter, but we will then reap the huge benefits of this material in the applications in the remaining chapters.

## 8.1 Review of a Matrix Rank Property

In the last chapter, we presented the concepts of matrix row and column rank, defined to be the maximal number of linearly independent combinations of the rows or columns of the matrix, respectively. We proved that

For any matrix $B$,

$$\text{rowrank}(B) = \text{colrank}(B) = \text{rowrank}(B_{rref}) = \text{colrank}(B_{rref})$$

We can say something stronger:

**Theorem 8.1.** *Let $V$ denote the* span *of the rows of $B$, i.e. the set of all possible linear combinations of rows of $B$. Define $V_{rref}$ similarly for $B_{rref}$. Then*

$$V = V_{rref}$$

*The analogous result holds for columns.*

*Proof.* Actually, the theorem follows immediately from our comment following Lemma 7.1: "every nonzero linear combination of the rows of $A$ corresponds to one for the rows of $A_{rref}$, and vice versa." This showed a one-to-one correspondence between the two sets of linear combinations.

□ □

So, not only do the two matrices have the same maximal numbers of linearly independent rows, they also generate *the same linear combinations* of those rows.

The sets $V$ and $V_{rref}$ are called the *row spaces* of the two matrices, and yes, they are examples of vector spaces, as we will now see.

## 8.2 Vector Space Definition

A set of objects $W$ is called a *vector space* if it satisfies the following conditions:

- Some form of addition between vectors $u$ and $v$, denoted $u + v$, is defined in $W$, with the result that $u$+$v$ is also in $W$. We describe that latter property by saying $W$ is *closed* under addition.
- There is a unique element called "0" such that $u + 0 = 0 + u = u$.
- Some form of scalar multiplication is defined, so that for any number $c$ and any $u$ in $W$, $cw$ exists and is in $W$. We describe that latter property by saying $W$ is *closed* under scalar multiplication
- This being a practical book with just a dash of theory, we'll skip the remaining conditions, involving algebraic properties such as commutativity of addition ($u + v = v + u$).

### 8.2.1 Examples

### 8.2.2 $R^n$

In the vast majority of examples in this book, our vector space will be $R^n$.

Here $R$ represents the set of all real numbers, and $R^n$ is simply the set of all vectors consisiting of $n$ real numbers. In an $m \times k$ matrix the rows are members of $R^m$ and the columns are in $R^k$.

### 8.2.3 The set C(0,1) of all continuous functions on the interval [0,1]

Here elements of the vector space are functions, as in calculus. Each function is an object in the vector space. Vector addition and scalar multiplication are done as functions. If say $u$ is the square root function and $v$ is the sine function, then for example $3u$ is the function

$$f(x) = 3x^{0.5}$$

and $u + v$ is defined to be

$$f(x) = x^{0.5} + \sin(x)$$

### 8.2.4 A set $RV(\Omega)$ of random variables

This vector space will consist of all random variables $X$ defined on some probability space $\Omega$, with the additional restriction that $E(X^2) < \infty$, i.e. $X$ has finite variance.

Consider the example in {Section 5.3.6} on major league baseball players. We choose a player at random. Denote weight, height and age by $W$, $H$ and $A$. Since the player is random, these three quantities are random variables.

Vector addition and scalar multiplication are defined in a straightforward manner. For instance, the sum of $H$ and $A$

is simply height + age. That is a random variable, thus a member of this vector space.

This may seem like a rather nonsensical sum, but it fits the technical definition, and moreover, we have already been doing things like this! This after all is what is happening in our prediction expression from the baseball data, e.g.

$$\text{predicted weight} = -187.6382 + 4.9236H + 0.9115A$$

In fact, in this vector space, the above is a linear combination of the random variables 1, $H$ and $A$. (1 is a nonrandom constant, but still counts as a random variable.) Note that random variables such as $HW^{1.2}$ and so on are also members of this vector space, essentially any function of $W$, $H$ and $A$ is a random variable and thus in this vector space.

## 8.3 Subspaces

Say $W_1$ a subset of a vector space $W$, such that $W_1$ is closed under addition and scalar multiplication. $W_1$ is called a *subspace* of $W$. Note that a subspace is also a vector space in its own right.

### 8.3.1 Examples

$R^3$:

For instance, take $W_1$ to be all vectors of the form (a,b,0). Clearly, $W_1$ is closed under addition and scalar multiplication, so it is subspace of $R^3$.

Another subspace of $R^3$ is the set of vectors of the form (a,a,b), i.e. those vectors whose first two element are equal. The reader should check that this set of vectors is closed under addition and scalar multiplication, thus is a subspace of $R^3$.

What about vectors of the form (a,b,a+b)? Yes.

We saw earlier that the *row space* of an $m \times n$ matrix $A$, consisting of all linear combinations of the rows of $A$, is a subspace of $R^m$. Similarly, the *column space*, consisting of all linear combinations of columns of $A$, is a subspace of $R^n$.

Another important subspace is the *null space*, the set of all $x$ such that $Ax = 0$. The reader should verify that this is indeed a subspace of $R^n$.

$C(0, 1)$:

One subspace is the set of all polynomial functions. Again, the sum of two polynomials is a polynomial, and the same holds for scalar multiplication, so the set of polynomials is closed under those operations, and is a subspace.

$RV(\Omega)$:

The set of all random variables that are functions of $H$, say, is a subspace of $RV(\Omega)$. Another subspace is the set of all random variable with mean 0.

### 8.3.2 Span of a set of vectors

As noted earlier, the *span* of a set of vectors $G = v_1, ..., v_k$ is the set of all linear combinations of those vectors. It's a subspace of the main space that the $v_i$ are members of.

## 8.4 Basis

Consider a set of vectors $u_1, ...u_r$ in a vector space $W$. Recall that the span of these vectors is defined to be the set of all linear combinations of them. In verb form, we say that $u_1, ...u_r$ *spans* $W$ if we can generate the entire vector space from those vectors via linear combinations. It's even nicer if the vectors are linearly independent:

> We say the vectors $u_1, ...u_r$ in a vector space $W$ form a *basis* for $W$ if they are linearly independent and span $W$.

Note that subspaces are vector spaces in their own right, and thus also have bases.

### 8.4.1 Example: $R^3$:

The vectors (1,0,0), (0,1,0) and (0,0,1) are easily seen to be a basis here. They are linearly independent, and clearly span $R^3$. For instance, to generate (3,1,-0.2), we use this linear combination:

Our convention has been that vectors are considered in matrix terms as column vectors by default. However, in nonmatrix contexts, it will be convenient to write $R^n$ vectors as rows.

$$(3, 1, -0.2) = 3(1, 0, 0) + 1(0, 1, 0) + (-0.2)(0, 0, 1)$$

But bases are not unique; for instance, the set (1,0,0, (0,1,0), (0,1,1) works equally well as a basis for this space (as do infinitely many others). For instance,

$$(3, 1, -0.2) = 3(1, 0, 0) + 1.2(0, 1, 0) + (-0.2)(0, 1, 1) \tag{8.1}$$

We refer to the coefficients in those linear combinations as *coordinates* (evoking the old "(X,Y} coordinates" in elementary algebra courses). In the first example above, we say:

> The coordinates of the vector (3,1,-0.2) with respect to the basis (1,0,0), (0,1,0) and (0,1,1) are 3, 1.2 and -0.2.

A basis for the subspace of vectors of the form (a,a,b) is (1,1,0) and (0,0,1).

### 8.4.2 Infinite-dimensional vector spaces

Note too that we have implictly assumed above that our vector spaces here are *finite-dimensional*, i.e. that any basis consists of finitely-many vectors ($r$ in the above definition). This need not be the case.

Example: $Poly(0, 1)$

This is the space of all polynomials defined on the interval [0,1], such as $f(t) = 2.5t$ and $g(t) = t^{13} - t^{12} - t^5 + t + 1$. Vector addition and scalar multiplication can be defined in the obvious manner, e.g.

$$h(t) = 2f(t) + 3g(t) = 3t^{13} - 3t^{12} - 3t^5 + 2t + 3$$

A natural basis is

$$u_i(t) = t^i,\ \ i = 1, 2, 3, ... \tag{8.2}$$

$C(0, 1)$:

There is no basis here, finite or infinite. But with more sophisticated mathematics, something like an infinite basis can be formed. A full account would be beyond the scope of this book.

But just to tantalize the reader, we point out that under the more general definitiion, Equation 8.2 is actually a basis for $C(0, 1)$! Again, there are mathematics niceties that must be filled in, but it turns out that any continuous function on a finite interval can be approximated by some polynomial, with as close a degree of approximation as we want. Basis is then defined in the context of approximation.

For those with a background in mathematical analysis, here is how the problem is handled. One starts with a subspace that is *dense* in the full space, i.e. any vector in the full space can be approximated to any precision by some linear combination of vectors in the dense set. For instance, the famoua Stone-Weierstrauss Theorem says that the set of all polynomials is dense in $C(0, 1)$. One then defines the dense set to be a "basis." The use of trig functions as the dense set is much more common than use of polynomials, in the familiar *Fourier series.*

$RV(\Omega)$:

The situation here is the same as for C(0,1).

## 8.5 Dimension

Geometrically, we often refer to what is called $R^3$ here as "3-dimensional.'' We extend this to general vector spaces as follows:

> The *dimension* of a vector space is the number of vectors in any of its bases.

There is a bit of a landmine in that definition, as it presumes that all the bases do consist of the same number of vectors. This is true, but must be proven. We will not do so here, and refer the interested reader to the elegant proof by AF Beardon (*Algebra and Geometry*, 2005, Cambridge).

## 8.6 Linear Transformations

Consider vector spaces $V_1$ and $V_2$. A function $f$ whose input is in $V_1$ and output is in $V_2$ is called a *transformation* (or *map*) from $V_1$ to $V_2$.

An important special case is that in which $f$ is linear, i.e.

$$f(av + bw) = af(v) + bf(w)$$

for all scalars $a$ and $b$, and all vectors $v$ and $w$ in $V_1$.

A further important special case is that in which $V_1$ and $V_2$ are $R^n$ and $R^m$, respectively. Then it can be shown that there a some $m \times n$ matrix $A$ that does the work of $f$, i.e.

$$f(x) = Ax$$

for all $x$ in $R^n$.

Note that it is not necessarily true that a linear transformation will be *one-to-one* (*injective*), meaning that $x \neq y$ implies $f(x) \neq f(y)$. Nor is it necessarily *onto* (*surjective*), i.e. thst for any $w$ in $R^m$, there exists some $v$ such that $f(v) = w$.

To see that the first is false, consider the matrix $A$ above. If say $f(v) = w$ then we will also have $f(v + s) = f(v) + f(s) = w$ for any $s$ in the null space of $A$, a contradiction if $A$'s null space consists of more than the 0 vector. The reader is encouraged to verify that the second property also does not necessarily hold either.

## 8.7 Your Turn

**Your Turn:** In the vector space $C(0, 1)$, consider the subset of all polynomial functions of degree $k$ or less. Explain why this is a subspace of $C(0, 1)$. What is its dimension? Give an example of a basis for this subspace, for $k = 3$.

**Your Turn:** Say $U$ and $V$ are subspaces of $W$. Explain why $U \cap V$ is also a subspace, and that its dimension is at most the minimum of the dimensions of $U$ and $V$. Show by counterexample that the result does not hold for union.

**Your Turn:** Citing the properties of expected value, E(), show that the set of all random variable with mean 0 is a subspace of $RV(\Omega)$.

**Your Turn:** Prove that the coefficients in a basis representation are unique. In other words, in a representation of the vector $x$ in terms of a basis $u_1, ..., u_n$, there cannot be two different sets of coefficients $c_1, ..., c_n$ such that $c_1u_1 + ... + c_nu_n = x$.

# 9 Inner Product Spaces

Goals of this chapter:

The usefulness of vector spaces is greatly enhanced with the addition of an *inner product* structures. We motivate and define such structures here, and present applications. Among other things, we will analyze a method for removing racial, gender etc. bias in machine learning algorithms.
The material in this chapter will be crucial to the remaining chapters. Extra attention on the part of the reader will yield major dividends.

## 9.1 Geometric Aspirations

You may recall from your high school geometry course the key concept of perpendicularity, represented by the  symbol. You may also recall that in 2-dimensional space, given a point P and a line L, the line drawn from point P to the closest point P' within L is perpendicular to L. The same is true if L is a plane. The point P' is called the *projection* of P onto L.

This was shown in this book's cover, shown here:

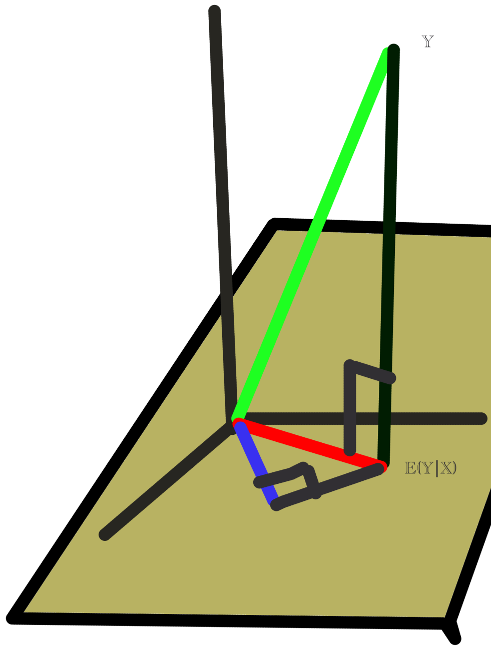


Figure 9.1: Projections

The point at the end of the green vector is projected onto the mustard-colored plane, a subspace, producing the red vector. It in turn is projected onto the blue line, the latter being a subspace of the mustard-colored one. There are right angles in each case.

The early developers of linear algebra wanted to extend such concepts to abstract vector spaces. This aids intuition, and has very powerful applications.

## 9.2 Definition

You may have seen dot products in a course on vector calculus or physics. For instance, the dot product of the vectors (3,1,1.5)' and (0,5,6)' is

3x0 + 1x5 + 1.5x6 = 14

This in fact is a standard inner product on $R^3$, but the general definition is as follows.

An *inner product* on a vector space $V$, denoted by the "angle brackets" notation $< u, v >$, is a function with two vectors as arguments and a numerical output, with the following properties:

- $< u, v >=< v, u >$
- The function is bilinear:

$$< u, av + bw >= a < u, v > +b < u, w >$$

- $< u, u > \;\geq\; 0$, with equality if and only if $u = 0$.

## 9.3 Examples

### 9.3.1 $R^n$

As noted, ordinary dot product is the most common inner product on this space.

$< (a_1, ..., a_n), (b_1, ..., b_n >= a_1 b_1 + ... + a_n b_n$

But other inner products may be defined. For instance, say an $n \times n$ matrix $A$ is *positive definite* (Equation 4.9). Then it is easily seen that the function

$$g(q, r) = q'Ar$$

fits the definition of inner product.

### 9.3.2 *C(0,1)*

One inner product on this space is

$$< f, g >= \int_0^1 f(t)g(t)\ dt$$

For instance, with $f(t) = t^2$ and $g(t) = \sin(t)$, the inner product can be computed with R:

```
f <- function(t) t^2
g <- function(t) sin(t)
fg <- function(t) f(t) * g(t)
integrate(fg,0,1)
```

```
0.2232443 with absolute error < 2.5e-15
```

This clearly fits most requirements for inner products, but what about $< f, f >= 0$ only if $f = 0$? A non-0 $f$ will have $f^2(t) > 0$ for at least one $t$, and by continuity, $f^2(t) > 0$ on an interval containing that $t$, thus making a nonzero contribution to the integral and thus to the inner product.

Note that the 0 vector in this space is the function that is identically 0, not just 0 at some points

### 9.3.3 $RV(\Omega)$

We will focus on the subspace $RV(\Omega)_0$ of random variables that have mean 0, and take covariance as our inner product:

We need to restrict attention to this subspace, in order to meet the requirement that $< U, U >= 0$ if and only if $U = 0$. Otherwise, any constant random variable would violate that condition.

$$< U, V >= Cov(U, V) = E[(U - EU)(V - EV)] = E(UV) \tag{9.1}$$

The properties of expected value, e.g. linearity, show that the requirements for an inner product hold.

## 9.4 Norm of a Vector

This concept extends the notion of a the length of a vector, as we know it in $R^2$ and $R^3$.

*Definition:*

The *norm* of a vector $x$, denoted $||x||$, is

$$(< x, x >)^{0.5}$$

The *distance* from a vector $x$ to a vector $y$ is

$$||y - x||$$

## 9.5 Sets of Orthogonal Vectors

We say that vectors $u$ and $v$ are *orthogonal* if $< u, v >= 0$. This is the general extension of the notion of perpendicularity in high school geometry.

If the set of vectors $u_1, ..., u_r$ form a basis for a vector space, it is called an *orthogonal* basis. If each $u_i$ has length 1 (which we can arrange by dividing by its norm), the set is called an *orthonormal basis.*

## 9.6 The Cauchy-Schwarz Inequality

**Theorem 9.1** (Cauchy-Schwarz Inequality)**.** *Say $u$ and $v$ are vectors in an inner product space. Then*

$$| < u, v > | \leq ||u|| \; ||v||$$

*Proof.* See the Your Turn problem below. □

### 9.6.1 Application: Correlation

*Correlation coefficients* are ubiquitous in data science. It is well known that their values fall into the interval [-1,1]. Let's prove that.

In Chapter 4, in introducing the notion of the covariance between two random variables, we remarked that covariance is intuitively like correlation, but that the latter is a scaled form. Formally,

$$\rho(X,Y) = \frac{E[(X-EX)(Y-EY)]}{\sqrt{Var(X)}\sqrt{Var(Y)}} \tag{9.2}$$

By dividing the covariance by the product of the standard deviations, we obtain a unitless quantity, i.e. free of units such as centimeters and degrees.

Now, say $X$ and $Y$ are in $RV(\Omega)_0$. Recalling our inner product for this space, we have

$$< X,Y >= E(XY)$$

and

$$||X||^2 =< X,X >= E(X^2) = Var(X)$$

with the analogous relations for $Y$.

Cauchy-Schwarz then says

$$|E(XY)| \leq \sqrt{Var(X)}\sqrt{Var(Y)}$$

which says the correlation is between -1 and 1 inclusive.

## 9.7 The Triangle Inequality

In the world of ordinary physical geometry, we know the following

*The distance from $A$ to $B$ is less than or equal to the sum of the distances from $A$ to $C$ and $C$ to $B$.* This is true as well in general, abstract inner product spaces.

**Theorem 9.2** (Triangle Inequality)**.** *In a general inner product space,*

$$||x - z|| \leq ||x - y|| + ||y - z||$$

*Proof.* See the Your Turn problem below. □

## 9.8 The Pythagorean Theorem

That this ancient theorem in geometry still holds in general inner product spaces is a tribute to the power of abstraction.

**Theorem 9.3.** *If vectors $X$ and $Y$ are orthogonal, then*

$$||X + Y||^2 = ||X||^2 + ||Y||^2 \tag{9.3}$$

*Proof.* Replace the norms by expression in inner products. Simplify using properties of inner product. □

## 9.9 Your Turn

**Your Turn:** Consider the space $C(0, 1)$. For function $f$ and $g$ of your own choosing, verify that the Cauchy-Schwarz and Triangle Inequalities hold in that case.

**Your Turn:** Derive the Cauchy-Schwarz Inequality, using the following algebraic outline:

- The inequality

$$0 \ \leq \ < (au+v), (au+v) >$$

  holds for any scalar $a$.

- Expand the right-hand side (RHS), using the bilinear property of inner products.

- Minimize the resulting RHS with respect to $a$.

- Collect terms to yield

$$< u, v >^2 \leq ||u||^2 ||v||^2$$

**Your Turn:** Use the Cauchy-Schwarz Inequality to prove the Triangle Inequality, using the following algebraic outline.

- Start with

$$||u+v||^2 = < (u+v, u+v >$$

- Expand the RHS algebraically.

- Using Cauchy-Schwarz to make the equation an inequality.

- Collect terms to yield the Triangle Inequality.

**Your Turn:** Explain why any set of orthogonal vectors is linearly independent.

# 10 Projection Operators

Goals of this chapter:

Here we go into detail on the tremendous value derived by viewing certain operations as projections.

## 10.1 Projections

As mentioned, the extension of classical geometry to abstract vector spaces has powerful applications. There is no better example of this than the idea of *projections*.

### 10.1.1 Projection Decomposition

Recall that we say that vectors $u$ and $v$ are *orthogonal* if $< u, v >= 0$. Then we have the following:

**Theorem 10.1** (Projection Theorem)**.** *Consider an inner product space $V$, with subspace $W$. Then for any vector $x$ in $V$, there is a unique vector $z$ in $W$, such that $z$ is the closest vector to $x$ in $W$. Furthermore, $x - z$ is orthogonal to any vector $r$ in $W$.*

*We say that $z$ is the* projection *of $x$ onto $W$.*

*Proof.* The full proof is beyond the scope of this book, as it requires background in real analysis. Indeed, even the statement of the theorem is not mathematically tight.

For readers who do have such background, this is the Hilbert Projection Theorem. “Closest” is defined in terms of infimum, and $W$ needs to be a topologically closed set. See for example the Wikipedia entry.

□ □

In the case of $R^n$, It will be seen shortly that for each $W$ in the theorem, there is a matrix $P_W$ that implements the projection, i.e.

$$z = P_W x$$

Now consider a subspace $W$ of an inner product space $V$. The set of vectors having inner product 0 with all vectors in $W$ is denoted $W^\perp$. It too is a subspace, and jointly $W$ and $W^\perp$ span all of $V$.

We then have the famous relations:

**Theorem 10.2.** *Say the vector $x$ is in $R^n$, and $W$ is a subspace of the latter. Then there is a matrix $P_W$ such that*

- *$P_W x$ is the projection of $x$ onto $W$.*
- *$x$ can be uniquely represented as a sum of two orthogonal terms, one in $W$ and the other in $W^\perp$:*

$$x = P_W x + (I - P_W x)$$

□

Note that projection operators are *idempotent*, meaning that if you apply a projection twice, the effect is the same as applying it once. In the matrix equation above, this means $P_W^2 = P_W$. This makes sense; once you drop down to the subspace, there is no further dropping down to that same space.

## 10.2 Orthogonal Complements and Direct Sums

From Theorem 10.1, we know that for any $x$ in $V$, one can uniquely write

$$x = x_1 + x_2 \tag{10.1}$$

where $x_1$ and $x_2$ are in $W$ and $W^\perp$, respectively.

Say $u_1, ..., u_r$ and $v_1, ..., v_s$ are bases for $W$ and $W^\perp$. Then in Equation 10.1, we can generate $x_1$ from the $u_i$ and $x_2$ from the $v_j$. So together these two sets of vectors form a basis for all of $V$. Typically they are chosen to be orthonormal. (Section 10.5 shows how we can convert the $u_i$ and $x_2$ to an orthonormal basis for $V$.)

Finally, we say that $V$ is the *direct sum* of $W$ and $W^\perp$, denoted $V = W \oplus W^\perp$, and that the two subspaces are *orthogonal complements* of each other.

## 10.3 Projections in the Linear Model

The case of the linear model will deepen our understanding, and will lead to a method for outlier detection that is commonly used in practice.

### 10.3.1 The least-squares solution is a projection

Armed with our new expertise on inner product spaces, we see that Equation 5.7 is

$$< S - Ab, S - Ab > \tag{10.2}$$

in the vector space $R^n$, where $n$ is the number of our data points.

Of course, "data points" means rows in the data frame. In the statistics realm, people often speak of "observations."

As usual, let $p$ denote the number of columns of $A$, and write $\widehat{\beta}$ for the value of $b$ that minimizes Equation 10.2.

Keep in mind, we are minimizing Equation 10.2 with respect to $b$. The set $W$ of all $Ab$, as $b$ varies, is a subspace of $R^n$. In minimizing Equation 10.2, *we are finding the value of $b$ that makes $Ab$ closest to $S$*. That means that the minimizing $Ab$ *is the projection of $S$ onto $W$*.

Then from Theorem 10.2, we have that

$$S - A\widehat{\beta}$$

is orthogonal to $A\widehat{\beta}$. In other words, it must be true that

$$\sum_{i=1}^{n} A_i \widehat{\beta}(S_i - A_i \widehat{\beta}) = 0$$

where $A_i$ denotes row $i$ of $A$.

## 10.3.2 The "hat" matrix

Recall Equation 5.10, which showed the general solution to our linear regression model:

$$\widehat{\beta} = (A'A)^{-1}A'S$$

The projection itself, i.e. the matrix $P_W x$ in Section 10.1, is then

$$A\widehat{\beta} = A(A'A)^{-1}A'S = HS$$

where

$$H = A(A'A)^{-1}A'$$

As discussed above, $H$ projects $S$ onto the space of all $Ab$. It is called the *hat matrix*.

As a projection, $H$ is idempotent, which one can easily verify by multiplication. $H$ is also symmetric.

## 10.3.3 Application: identifying outliers

An *outlier* is a data point that is rather far from the others. It could be an error, or simply an anomalous case. Even in the latter situation, such a data point could distort our $\widehat{\beta}$, so in both cases, identifying outliers, and possibly removing them, is important.

Let $h_{ii}$ denote element $i$ of the diagonal of $H$, and again with $A_i$ denoting row $i$ of A. One can show that

$$h_{ii} = A_i(A'A)^{-1}A_i' \tag{10.3}$$

The quantity $h_{ii}$ is called the *leverage* for datapoint $i$, with the metaphor alluding to the impact of datapoint $i$ on $\widehat{\beta}$.

Using the material on circular shifts in Section 2.11.3, we have

$$tr(H) = tr[\underbrace{A(A'A)^{-1}}A'] = tr[A'\underbrace{A(A'A)^{-1}}] = tr(I) = p$$

for $A$ of size $n \times p$.

Thus the average value of the $n$ quantities $h_{ii}$ is $p/n$. Accordingly, we might suspect that $A_i$ is an outlier if $h_{ii}$ is considerably larger than $p/n$.

For example, let's look at the Major League Baseball player data we've seen earlier (Section 5.3.6):

```
library(qeML)
data(mlb1)
ourData <- as.matrix(mlb1[,-1]) # must have matrix to enable %*%
A <- cbind(1,ourData[,c(1,3)])
dim(A)
```

```
[1] 1015    3
```

```
S <- as.vector(mlb1[,3])
H <- A %*% solve(t(A) %*% A) %*% t(A)
```

Let's take a look:

```
hist(diag(H))
```

The ratio $p/n$ here is 3/1015, about 0.003. We might take a look at the observations having $h_{ii}$ above 0.01, say.

## 10.4 Matrix Form of Projections in $R^n$

Our main question in this section:

> Say we have a subspace $W$ of $R^n$, and a vector $x$ in $V$. How do we find the projection of $x$ in $W$?

Call the projection $z$, and let $u_1, ..., u_k$ be an orthonormal basis for $W$. Then there exist $a_1, ..., a_k$ such that

$$z = a_1 u_1 + ..., +a_k u_k \tag{10.4}$$

We can find $z$ by first finding its coordinates $a_i$ with respect to the $u_i$, as follows. .

Below, keep in mind that $x$ is known but $z$ is not. We desire a method by which we can find $z$ from $x$.

### 10.4.1 Exploiting orthonormality

We do have a hint to work from: We know that $x - z$ is orthogonal to every vector in $W$ – including the $u_i$. So

$$0 =< x - z, u_i >=< x, u_i > - < z, u_i >$$

and thus

$$< x, u_i >=< z, u_i >$$

Since $< z, u_i >= a_i$, we have

$$a_i =< x, u_i >= u_i' x$$

In partitioned matrix forms, the vector $a$ of the coordinates of $z$ is

$$a = \left( \begin{array}{c} u_1' \\ ... \\ u_k' \end{array} \right) x$$

So, we're almost done! We wanted to determine the coordinates $a_i$ of $z$ in Equation 10.4, and now we see that we can easily obtain them by calculating $< x, u_i >$. Our projection $z$ is

$$(u_1 | ... | u_k) \left( \begin{array}{c} a_1 \\ ... \\ a_k \end{array} \right)$$

Combining the last two equations, we have

$$z = (u_1|...|u_k) \left( \begin{array}{c} u_1' \\ ... \\ u_k' \end{array} \right) x = QQ'x$$

Wrapping up:

**Theorem 10.3** (Calculating a Projection Matrix)**.** *Given: a subspace $W$ of $R^n$ with orthonormal basis $u_1, ..., u_k$; and a vector $x$ in $V$. Then the projection of $x$ onto $W$ is equal to $QQ'x$, where $Q$ is the matrix whose columns are the $u_i$.*

## 10.5 The Gram-Schmidt Method

As seen in the last section, it is desirable to have an orthogonal basis, and it's even more convenient if its vectors have length 1 (an *orthonormal* basis). As noted, converting to length 1 is trivial – just divide the vector by its length, but how do we obtain an orthogonal basis? In other words, if we start with a basis $b_1, b_2, ..., b_m$, how can we generate an orthonormal basis from this?

> *The Gram-Schmidt Method*
>
> Say we have a basis $b_1, ..., b_k$ for some vector space. Convert it to an orthonormal basis as follows.
>
> 1. Set $u_1 = b_1/||b_1||$.
> 2. For each $i = 2, ..., k$, find the projection $q$ of $b_i$ onto the subspace generated by $u_1, ..., u_{i-1}$. Set $u_i$ to $b_i - q$, and normalize it.

Wny does this work? Let $W$ denote the subspace generated by $u_1, ..., u_{i-1}$. Since $q$ is the projection of $b_i$ onto $W$, $b_i - q$ will be orthogonal to $W$, thus to $u_1, ..., u_{i-1}$ – exactly what we need.

## 10.6 Projections in $RV(\Omega)_0$

Here is a good example of how a very abstract vector space becomes useful in practical applications, such as will be presented in Section 10.7. The material is rather involved, consisting of computation of various probabilistic quantities. Since $RV(\Omega)_0$ is a vector space of random variables, each entity is both a random variable and a vector. Sometimes the latter will be the focus, sometimes the former. We request the reader's patience in following this duality.

Recall first the definition of inner products in this space, Equation 9.1, a connection between the notion of covariance in probability theory and the notion of inner product in linear algebra.

$$< U, V >= Cov(U, V) = E[(U - EU)(V - EV)] = E(UV)$$

Among other things, this implies:

**Orthogonality and Uncorrelatedness**

Recall from Equation 9.2 that correlation is the quotient of covariance and the product of standard deviations. Thus random variables $U$ and $V$ in $RV(\Omega)_0$. are uncorrelated if and only if they are orthogonal as vectors.

### 10.6.1 Conditional expectation

It will turn out that in $RV(\Omega)_0$, projections take the form of conditional means. Let's see how that arises.

Our definition of $RV(\Omega)_0$ restricts to random variables with mean 0. Thus technically the derivation here should subtract means; e.g. the random variable $Y$ below should be replaced by $\tilde{Y} = Y - EY$. However, in order to avoid clutter, we do not take this step.

One of the Your Turn problems at the end of this chapter covers this setting:

> Say we roll a die once, producing $X$ dots. If $X = 6$, we get a bonus roll, yielding $B$ additional dots; otherwise, $B = 0$. Let $Y = X + B$.

*The basic form, conditioning value specified:*

Now, what is $E(Y|B = 2)$? If $B = 2$, then we got the bonus roll, so $X = 6$ and $Y = X + B = 8$:

$$E(Y|B = 2) = 8$$

Similarly,

$$E(Y|B = 3) = 9$$

and so on.

But what about $E(Y|B = 0)$? In that case, $Y = X$, so

$$P(Y = i|B = 0) = P(X = i|X \neq 6) = \frac{1}{5},\ i = 1, 2, 3, 4, 5$$

We then have $E(Y|B = 0) = 3$.

So our answer is

$$E(Y|B = i) = \begin{cases} 3 & i = 0 \\ 6 + i & i = 1, 2, 3, 4, 5 \end{cases}$$

*The random variable form:*

The quantity $E(Y|B = i)$, a number, *can be converted to a random variable*, in the form of a function of $B$, which we will call $q(B)$. Write

$$q(B) = \begin{cases} 3 & B = 0 \\ 6 + B & B = 1, 2, 3, 4, 5 \end{cases}$$

It is standard to denote $q(B)$ by $E(Y|B)$ – without '= i'.

$B$ is random, so $q(B)$ is also random. In other words, $E(Y|B)$ is a random variable.

We need one more thing:

*The Law of Iterated Expectation:*

For random variables $U$ and $V$, set

$$R = E[V|U]$$

Then

$$E(R) = E(V)$$

More concisely:

$$E[E(V|U)] = E(V) \tag{10.5}$$

Intuitive explanation: Say we wish to compute the mean height $E(H)$ of all students at a university. We might ask each department $D$ to measure their own students, and report to us the resulting mean $E(H|D)$. We could then average all those departmental means to get the overall mean for the university:

$$E[E(H|D)] = E(H)$$

Note, though that that outer $E()$ (the first 'E') is a weighted average, since some departments are larger than others. The weights are the distribution of $D$.

### 10.6.2 Projections in $RV(\Omega)_0$: how they work

Consider random variables $X$ and $Y$ in $RV(\Omega)_0$, and consider the subspace $W$ consisting of all functions of $X$. (By definition, this means that each such function, say $g(X)$, has finite variance and mean 0.)

Theorem 10.1 talks of a closest vector $C$ in $W$ to $Y$. Let's see what form $C$ would take in this vector space.

Remember, the (squared) distance from $Y$ to $C$ is

$$||Y - C||^2 =< Y - C, Y - C >= E[(Y - C)^2]$$

By Equation 10.5, that last term is

$$E[E((Y - C)^2|X)]$$

For any random variable $T$ of finite variance, the minimum value of $E[(T - d)^2]$ over all constants $d$ is attained by taking $d$ to be the mean of $T$, i.e. $d = E(T)$. (See Your Turn problem below.) So, the minimum of $E((Y - C)^2|X)$, for all random variables $C$, is attained by the *conditional* mean,

$$C = E(Y|X)$$

In other words:

> Projections in $RV(\Omega)_0$ take the form of conditional means. Here, the projection of $Y$ onto $W$ is $E(Y|X)$.
>
> Moreover:
>
> Since the difference between a vector and its projection onto a subspace is orthogonal to that subspace we have:
>
> The vector $Y - E(Y|X)$ is uncorrelated with $E(Y|X)$. In other words, the prediction error (also called the *residual*) has 0 correlation with the prediction itself.

## 10.7 Application: Fairness in Algorithms

COMPAS is a software tool designed to aid judges in determining sentences in criminal trials, by assessing the probability that the defendant would recidivate. It is a commercial product by Northpointe.

COMPAS came under intense scrutiny after an investigation by *ProPublica*, which asserted evidence of racial bias against black defendants compared to white defendants with similar profiles. Northpointe contested these findings, asserting that their software treated black and white defendants equally.

It should be noted the *ProPublica* did not accuse Northpointe of intentional bias. Instead, the issue largely concerns *proxies*, variables that are related to race, rather than race (or gender etc.) itself. If for example COMPAS were to use a person's ZIP code as a predictor, that might be correlated with race, and thus would be unfair to use in prediction.

Let $S$ denote a vector of one or more sensitive variable, e.g. race. The point here is that, due to proxies, we cannot solve the problem by simply removing $S$ from our analysis; we would still be using correlates of $S$.

This book of course does not take a position on the specific dispute between Northpointe and *ProPublica*. The above is simply a motivational example.

### 10.7.1 Setting

We consider prediction of a variable $Y$ from a feature vector $X$ and a vector of sensitive variables $S$. The target $Y$ may be either numeric (in a regression setting) or dichotomous (in a two-class classification setting where $Y = 1$ or $Y = 0$). We will consider only the numeric case here.

*Our goal is to eliminate the influence of $S$.*

### 10.7.2 The method of Scutari *et al*

The basic assumption (BA) amounts to $(Y, X, S)$ having a multivariate Gaussian distribution, with $Y$ scalar and $X$ being a vector of length $p$. Again, all variables are assumed centered, by subtracting their means, so that they now have mean 0.

Let's review the material in Section 4.3: Say we have $W$ with a multivariate normal distribution, and wish to predict one of its components, $Y$, from a vector $X$ consisting of one or more of the other components, or linear combinations of them. Then:

- The distribution of $Y|X$ is univariate normal.
- E(Y|X=t) is a linear function of $t$.
- Var(Y|X=t) is independent of $t$.

One can also show that,

- Though having 0 correlation does not in general imply statistical independence, it does so in the multivariate normal case.

One first applies a linear model in regressing $X$ on $S$,

$$E(X|S) = \gamma' S$$

where $\gamma$ is a length-$p$ coefficient vector. Here we are predicting the predictors (of $Y$), seemingly odd, but as a first step in ridding ourselves from the influence of $S$.

Now consider the resulting prediction errors (*residuals)*,

$$U = X - \gamma' S$$

$U$ can be viewed as the part of $X$ that is unrelated to $S$; think of $U$ as "having no $S$ content." $U$ is a vector of length $p$.

Note the following:

- From Section 10.6.2, $E(X|S)$ is the projection of $X$ onto the subspace of all functions of $S$.
- $X - E(X|S)$ (original vector minus the projection) is orthogonal to $S$.
- That is,

$$\begin{aligned} 0 &=< S, X - E(X|S) > && (10.6) \\ &= E[S'(X - E(X|S))] && (10.7) \\ &= E(S'U) && (10.8) \\ &= Cov(S, U) && (10.9) \end{aligned}$$

- Thus $S$ and $U$ are uncorrelated.
- Due to the BA, that means $S$ and $U$ are independent.
- In other words, our intution above that $U$ "has no $S$ content" was mathematically correct.

- Bottom line: Instead of predicting $Y$ from $X$, use $U$ as the predictor vector. This will enable truly $S$-free prediction.

Goal achieved.

## 10.8 Your Turn

**Your Turn:** Consider the space $C(0,1)$. For function $f$ and $g$ of your own choosing, verify that the Cauchy-Schwarz and Triangle Inequalities hold in that case.

**Your Turn:** Show that for any random variable $Q$ of finite variance, the minimum value of $E[(Q-d)^2]$ over all constants $d$ is attained by taking $d$ to be the mean of $Q$, i.e. $d = E(Q)$. Hint: First expand $(Q-d)^2$ as $Q^2 - dQ + d^2$.

**Your Turn:** In the die rolling example, verify that

$$E[E(Y|B)] = E(Y)$$

by calculating both sides.

**Your Turn:** Say we roll a die once, producing $X$ dots. If $X = 6$, we get a bonus roll, yielding $B$ additional dots; otherwise, $B = 0$. Let $Y = X+B$. Verify that $X$ and $B$ satisfy the Cauchy-Schwarz and Triangle Inequalities, and also find $\rho(X, B)$.

**Your Turn:** Derive the Cauchy-Schwarz Inequality, using the following algebraic outline:

- The inequality

  $$0 \ \leq \ < (au+v), (au+v) >$$

  holds for any scalar $a$.

- Expand the right-hand side (RHS), using the bilinear property of inner products.

- Minimize the resulting RHS with respect to $a$.

- Collect terms to yield

$$< u, v >^2 \leq ||u||^2 ||v||^2$$

**Your Turn:** Use the Cauchy-Schwarz Inequality to prove the Triangle Inequality, using the following algebraic outline.

- Start with

$$||u + v||^2 =< (u + v, u + v >$$

- Expand the RHS algebraically.
- Using Cauchy-Schwarz to make the equation an inequality.
- Collect terms to yield the Triangle Inequality.

**Your Turn:** Consider a set of vectors $W = v_1, ..., v_k$ in an inner product space $V$. Let $u$ be another vector in $V$. Show that there exist scalars $a_1, ..., a_k$ and a vector $v$ such that

$$u = a_1 v_1 + ... + a_k v_k + v$$

with

$$< v, v_i >= 0 \text{ for all i}$$

**Your Turn:** Consider the quadratic form $x'Px$. Explain why, if $P$ is a projection matrix, the form equals $||Px||^2$.

**Your Turn:** Explain why any set of orthogonal vectors is linearly independent.

**Your Turn:** Prove that the vector $Y - E(Y|X)$ is uncorrelated with $E(Y|X)$

**Your Turn:** For $X$ and $Y$ in $RV(\Omega)_0$, prove that

$$Var(Y) = E[Var(Y|X)] + Var[E(Y|X)],$$

first algebraically using Equation 10.5 and the relation $Var(R) = E(R^2) - (E(R))^2$, and then using the Pythagorean Theorem for a much quicker proof. As before, assume $X$ and $Y$ are centered.

**Your turn:** Show that Equation 10.3 holds.

**Your Turn:** Consider the space $RV(\Omega)_0$. In order for the claimed inner product to be valid, we must have that if $< X, X >= 0$, then $X$ must be the 0 vector. Prove this.

**Your Turn:** Say $V$ is $R^n$. Form the matrix $A$ whose columns are the $u_i$, and let $P = AA'$. Show that

$$Px =< x, u_1 > u_1 + ... + < x, u_k > u_k$$

so that $P$ thereby implements the projection.

**Your Turn:** Let $A$ be an $m$ x $n$ matrix. Consider the possible inner product on $R^n$ defined by $< x, y >= x'A'Ay$. State a condition on $A$ that is necessary and sufficient for the claimed inner product to be value, and prove this.

**Your Turn:** Show that for any vector $w$ and symmetric matrix $M$, the quadratic form $w'Mw \geq 0$.

**Your Turn:** Say in $C(0, 1)$ we want to approximate functions by polynomials. Specifically, for any $f$ in $C(0, 1)$, we want to find the closest polynomial of degree $m$ or less. Write functions to do this, with the following call forms:

```
gsc01(f,m)  # performs Gram-Schmidt and returns the result
bestpoly(f,gsout)  # approx. f by output from gsc01
```

Hint: Note that the set of polynomials of a given degree or less is a subspace of $C(0, 1)$. Also since the vectors here are functions, you'll need a data structure capable of storing functions. An R **list** will work well here.

**Your Turn:** We noted that projection matrices in $R^n$ are idempotent. Prove the converse, i.e. that any idempotent matrix $A$ must be a projection to some subspace. Hint: Try the column space of $A$.

# 11 Four Fundamental Spaces

ℹ Goals of this chapter:

This chapter will introduce four subspaces considered central to linear algebra.

## 11.1 The Four Fundamental Subspaces of a Matrix

Let us first define four fundamental subspaces for any $m$ x $n$ matrix $A$:

- row space$(A), \mathcal{R}(A)$: $\{x'A\}$ (all linear combinations of rows of $A$)
- column space$(A), \mathcal{C}(A)$: $\{Ax\}$ (all linear combinations of columns of $A$)
- null space$(A), \mathcal{N}(A)$: $\{x : Ax = 0\}$
- left null space$(A) = \mathcal{N}(A')$: $\{x : x'A = 0\}$

The null space is also called the *kernel* of $A$, viewing the matrix as a function from $\mathcal{R}^n$ to $\mathcal{R}^m$; what vectors are mapped to 0?

We will use 'dim()' to indicate vector space dimension, and 'rank()' for matrix rank. Following are a few important facts about these spaces.

**Theorem 11.1.**

$$rank(A) = dim(\mathcal{R}(A))$$

*Proof.* Parallel to the column space proof below. □

**Theorem 11.2.**

$$rank(A) = dim(\mathcal{C}(A))$$

*Proof.* Let $z_1, ..., z_r$ be a maximal linearly independent set of columns of $A$, where $r$ is the rank of that matrix. Then this set is in $\mathcal{C}(A)$, and in fact must span that subspace. If not, there would be some vector $x$ outside the span of $z_1, ..., z_r$, i.e. no linear combination of those vectors would equal $x$. Then $z_1, ..., z_r, x$ would be a linearly independent set, contradicting the maximal nature of the $z_i$.

In other words, the $z_i$ form a basis for $\mathcal{C}(A)$, and thus the dimension of that subspace is $r$.

□ □

**Theorem 11.3.** *The $\mathcal{C}(A)$ and $\mathcal{N}(A)$ subspaces are closely related:*

*(a)*

$$\mathcal{C}(A')^\perp = \mathcal{N}(A)$$

*(b)*

$$dim(\mathcal{C}(A')) + dim(\mathcal{N}(A)) = n$$

*Proof.* Say $w$ is in $\mathcal{N}(A)$. Using partitioning, we have

$$\left( \begin{array}{c} 0 \\ ... \\ 0 \end{array} \right) = Aw = \left( \begin{array}{r} a_1 w \\ ... \\ a_m w \end{array} \right)$$

where $a_i$ is row $i$ of $A$. Thus $w$ is orthogonal to the rows of $A$, thus to the row space of $A$. The latter is the column space of $A'$. Thus $\mathcal{N}(A)$ is a subset of $\mathcal{C}(A')^\perp$. This argument works exactly in reverse, so claim (a) is established.

Claim (b) follows from (a) by forming an orthonormal basis $q_1, ..., q_r$ for $\mathcal{C}(A')$, then extending it to one for all of $\mathcal{R}^m$, $q_1, ..., q_r, q_{r+1}, ..., q_m$. Claim (a) then implies that $q_{r+1}, ..., q_m$ is a basis for $\mathcal{N}(A)$.

□ □

# 12 Shrinkage Estimators

Goals of this chapter:

In the previous chapter, we introduced the norm of a vector. In many data science applications, solutions with smaller norms may be more accurate. This led to the devlopment of two widely-used data analytic tools, as we will see here.

## 12.1 Classic View of Shrinkage Estimators: Multicollinearity

The notion of *multicollinearity* was the original motivation for shrinkage estimators. It refers to settings in which the following concerns arise:

- One column of the matrix $A$ in Equation 5.10 is nearly equal to some linear combination of the others.
- Thus $A$ is nearly not of full rank.
- Thus $A'A$ is nearly not of full rank.
- Thus $\widehat{\beta}$ is unstable, in the form of high variance.

### 12.1.1 The Cause of multicollinearity

The matrix $A'A$ has a very important interpretation, both in the present context and others that will arise in later chapters:

$A'A$ as a Measure of Relations between the Predictor Variables

Roughly speaking, the row $i$, column $j$ element of $A'A$ is an indicator of the strength of the relation between predictor variables $i$ and $j$. If we center and scale our data (Section 5.6), then that element is exactly $n$ times the (sample) correlation between these two predictors.

To see this, recall the definition of (population) correlation, Equation 9.2. Substituting sample analogs for population quantities, we have the sample estimate,

$$\widehat{\rho} = \frac{\frac{1}{n}\sum_{m=1}^{n}(X_m - \overline{X})(Y_m - \overline{Y})}{s_X s_Y}$$

If we center and scale the data, the sample means become 0 and the sample standard deviations become 1, yielding

$$\widehat{\rho} = \frac{1}{n}\sum_{m=1}^{n}(X_m Y_m) \tag{12.1}$$

Now take $X$ and $Y$ to be columns $i$ and $j$ of $A$ in Equation 5.10. The element $(i,j)$ in $A'A$) is exactly Equation 12.1, without the $\frac{1}{n}$ factor.

In other words, multicollinearity arises in settings in which at least some of the predictor variables are highly correlated.

### 12.1.2 The Variance Inflation Factor

Earlier, we noted that due to multicollinearity, "$\widehat{\beta}$ is unstable, in the form of high variance." That point is often quantified by the *Variance Inflation Factor*. To motivate it, consider the "R-squared" value from linear regression analyis, which is the squared correlation between true "Y" and predicted "Y". Let $R_j^2$ denote that measure in the case of predicting column $j$ of $A$ from the other columns. The quantity

$$VIF_j = \frac{1}{1 - R_j^2}$$

then measures negative impact due to multicollinearity on estimating $\widehat{\beta}$. The intuition is that if, say, column 3 of $A$ can be predicted well using a linear model, then that column is approximately equal to a linear combination of the other "X" columns. This is worrisome in light of the problems described above.

Needless to say, the word “nearly” above, e.g. in “nearly not of full rank,” is vague, and leaves open the question of “What can we do about it?” We will present several answers to these questions in this and the succeeding chapters.

## 12.2 Example: Million Song Dataset

Let’s consider the Million Song Dataset, varous versions of which are on the Web.

Ours is a 50,000-line subset of the one with 515345 rows and 91 columns. The first column is the year of release, followed by 90 columns of various audio measurements. The goal is to predict the year, **V1**, from the audio variables **V2** through **V91**.

The function **regclass::VIF** will compute the VIF values for us.

```
library(WackyData)
data(MillSong50K) # loads s50
lmout <- lm(V1 ~ .,data=s50)
library(regclass)
VIF(lmout)
```

```
      V2       V3       V4       V5       V6       V7       V8       V9
3.215081 2.596474 4.236853 7.414375 1.534492 5.844729 2.794018 3.192805
     V10      V11      V12      V13      V14      V15      V16      V17
2.072851 3.673590 4.705886 1.708520 2.446279 2.827296 3.672250 7.409782
     V18      V19      V20      V21      V22      V23      V24      V25
2.634033 9.472261 4.217311 7.147952 5.122114 7.984860 9.675134 3.591586
     V26      V27      V28      V29      V30      V31      V32      V33
1.818596 1.758043 3.879968 1.663670 2.108174 2.321385 2.056017 1.854913
     V34      V35      V36      V37      V38      V39      V40      V41
3.011534 2.040587 2.760111 2.879667 1.918229 2.176048 2.074103 1.946859
     V42      V43      V44      V45      V46      V47      V48      V49
1.704259 2.138794 1.690651 1.556782 1.817380 3.000759 1.547592 2.140282
     V50      V51      V52      V53      V54      V55      V56      V57
2.496121 1.612253 2.042571 2.208492 1.723669 2.024290 2.016403 2.033654
     V58      V59      V60      V61      V62      V63      V64      V65
2.004260 2.998469 2.074152 3.410124 2.153116 1.378160 3.270617 1.502543
```

```
     V66      V67      V68      V69      V70      V71      V72      V73
2.581171 1.725809 2.167673 2.379354 2.062862 1.703360 2.036596 1.984427
     V74      V75      V76      V77      V78      V79      V80      V81
2.557163 1.465020 1.515436 2.260728 1.840509 2.078497 3.604771 1.595064
     V82      V83      V84      V85      V86      V87      V88      V89
2.528307 2.005876 2.283956 1.448379 1.895053 1.601004 1.581099 2.252362
     V90      V91
1.332590 1.570891
```

As a rough guide, values of VIF about 5.0 are considered concerning by many analysts. Under that criterion, variables **V5**, **V7**, **V17** and so on look troublesome.

What can be done? One simple approach would be to delete those columns from the dataset. This is indeed is a common solution, but another is *ridge regression*, which we present next.

## 12.3 Ridge Regression

In a seminal paper, Hoerl and Kennard presented a new approach to the problem of multicollinearity of predictor variables in a linear model.

*Technometrics*, February 1970

### 12.3.1 The ridge solution

Their solution is simple: Add some quantity to the diagonal of $A'A$. Specifically, Equation 5.10 now becomes

$$\widehat{\beta} = (A'A + \lambda I)^{-1} A'S \tag{12.2}$$

where $\lambda$ is a positive number chosen by the analyst. Here $A$ has dimensions $n \times p$, and $I$ is the $p \times p$ identity matrix.

Why do this? Say some linear combination of the rows of $A'A$ is nearly 0. The same linear combination of the rows of $I$ will be nonzero, so we avoid near-singularity.

## 12.3.2 Matrix formulation

Using partitioned matrices helps understand ridge. Replace $A$ and $S$ by

$$A_{new} = \begin{pmatrix} A \\ \lambda^{0.5} I \end{pmatrix}$$

and

$$S_{new} = \begin{pmatrix} S \\ 0 \end{pmatrix}$$

where 0 means a vector consisting of $p$ 0s. In essence, we are adding artificial data here, consisting of $p$ new rows to $A$, and $p$ new elements to the vector $S$. So Equation 12.2 is just the result of applying Equation 5.10 to $A_{new}$ and $S_{new}$. For example, using matrix partitioning, we have

$$A'_{new} A_{new} = (A' | \lambda^{0.5} I) \begin{pmatrix} A \\ \lambda^{0.5} I \end{pmatrix} = A'A + \lambda I$$

Loosely speaking, we can think of the addition of $\lambda I$ to $A'A$ as making the latter “larger”, and thus making its inverse smaller. In other words, we are “shrinking” $\widehat{\beta}$ towards 0, hence the title of this chapter, Shrinkage Estimators. This effect is made even stronger by the fact that we added 0s data to $S$. This will be made more precise below.

## 12.3.3 Example: Million Song dataset

We will use **glmnet**, one of the most widely-used R packages.

```
library(glmnet)
x <- s50[,-1]
y <- s50[,1]
glmOut <- glmnet(x,y,
            alpha=0,  # ridge
            lambda=0.1)
coef(glmOut)
```

```
91 x 1 sparse Matrix of class "dgCMatrix"
                        s0
(Intercept)  1.952611e+03
V2           8.573258e-01
V3          -5.589618e-02
V4          -4.521365e-02
V5           8.939743e-04
V6          -9.630525e-03
V7          -2.075296e-01
V8          -4.753624e-03
V9          -9.733144e-02
V10         -6.383153e-02
V11          2.778348e-02
V12         -1.486791e-01
V13         -1.326579e-02
V14          4.756877e-02
V15          3.197157e-04
V16         -4.765018e-04
V17          4.951670e-04
V18          5.276124e-04
V19          1.254372e-03
V20          1.518736e-03
V21          2.206416e-03
V22         -4.304329e-04
V23          5.792964e-04
V24          7.717453e-03
V25          3.205090e-03
V26         -3.527333e-03
V27          4.785205e-05
V28          1.510325e-03
V29          2.996810e-04
V30          6.384049e-04
V31         -2.560997e-04
V32         -4.861561e-04
V33         -6.250266e-04
V34         -3.757523e-03
V35          3.563733e-04
V36          1.288622e-03
V37         -4.417041e-03
V38         -2.502325e-04
V39          9.405005e-04
```

```
V40           1.490186e-03
V41          -1.534097e-03
V42          -1.510983e-03
V43          -1.777780e-03
V44          -1.814201e-03
V45          -1.865966e-03
V46          -1.107727e-03
V47           5.906898e-03
V48           6.578124e-04
V49          -2.040170e-03
V50           4.795372e-04
V51           1.162657e-03
V52           6.164815e-04
V53          -9.613775e-04
V54           1.571230e-03
V55          -1.230929e-03
V56          -1.395143e-03
V57           2.158453e-04
V58          -1.975641e-03
V59           2.039760e-03
V60          -1.302405e-03
V61           6.875221e-04
V62          -3.480975e-03
V63          -3.285816e-03
V64          -9.199332e-03
V65           1.283424e-03
V66          -1.444237e-03
V67          -5.957302e-05
V68           1.139299e-03
V69          -9.805816e-04
V70          -3.734949e-03
V71          -5.127353e-03
V72          -1.071399e-03
V73           1.808175e-04
V74          -1.034781e-05
V75           4.315147e-03
V76           3.382526e-03
V77           1.111779e-02
V78           3.162072e-04
V79          -4.565407e-03
V80           3.729735e-05
```

```
V81          2.362107e-04
V82         -9.349464e-04
V83         -2.846584e-04
V84          1.439897e-03
V85          1.360484e-03
V86          2.442744e-02
V87         -6.838186e-04
V88          8.555403e-04
V89         -3.329155e-02
V90         -1.793489e-03
V91         -3.614719e-05
```

We had earlier flagged variable $V5$ as causing multicollinearity. As noted then, we could simply exclude it, but here under ridge, we see that it has been assigned a very small regression coefficient compared to many of the others.

So, did the estimated coefficient vector shrink?

```r
l2norm <- function(x) sqrt(sum(x^2))
bh01<- coef(glmOut)
l2norm(bh01)
```

```
[1] 1952.612
```

```r
bhOLS <- coef(lm(V1 ~ .,s50)) # OLS means Ordinary Least Squares
l2norm(bhOLS)
```

```
[1] 1951.028
```

No, the vector got larger!

The culprit is the intercept term, $\hat{\beta}_0$:

```r
bh01[1]
```

```
[1] 1952.611
```

```r
bhOLS[1]
```

```
(Intercept)
   1951.028
```

```r
l2norm(bh01[-1])
```

```
[1] 0.9080075
```

```r
l2norm(bhOLS[-1])
```

```
[1] 0.9451025
```

The rest of the vector did shrink (though not necessily element-by-element).

Actually, we should have centered and scaled "X" before applying ridge, since the predictors are of such different magnitudes. This also makes the intercept 0.

```r
s50a <- s50
s50a[,-1] <- scale(s50[,-1])
s50a[,1] <- s50a[,1] - mean(s50a[,1])
bh01<- coef(glmnet(s50a[,-1],s50a[,1],alpha=0,lambda=0.1))
l2norm(bh01)
```

```
[1] 7.570367
```

```r
bhOLS <- coef(lm(V1 ~ .,s50a))
l2norm(bhOLS)
```

```
[1] 7.886902
```

Again, the coefficients vector shrank.

## 12.4 Choosing the Value of $\lambda$

So, how do we choose $\lambda$? We need to note a very important principle first.

## 12.5 P-hacking, Both in Hypothesis Testing and Generally

"In a set of 10,000 randomly typing monkeys, one of them will accidentally type a Shakespearian sonnet."

Some readers have heard this before in the context of *p-hacking*, in which an analyst poses a large collection of research questions, and runs a statistical hypothesis test on each of them. Even if the null hypothesis is true for all of them, each will have a 5% chance of being rejected, and since there are many, the chance is high that at least one will be rejected and declared "significant," even if $H_0$ is true.

One should not be doing hypothesis tests in the first place. See these notes.

This problem arises in many, many Data Science contexts, and a good analyst must be vigilant to recognize the potential to mislead. In our context here, if we look at many values of $\lambda$, one of them may by accident look very promising, e.g. result in very good prediction accuracy on our training set yet actually overfit.

## 12.6 Cross-validation

A common way to choose among models is *cross-validation*: We set aside a subset of the data, known as the *holdout* or *test* set, for use in assessing predictive accuracy. The remaining data is the *training set*. For each of our competing models – in this case, competing values of $\lambda$ – we fit the model on the training set, then use the result to predict the test set. We then use whichever model – again, in this case whichever value of $\lambda$ – does best in the test set.

The test set serves as “fresh data,” simulating how our fitted model might do in the real world (assuming our data is representative of the real world). Predicting on the training set is not as good, since our fit was by design tailored to that data.

We still run the risk of p-hacking, but cross-validation works well as long as we keep the problem in mind. This topic will come up in future chapters as well.

## 12.7 Example: Million Song Data

```
> library(glmnet)
> glmOut <- cv.glmnet(x=as.matrix(s50[,-1]),y=s50$V1,alpha=0)
> glmOut
    Lambda Index Measure     SE Nonzero
min 0.2474   100   89.44 0.9201      90
1se 0.9987    85   90.28 0.9823      90
```

The $\lambda$ value that gave the smallest Mean Squared Prediction Error (MSE) was 0.2474. (Coincidentally, it was also the smallest value that the function tried; see **glmOut$lambda**.)

A more conservative value of $\lambda$ was 0.9987, the largest $\lambda$ giving MSE within one standard error of the minimum; it's conservative in the sense of being less likely to overfit (p-hacking); its MSE value, 90.28, was only slightly larger than the best one.

In each case, all 90 predictors had nonzero coefficient estimates. This part of the output will usually not be very meaningful for the ridge case, and is aimed mainly at the LASSO, to be presented later in this chapter.

We can then predict as usual. Say we have a song similar to that in **s50[1,]**, but with **V2** equal to 25.0.

```
z <- s50[1,-1]  # exclude Y
z[1,1] <- 25.0
predict(glmOut,z)
```

```
          s0
[1,] 2003.235
```

The year of release is predicted to be 2003.

### 12.7.1 Example: Census data

One can even apply ridge to situations of **exact** dependence among the columns of X, as opposed to the original motivation of dealing with *approximate* linear dependence.

We saw such a setting in Section 7.1. There we deliberately induced exact linear dependence by inclusion of both male and female dummy variables. Let's apply ridge:

```
library(qeML)
data(svcensus)
svc <- svcensus[,c(1,4:6)]
# force having both male and female columns
svc$man <- as.numeric(svc$gender == 'male')
svc$woman <- as.numeric(svc$gender == 'female')
svc$gender <- NULL
a <- svc[,-2]  # the A matrix
a <- as.matrix(a)
lambda <- 0.1
# apply ridge procedure
betahat <- solve(t(a) %*% a + lambda * diag(4))
betahat %*% t(a) %*% as.matrix(svc$wageinc)
```

```
               [,1]
age        496.6716
wkswrkd   1372.7052
man     -18677.8751
woman   -29378.3086
```

The results essentially are the same as what we obtained by having only one dummy, thus no linear dependence: Men still enjoy about an $11,000 advantage. But ridge allowed us to avoid deleting one of our dummies. Such deletion is easy in this case, but for large $p$, say in the hundreds or even more, some analysts prefer the convenience of ridge.

## 12.8 Modern View of Shrinkage Estimators

There are many ways to deal with multicollinearity other than shrinkage, and indeed, these days discussions of shrinkage seldom mention multicollinearity. Instead, the goal of shrinkage is *dimension reduction*, meaning to reduce the complexity of a model in order to avoid overfitting. The LASSO, introduced below, does this explicitly, while ridge accomplishes it via pure size reduction.

We will discuss this further in Section 12.11. Let's start by addressing the point of what we really mean by "shrinkage."

## 12.9 Formalizing the Notion of Shrinkage

How in the world did statisticians develop an interest in shrinking estimators? A watershed event occurred in the early 1980s, when the statistical world was shocked by research by James and Stein that found, in short, that:

Say $W$ has $q$-dimensional normal distribution with mean vector $\mu$ and independent components having variance $\sigma^2$, each. We have a random sample of size $n$, i.e. $n$ independent observations on $W$. Then if $q \geq 3$, in terms of Mean Squared Estimation Error, the best estimator of $\mu$ is NOT the sample mean $\bar{W}$. Instead, it's a shrunken version of $\bar{W}$,

$$\left(1 - \frac{(q-2)\sigma^2/n}{||\bar{W}||^2}\right)\bar{W} \tag{12.3}$$

The quantity within the parentheses is typically smaller than 1, giving us the shrinkage property. Note, though, that with larger $n$, the amount of shrinkage is minor.

In the case of linear regression, a version of shrinkage works there too, with $q$ being the number of columns in the $A$ matrix.

So, let's view the issue of shrinkage more formally, first for ridge and later for the LASSO.

### 12.9.1 Shrinkage through length penalization

Say instead of minimizing Equation 5.7, we minimize

$$(S - Ab)'(S - Ab) + \lambda ||b||^2 \tag{12.4}$$

We say that we *penalize* large values of $b$, an indirect way of pursuing shrinkage. Now take the derivative and set to 0:

$$0 = A'(S - Ab) + \lambda b \tag{12.5}$$

i.e.

$$(A'A + \lambda I)b = A'S$$

and thus

$$\hat{\beta} = (A'A + \lambda I)^{-1} A'S$$

It's ridge! Exactly what we had in Equation 12.2. So ridge, originally motivated by "almost singular" settings, also turns out to be justified as a shrinkage estimator..

### 12.9.2 Shrinkage through length limitation

Instead of penalizing $||b||$, we could simply constrain it, i.e. we could set our optimization problem to:

minimize $(S - Ab)'(S - Ab)$, subject to the constraint $||b||^2 \leq \gamma$

We say that this new formulation is the *dual* of the first one. One can show that they are typically equivalent.

## 12.10 The LASSO

The LASSO (Least Absolute Shrinkage and Selection Operator) was developed by Robert Tibshirani in 1996, following earlier work by Leo Breiman and Tin Kam Ho. It takes $\widehat{\beta}$ to be the value of $b$ that minimizes

$$(S - Ab)'(S - Ab) + \lambda ||b||_1 \tag{12.6}$$

where the "l1 norm" is

$$||b|| = \sum_{i=1}^{p} |b_i|$$

We will write our original norm as $||b||_2$.

This is a seemingly minor change, but with important implications. What the researchers were trying to do was to obtain a *sparse* $\widehat{\beta}$ , i.e. a solution with lots of 0s, thereby providing a method for predictor variable selection. This is important because so-called "parsimonious" prediction models are desirable.

### 12.10.1 Properties

To that end, as noted above, it can be shown that, under some technical conditions, that the ridge solution minimizes

$$(S - Ab)'(S - Ab)$$

subject to the constraint

$$||b||_2 \leq \gamma$$

while in the LASSO case the constraint is

$$||b||_1 \leq \gamma$$

As with $\lambda$ in the original formulation, $\gamma$ is a positive number chosen by the analyst.

## 12.11 Ridge vs. LASSO for Dimension Reduction

Today's large datasets being so common, we need a way to "cut things down to size," i.e. *dimension reduction*, aimed at reducing the number of predictor variables. This is done both for the sake of simplicity and to avoid overfitting, in which fitting an overly complex model can reduce predictive power.

### 12.11.1 Geometric view

Comparison between the ridge and LASSO concepts is often done via this graph depicting the LASSO setting:

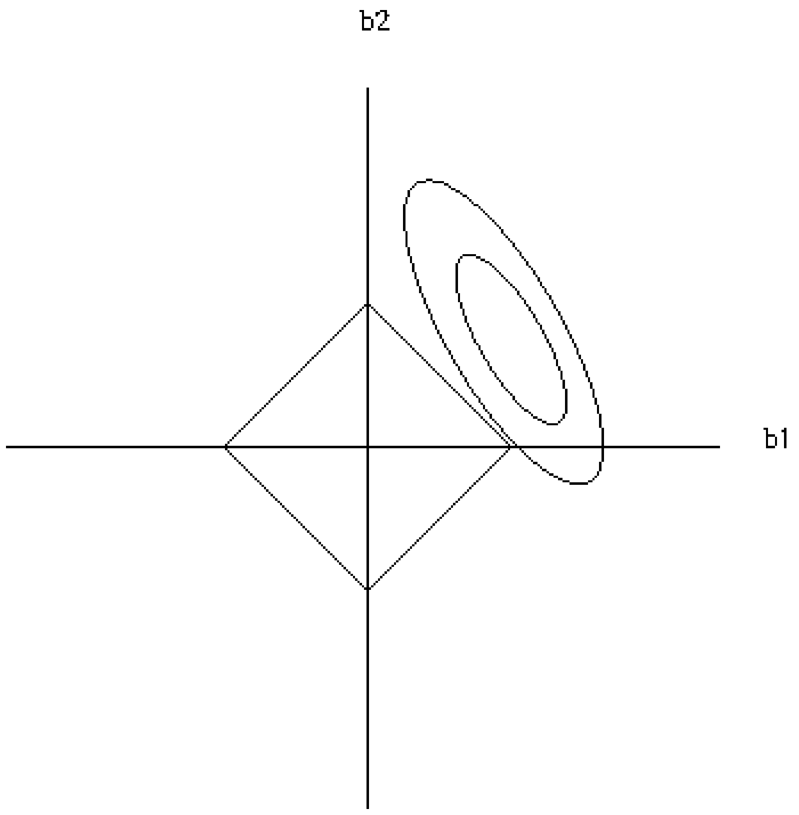


Figure 12.1: LASSO sparsity

Here $p = 2$, with $b = (b_1, b_2)'$. The horizontal and vertical axes represent $b_1$ and $b_2$. Which point in the graph will be the LASSO solution $(b_1, b_2)' = (\widehat{\beta}_{1,LASSO}, \widehat{\beta}_{2,LASSO})'$?

Reason as follows.

- The constraint $||b||_1 \leq \gamma$ then takes the form of a diamond, with corners at $(\gamma, 0)$, $(0, \gamma)$, $(-\gamma, 0)$ and $(0, -\gamma)$. The constraint $||b||_1 \leq \gamma$ requires us to choose a point $b$ somewhere in the diamond, including the boundary.
- The concentric ellipses depict the values of $c(b) = (S - Ab)'(S - Ab)$, as follows.
    - Consider one particular value of $c(b)$, say 1.68.

  - Many different points $b$ in the graph will have $c(b) = 1.68$; in fact, the locus of all such points is an ellipse.
  - There is one ellipse for each possible value of $c(b)$. So, there are infinitely many ellipses, though only two are shown here.
  - Larger values of $c(b)$ yield larger ellipses.
  - By the way, the common center of these ellipses is the ordinary (i.e. non-shrunken) least squares (OLS) solution $\widehat{\beta}_{OLS}$, and the smallest ellipse has c(b) equal to the OLS sum of squares.

- On the one hand, we want to choose a $b$ for which $c(b)$ – our total squared prediction error – is small, thus a smaller ellipse.
- But on the other hand, we need at least one point on the ellipse to be in common with the diamond.
- The solution is then a point $b$ in which the ellipse just barely touches the diamond.
- Picture in your mind an ellipse, say the inner one in the graph, growing larger and larger, while retaining the same center and orientation, until it hits the diamond. That is the outer ellipse, which indeed hits the diamond at one of the corners.
- Note that each of the four corners of the diamond represents a sparse solution. For instance, the point $(0, \gamma)$ has $b_1 = 0$.
- Then picture other ellipses, at other centers with other orientations, and go through the same process in your mind's eye. You will see that typically the solution turns out to be one of the corners. Again, this is important because it gives us a sparse solution.

Of course, we are in just two dimensions here. With $p = 10$ predictor variables, the graph would be in 10 dimensions, beyond our human ability to picture. But we still would have a diamond in that space, with $2p$ corners etc.

### 12.11.2 Implication for dimension reduction.

Again, the key point is that that "barely touching" point will be one of the four corners of the diamond, points at which either $b_1 = 0$ or $b_2 = 0$ – *hence a sparse solution*, meaning one in which many/most of the coefficients in the fitted model will be 0. This achieves the goal of dimension reduction.

Ridge will not produce a sparse solution. The diamond would now be a circle (not shown). The "barely touching point" will almost certainly will be at a place in which both $b_1$ and $b_2$ are nonzero. Hence no sparsity.

### 12.11.3 Avoidance of overfitting *without* dimension reduction

As we've seen, both ridge and LASSO reduce the size of the $\widehat{\beta}$ vector of estimated coefficients. Smaller quantities have smaller statistical variances, hence a guard against overfitting. So, ridge can be employed as an approach to the overfitting problem, even though it does not provide a sparse solution.

Moreover, in some settings, it may be desirable to keep all predictors, as seen in the next section.

## 12.12 Example: NYC Taxi Data

The purpose of this data is to predict trip time in the New York City taxi system. The **qeML** package includes a 10,000-row subset.

```
library(qeML)
data(nyctaxi)
head(nyctaxi)
```

```
        trip_distance PULocationID DOLocationID tripTime DayOfWeek
2969561          1.37          236           43      598         1
7301968          0.71          238          238      224         4
3556729          2.80          100          263      761         3
```

```
7309631          2.62          161          249       888          4
3893911          1.20          236          163       648          5
4108506          2.40          161          164       977          5
```

```
dim(nyctaxi)
```

```
[1] 10000     5
```

```
length(unique(nyctaxi$PULocationID))
```

```
[1] 143
```

```
length(unique(nyctaxi$DOLocationID))
```

```
[1] 205
```

If we fit, say, a linear model, **lm** will form a dummy variable for each of the pickup and dropoff locations. Thus we will have $p = 1+143+205+1 = 350$. An old rule of thumb says that if we have $p$ predictors and $n$ data points, we should keep $p < \sqrt{n}$ to avoid overfitting. As we will see in a later chapter, these days that rule is being questioned, but it is still useful. Since here we have $\sqrt{n} = 100$, there is a strong suggestion that we do some dimension reduction.

The problem would be even worse if we add pickup/dropoff location interaction variables, basically products of the pickup and dropoff dummy variables.

Thus we either should delete some of the pickup and dropoff variables, or use all of them but temper the fit using ridge. The latter may be more attractive, as riders would like a time estimate for their particular pickup and dropoff locations.

```
library(glmnet)
nycwide <- factorsToDummies(nyctaxi[,-1])
glmOut <- cv.glmnet(x=nycwide,y=nyctaxi[,1],alpha=0)
glmOut
```

```
Call:  cv.glmnet(x = nycwide, y = nyctaxi[, 1], alpha = 0)

Measure: Mean-Squared Error

    Lambda Index Measure     SE Nonzero
min 0.3002   100   2.983 0.1467     356
1se 0.9169    88   3.110 0.1551     356
```

The best $\lambda$ value was found to be 0.3002. But as noted above, we might use the more conservative value, 1.0063, to try to avoid p-hacking.

## 12.13 Iterative Calculation

An advantage of ridge over LASSO and other l1 shrinkage estimators is that the former has an explicit (we say *closed-form*) solution, which is not the case for the LASSO. In fact, as will be seen often in this book, many algorithms in statistics/machine learning lack closed-form solutions, in which case one must resort to *iterative* computation.

These means we make a series of guesses as the to value of the desired quantity, hopefully each more accurate than the last, eventually settling on a final guess.

This may or may not work well. Here are some of the major issues/perils:

- choice of initial guess
- updating method
- learning rate
- convergence
- presence or lack of calculus derivatives

The basic idea is to first (somehow) make some guess as to the value of the desired quantity, say $\widehat{\beta}$ in the LASSO. The algorithm crunches this to make a new, updated guess, hopefully one that is more accurate than the first. One then updates the new guess, continuing this process until, hopefully, it *converges*, meaning that it no longer changes much from one iteration to the next. The current guess is then deemed to be the correct value.

The case of computation of the LASSO is further complicated by its being based on the $l_1$ norm, which in turn uses absolute values, i.e. $|x|$. These have no derivative in the calculus sense, say as used in Equation 12.5 for ridge. This is a problem because many iterative methods are based on derivatives, as follows.

Say we have a function $f$ whose root $r$ is of interest to us. We might make a series of guesses for $r$ by considering the derivative $f'$. This is illustrated in the figure below. Unknown to us, $r = 2$. Our current guess is $x = 3$. We draw $f'$, i.e. the tangent line to the curve at our current guess, and temporarily pretend that the line is the curve. We thus compute the root for the line, which is seen here to be near 2.0, and then take this tangent root as our updated guess for the root of the curve.

Our quest for a root may arise for instance in a minimization problem, where we set a derivative to 0 and solve for that root.

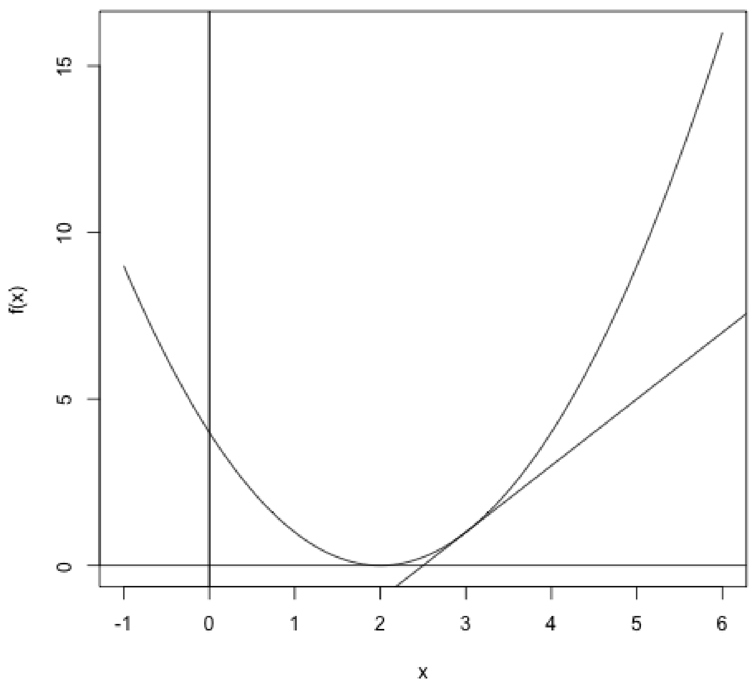


Figure 12.2: root hunting

Some machine learning algorithms have a parameter called the *learning rate*, which is motivated by a concern that the process may overshoot the root, or converge to the wrong root. . A smaller learning rate value directs the algorithm to take smaller steps in generating new guesses. In this case, we might go only partway to the tangent root. On the one hand, this can slow the computation but on the other, we may be less likely to overshoot the true value. On the other hand, if we are concerned about settling on the wrong root, we might set a large value for the learning rate.

Again, the desired root may arise in a minimum/maximum problem. There could be many local mimima/maxima, which we use to avoid.

At any rate, if the quantity $f$ that we are working with does not have a derivative, our work is extra difficult.

At first one may think that such internal details of computation need not concern the end user of the software. But the fact is that often an algorithm will fail to converge, and the user will need to get more directly involved, say in trying a different value for the initial guess.

So, if say **glmnet** fails to converge, what can be done? For example, in **glmnet**, the argument **thresh** defines what we meant by our phrasing "no longer changes much"; we can decrease or even increase that value. One can make sure to center and scale the X data. Tweaking other parameters may help as well, such as changing the updating method. (In fitting neural network models, there are actually many different methods to choose from.) But in the end, there are no magic solutions. It may well be that one's basic model is flawed.

## 12.14 The Kernel Trick and Kernel Ridge Regression

How often do you see a named trick in a math book? Well, there is indeed one here, one that is of great practical value.

### 12.14.1 Polynomial Regression

The world is not linear, not even approximately so. Thus early in the development of statistics, analysts started using polynomial models. To predict human weight from height, for instance, one might fit the model

$$E(\text{weight}|\text{height}) = \beta_0 + \beta_1\text{height} + \beta_2\text{height}^2$$

The key point is that *this is still a linear model.* Though it is a nonlinear function of height, it is linear in the $\beta_i$. If say, we multiple all the $\beta_i$ by 2, the value of the above expression is doubled. In Equation 5.6, we now would tack on a column consisting of the $V_i^2$, but $E(C|V)$ would still be a linear expression in $\beta$:

$$E(C|V) = A\beta = \beta_0 + \beta_1\text{V} + \beta_2\text{V}^2$$

Nice, but the size of our model grows rapidly. Say we predict weight from not only height but also age. Then our matrix $A$ will have columns not only for height and height-squared, but also age and age-squared, as well as height times age. In my machine learning book , there is an example in which $A$ originally had 54 columns, but with a polynomial model of degree 2, the number of columns grew to 1,564!

N. Matloff, *The Art of Machine Learning*, NSP, 2023

This is especially problematic in the case $p > n$, which is increasingly common these days, and motivates the following material.

### 12.14.2 The Kernel Trick

We will first introduce a computational shortcut, and then the Kernel Trick itself. Our context will be ridge regression.

Using some algebraic manipulation, Equation 12.2 can be shown to be equivalent to

$$\hat{\beta} = A'(AA' + \lambda I)^{-1} S$$

Note that the product $AA'$ will be $n \times n$, as opposed to our original $p \times p$ product $A'A$. With $p > n$, this change means a large saving in memory storage space and computation time.

But we reap even larger benefit by looking at fitted or predicted values. Let $x_{new}$ be a matrix of "X" values at which we wish to predict "Y," in the same format as the rows of $X$. Then the fitted/predicted values are

$$x_{new}\hat{\beta} = x_{new}A'(AA' + \lambda I)^{-1} = [x_{new}A'][(AA' + \lambda I)^{-1}]$$

What is so special about this expression, or more specifically, the two bracketed expressions? The answer is that every major computation here is a dot product:

- In $x_{new}A'$, we are taking dot products of rows of $x_{new}$ and rows of $A$.

- In $AA'$, we are taking dot products of rows of $A$ and rows of $A$.

The idea of the Kernel Trick is to replace dot products by kernel computations. In computing $AA'$, for example, denote row $i$ of $A$ by $a_i$. Then we replace the expression for the row $i$, column $j$ element of $AA'$,

$$a_i a_j' \tag{12.7}$$

by

$$k(a_i, a_j)$$

Here $k$ is a *kernel function* satisfying

- $k$ is a symmetric function in its arguments, and
- $k$ is *nonnegative definite*.

  The latter is a generalization of nonnegative definite matrices. If we have a set of vectors $u_1, ..., u_r$ and form the matrix having its row $i$, column $j$ value equal to $k(u_i, u_j)$, then the matrix is required to be nonnegative definite.

What does that give us? Suppose we wish to do polynomial regression of degree 2. As noted above, we could add the appropriate columns to the $A$ matrix, thus applying the transformation (e.g. for the case $p = 2$),

$$x \rightarrow \phi(x) = (x_1, x_1^2, x_2, x_2^2, x_1 x_2) \tag{12.8}$$

We say that $\phi$ *lifts* us from our *ambient* space (i.e. original space, dimension $p$) to a desired higher-dimensional space, the *latent* space.

But again, this would increase the dimensionality, going from 2 to 5 in this $p = 2$ case, but the main point for now is that we still would be computing dot products. For instance, the new value of Equation 12.7 would now be

$$\phi(a_i)\phi(a_j)'$$

The Kernel Trick involves finding a kernel $k$ for which

$$k(u,v) = \phi(u)'\phi(v)$$

It turns out that such a kernel is

$$k(u,v) = (c + u'v)^d \tag{12.9}$$

for polynomials of degree $d$, where $c$ is a constant chosen by the user.

In other words:

> The Kernel Trick enables us to stay in the ambient space yet still achieve the nonlinearity of the lifted space.

In fact, although it turns out that Equation 12.9 corresponds to

$$\phi(t) = (t_1^2, t_2^2, \sqrt{2}\ t_1 t_2, \sqrt{2c}\ t_1, \sqrt{2c}\ t_2, c)$$

not only do we avoid computing $\phi(t)$, we don't even need to know its form.

Note again that all this worked only because our quantities of interest here consist of dot products.

### 12.14.3 Kernel Ridge Regression: the Code

```
# kernel ridge regression

# fits 'data' with KRR, ridge parameter 'lamb' and kernel 'kern', and
# then predicts "Y" from 'newX'

krr <- function(data,yName,lamb,kern,newX)
```

```
{
   if (!allNumeric(data)) stop("'data' must be all numeric")

   ycol <- which(names(data) == yName)
   data <- as.matrix(data)

   x <- data[,-ycol,drop=FALSE]
   x <- cbind(1,x)
   n <- nrow(x)
   p <- ncol(x)

   newX <- as.matrix(newX)
   if (ncol(newX) == 1) newX <- t(newX)
   newX <- cbind(1,newX)

   part1 <- kernAB(newX,t(x),kern)
   part2 <- kernAB(x,t(x),kern)
   part2 <- solve(part2 + lamb * diag(n)) %*% y

   predsKRR <- part1 %*% part2
   return(cbind(predsKRR))
}

# finds the matrix product ab, but with the kernel k evaluated at each
# entry

kernAB <- function(a,b,k)
{
   ab <- matrix(nrow=nrow(a),ncol=ncol(b))
   for (i in 1:nrow(a)) {
      arow <- a[i,]
      for (j in 1:ncol(b)) {
         ab[i,j] <- k(arow,b[,j])
      }
   }
   ab
}
```

## 12.15 A Warning

Many statistical quantities now have *regularized*, i.e. shrunken versions. It is also standard practice in neural networks. This may be quite helpful in prediction contexts. However, note the following:

! No Statistical Inference on Shrinkage Estimators

Shrinkage produces a bias, of unknown size. Thus classical statistical inference (confidence intervals, hypothesis tests) is not possible.

## 12.16 Your Turn

**Your Turn:** Show that $A_{new}$ in Section 12.3.2 is of full rank, $p$.

**Your Turn:** In Section 12.12, it was pointed out that in some settings we may prefer to retain all of our predictor variables, rather than do dimension reduction, thus preferring ridge to LASSO. But we might pay a price for that preference, in that the LASSO may actually give us better predictive power. Write an R function to investigate this, with call form

```
compareRidgeLASSO(data,yName)
```

where **data** and **yName** are in the format of the predictive $qeML$ functions, and the minimum Mean Squared Prediction Error is returned for both algorithms. Try your function on some of our datasets, or others.

**Your Turn:** The LASSO will tend to produce solutions with lesser sparsity if the dataset is large. Write an R function to illustrate this, with call form

```
dependN(data,yName,n=seq(1,nrow(data),100,nReps=1)
```

where: where **data** and **yName** are in the format of the predictive *qeML* functions; the LASSO is applied to **n** randomly chosen rows of **data**; and **nReps** is the number of replicates to run at each value of **n**. The function will compute the number of nonzero elements in the $\widehat{\beta}$ chosen by cross-validation in **cv.glmnet**. Try your code on a few datasets.

**Your Turn:** Consider a generalization of ridge regression, in which we find

$$\text{argmin}_b \; ||S - Ab||^2 + ||Db||^2$$

for a diagonal matrix $D$. The idea is to allow different shrinkage parameters for different predictor variables. Show that

$$b = \left[A'A + D^2\right]^{-1} A'S$$

# 13 Eigenanalysis

Goals of this chapter:

The concept of eigenvalues and eigenvectors is one of the most important of all applications of linear algebra to data science. We introduce the basic ideas and properties in this chapter.

## 13.1 Example: African Soils Data

To get things started, let's consider the African Soils dataset.

Let's take a look around:

```
library(WackyData)
data(AfricanSoil)
dim(AfricanSoil)
```

```
[1] 1157 3600
```

```
names(AfricanSoil)[1:25]
```

```
 [1] "PIDN"     "m7497.96" "m7496.04" "m7494.11" "m7492.18" "m7490.25"
 [7] "m7488.32" "m7486.39" "m7484.46" "m7482.54" "m7480.61" "m7478.68"
[13] "m7476.75" "m7474.82" "m7472.89" "m7470.97" "m7469.04" "m7467.11"
[19] "m7465.18" "m7463.25" "m7461.32" "m7459.39" "m7457.47" "m7455.54"
[25] "m7453.61"
```

```
names(AfricanSoil)[3576:3600]
```

```
 [1] "m605.545" "m603.617" "m601.688" "m599.76"  "BSAN"     "BSAS"
 [7] "BSAV"     "CTI"      "ELEV"     "EVI"      "LSTD"     "LSTN"
[13] "REF1"     "REF2"     "REF3"     "REF7"     "RELI"     "TMAP"
[19] "TMFI"     "Depth"    "Ca"       "P"        "pH"       "SOC"
[25] "Sand"
```

Let's try predicting **pH**, the acidity. But that leaves 3599 possible predictors. As mentioned in an earlier chapter, there is an old rule of thumb that one should have $p < \sqrt{n}$, for $p$ predictors and $n$ data points, to avoid overfitting, a rule which in our setting of $n = 1157$ is grossly violated. We need to do dimension reduction. One way to accomplish this is to use Principal Components Analysis (PCA).

## 13.2 Overall Idea

The goal is to find a few important linear combinations of our original predictor variables–important in the sense that they roughly summarize our data. These new variables are called the *principal components* (PCs) of the data. PCA will be covered in detail in the next chapter, but our preview here will set the stage for important general concepts that we will develop in the current chapter.

### 13.2.1 The first PC

Let **X** denote a set of variables of interest (not necessarily in a prediction context). For concreteness, consider the **mtcars** dataset that comes with R.

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

In searching for good linear combinations $Xu$, we want to aim for ones with high variance. We certainly don't want ones with low variance; after all, a random variable with 0 variance is a constant. So we wish to find linear combinations

$$Xu$$

which maximize

$$Var(Xu) = u'Cov(X)u$$

where we have invoked Equation 4.6.

But that goal is ill-defined, since we could take larger and larger vectors $u$, thus larger and larger vectors $Xu$, no maximum variance. So, let's constrain it to vectors $u$ of length 1:

$$u'u = 1$$

Let's use the method of Lagrange multipliers (Section 6.6.2). In our case, we maximize

$$u'Cov(X)u + \gamma(u'u - 1)$$

with respect to $u$ and $\gamma$.

Setting derivatives to 0, we have

$$0 = 2Cov(X)u + 2\gamma u$$

In other words,

$$Cov(X)\ u = -\gamma u \tag{13.1}$$

We see a situation in which a matrix ($Cov(X)$) times a vector ($u$) equals a constant ($-\gamma$) times *that same vector*. We say that $-\gamma$ is an *eigenvalue* of the matrix $Cov(X)$, and that $u$ is a corresponding *eigenvector*. Seems innocuous, but it opens a huge new world!

Note too that from Equation 4.7,

$$\text{maximal variance} = u'Cov(X)u = u'(-\gamma u) = -\gamma \tag{13.2}$$

We will return to the notion of principal components in the next chapter, after laying the groundwork in the current chapter.

## 13.3 Definition

The concept itself is simple:

> Consider an $n \times n$ matrix $M$. If there is a nonzero vector $x$ and a number $\lambda$ such that
>
> $$Mx = \lambda x \tag{13.3}$$
>
> we say that $\lambda$ is an *eigenvalue* of $M$, with *eigenvector* $x$.

## 13.4 A First Look

Here are a few properties to start with:

- The definition is equivalent to

  $$(M - \lambda I)x = 0$$

  which in turn implies that $M - \lambda I$ is noninvertible. That then implies that

  $$det(M - \lambda I) = 0 \tag{13.4}$$

- As mentioned before, determinants can be defined in the form of polynomials. Here the left-hand side of this equation is a polynomial in $\lambda$. So for an $n \times n$ matrix $M$, there are $n$ roots of the equation and thus $n$ eigenvalues. Note that some roots may be repeated; if $M$ is the zero matrix, say, it will have an eigenvalue 0 with multiplicity $n$.

- In principle, that means we can solve the above determinant equation to find the eigenvalues of the matrix. There are much better ways to calculate the eigenvalues than this, though.

## 13.5 The Special Case of Symmetric Matrices

Some eigenvalues may be complex numbers, i.e. of the form $a+bi$, but it can be shown that if $M$ is symmetric, its eigenvalues are guaranteed to be real. This is good news for Data Science, as many matrices in that field are symmetric, such as covariance matrices and $A'A$ in Equation 5.10.

**Theorem 13.1.** *The eigenvalues of a symmetric matrix are real, not complex numbers, i.e. not of the form $a+bi$.*

We omit the proof.

**Theorem 13.2.** *The eigenvectors corresponding to distinct eigenvalues of a symmetric matrix $M$ are orthogonal.*

*Proof.* Let $u$ and $v$ be such eigenvectors, corresponding to eigenvalues $\mu$ and $\nu$. Keeping in mind that $M' = M$, $(ab)' = b'a'$, and that the transpose of a number is just that number, we calculate $u'Mv$ in two different ways.

$$u'Mv = u'\nu v = \nu u'v$$

$$(u'Mv)' = v'(u'M)' = v'Mu = \mu v'u = \mu u'v$$

So, $\mu u'v = \nu u'v$, forcing $u'v = 0$ since $\mu$ and $\nu$ are distinct.

□ □

A square matrix $R$ is said to be *diagonalizable* if there exists an invertible matrix $P$ such that $P^{-1}RP$ is equal to some diagonal matrix $D$. We will see that symmetric matrices fall into this category.

One application of this is the computation of powers of a diagonalizable matrix:

$$R^k = (P^{-1}DP)\ (P^{-1}DP)\ ...\ (P^{-1}DP) = P^{-1}D^kP$$

since $PP^{-1} = I$.

$D^k$ is equal to the diagonal matrix with elements $d_i^k$, so the computation of $R^k$ is easy.

**Theorem 13.3.** *Any symmetric matrix $M$ has the following properties:*

- *$M$ is diagonalizable.*
- *In fact, the matrix $P$ is equal to the matrix whose columns are the eigenvectors $u_i$ of $M$, chosen to have norm 1. Thus the same holds for the rows of $P'$, so that*

$$P' u_i = \lambda_i u_i \tag{13.5}$$

- *The associated diagonal matrix has as its diagonal elements the eigenvalues of $M$.*
- *The matrix $P$ has the property that $P^{-1} = P'$.*
- *Moreover, the rank of $M$ is equal to the number of nonzero eigenvalues of $M$.*

*Proof.* Let $D = diag(d_1, ..., d_m)$, where the $d_i$ are the eigenvalues of $M$. Set these eigenvectors $u_i$ to have length 1, by dividing by their lengths.

Recall from Theorem 13.2 that the eigenvectors of $M$, i.e. the columns of $P$, are orthogonal, provided the eigenvalues are distinct, which we will take as an extra assumption here (the theorem is valid without it). Again using partitioning, write $P$ as

$$P = (u_1|...|u_m)$$

so that

$$P'P = \begin{pmatrix} u_1' \\ ... \\ u_m' \end{pmatrix} (u_1|...|u_m)$$

Then the row $i$, column $j$ element of $P'P$ is equal to $u_i'u_j$, which is either 1 or 0, according to whether $i = j$ or $i \neq j$. Thus $P'P = I$, as claimed.

By the way, any square matrix $Q$ such that $Q'Q = I$ is said to be *orthogonal*.

Now use partitioning again:

$$\begin{align}
MP &= M(u_1|...|u_m) \tag{13.6}\\
&= (Mu_1|...|Mu_m) \tag{13.7}\\
&= (d_1P_1|...|(d_mP_m) \tag{13.8}\\
&= PD \tag{13.9}
\end{align}$$

Multiply on the left by $P'$, and we are done:

$$P'MP = P'PD = D$$

Regarding rank, Theorem 7.1 tells us that pre- or postmultiplying by an invertible matrix does not change rank, and clearly the rank of a diagonal matrix is the number of nonzero elements.

□ □

**Theorem 13.4.** *Any square matrix $A$ and its transpose have the same eigenvalues.*

*Proof.* If $\lambda$ is an eigenvalue of $A$, then it satisfies Equation 13.4. But recall that the determinant of a matrix is equal to the determinant of its transpose. (For example, we can expand either across the top row or the leftmost row.) Then $\lambda$ also satisfies

$$0 = \det[(A - \lambda I)'] = \det[(A' - \lambda I)]$$

Thus $\lambda$ is also an eigenvalue of $A'$. This argument works in reverse, or better, note the above shows that $A'$ has least the eigenvalues of $A$ – and no more than that, since both have $n$ eigenvalues.

□ □

## 13.6 Example: Census Dataset

Let's illustrate all this with the census dataset, modified to include both male and female columns as in Section 12.3.3. We will form the matrix $A'A$ in Section 5.3.4, which as mentioned, is symmetric.

Recall that in that example, $A$ is not of full rank. Thus we should expect to see a 0 eigenvalue.

```
library(qeML)
data(svcensus)
svc <- svcensus[,c(1,4:6)]
svc$man <- as.numeric(svc$gender == 'male')
svc$woman <- as.numeric(svc$gender == 'female')
svc$gender <- NULL
a <- as.matrix(svc[,-2]) # look at "X" only, not "Y"
a <- cbind(1,a)  # add the column of 1s
m <- t(a) %*% a
eigs <- eigen(m)
eigs
```

```
eigen() decomposition
$values
[1] 7.594762e+07 3.292955e+06 7.466971e+03 1.293594e+03 2.801489e-12

$vectors
             [,1]         [,2]         [,3]        [,4]          [,5]
[1,] -0.015881850  0.004389585 -0.035995373  0.81553633  5.773503e-01
[2,] -0.649660696  0.760031134  0.004461047 -0.01654550  1.561251e-17
[3,] -0.759952974 -0.649864420  0.004991922 -0.01108122 -4.061024e-17
[4,] -0.012070494  0.002130704 -0.724401141  0.37650952 -5.773503e-01
[5,] -0.003811356  0.002258881  0.688405767  0.43902681 -5.773503e-01
```

```
m %*% eigs$vectors[,1]
```

```
               [,1]
         -1206188.6
age     -49340181.0
wkswrkd -57716616.5
man       -916725.2
woman     -289463.4
```

```
eigs$values[1] %*% eigs$vectors[,1]
```

```
        [,1]      [,2]      [,3]      [,4]      [,5]
[1,] -1206189 -49340181 -57716617 -916725.2 -289463.4
```

Yes, that first column is indeed an eigenvector, with the claimed eigenvalue.

Note that the expected 0 eigenvalue shows up as 2.801489e-12, quite small but nonzero, due to roundoff error.

## 13.7 Application: Detecting Multicollinearity

Consider the basic eigenanalysis equation,

$$Ax = \lambda x$$

for a square matrix $A$, a conformable vector $x$ and a scalar $\lambda$. Suppose that, roughly speaking, $\lambda x$ is small relative to $A$. Then

$$Ax \approx 0$$

and since $Ax$ is a linear combination of the columns of $A$, we thus we have found multicollinearity in $A$, flagged by the presence of a small eigenvalue..

One often sees use of the *condition number* of a matrix, which is the ratio of the largest eigenvalue to the smallest one. This too might be used as a suggestion of multicollinearity, though the main usage is as a signal that matrix operations such finding inverses may have significant problems with roundoff error.

## 13.8 Example: Currency Data

This dataset tracks five pre-euro European currencies.

```
library(qeML)
data(currency)
head(currency)
```

```
  Can..dollar Ger..mark Fr..franc UK.pound J..yen
1          19       580     4.763       29    602
2          18       609     4.818       44    609
3          20       618     4.806       66    613
4          46       635     4.825       79    607
5          42       631     4.796       77    611
6          45       635     4.818       74    610
```

```
dim(currency)
```

```
[1] 762   5
```

```
crc <- currency
crc <- as.matrix(crc)
crcapa <- t(crc) %*% crc
eigs <- eigen(crcapa)
eigs
```

```
eigen() decomposition
$values
[1] 4.180990e+08 5.356321e+07 8.199388e+06 4.686379e+06 4.764149e+02

$vectors
```

```
              [,1]         [,2]         [,3]          [,4]          [,5]
[1,] -0.464701846 -0.589664107  0.573442227  0.3277874495 -0.0082362154
[2,] -0.548123722  0.364936898 -0.424252867  0.6216029591 -0.0008432535
[3,] -0.008118519 -0.002208129  0.005818183 -0.0005349635  0.9999475368
[4,] -0.541352395 -0.364398075 -0.476833987 -0.5888747450 -0.0027404807
[5,] -0.436445017  0.621551662  0.513584476 -0.3992385223 -0.0053728096
```

```
# that last eigenvalue is much smaller than the others

# illustrate Ax = lambda x approx 0
crcapa %*% eigs$vectors[,5]
```

```
                    [,1]
Can..dollar  -3.9238561
Ger..mark    -0.4017386
Fr..franc   476.3899505
UK.pound     -1.3056059
J..yen       -2.5596868
```

```
eigs$values[5] *  eigs$vectors[,5]
```

```
[1]  -3.9238561  -0.4017386 476.3899505  -1.3056060  -2.5596868
```

```
# is Ax small?
head(crcapa)
```

```
            Can..dollar Ger..mark  Fr..franc  UK.pound    J..yen
Can..dollar   112111475  93929522 1673631.29 113542760  66967756
Ger..mark      93929522 136033582 1795560.29 116882053 109220119
Fr..franc       1673631   1795560   28573.49   1859363   1433430
UK.pound      113542760 116882053 1859363.20 133130962  85746625
J..yen         66967756 109220119 1433430.43  85746625 103243878
```

The value for the franc is much smaller than the others. So yes, this dataset has some multicollinearity.

## 13.9 Computation: the Power Method

One way to compute eigenvalues and eigenvectors is the *power method*, a simple iteration. We begin with an initial guess, $x_0$ for an eigenvector. Substituting in Equation 13.3, we have the next guess:

$$x_1 = Mx_0$$

We keep iterating, generating $x_2$ from $x_1$,, generating $x_3$ from $x_2$ and so on. until convergence, meaning that $x_{i+1}$ and $x_i$ do not differ much from each other.

However, the $x_i$ may grow, so we normalize to length 1:

$$x_i \leftarrow \frac{x_i}{||x_i||}$$

Now, after obtaining an eigenvector in this manner, how do we get the associated eigenvalue? We can do this by calculating the *Rayleigh Quotient*: Denote the eigenvalue associated with $x$ by $\lambda$. Then

$$\lambda = \frac{(Ax)'x}{x'x}$$

To see this, simply substitute $Ax$ in the numerator by $\lambda x$.

It can be shown that the above iterative process yields the maximal eigenvalue of $A$. But what about the rest of the eigenvalues? This is achieved via *deflation* techniques, which we will not present here.

## 13.10 Application: Computation of Long-Run Distribution in Markov Chains

We showed in Section 3.3 how matrix inverse can be used to compute the long-run distribution $\nu$ in a Markov chain. However, this is inefficient for very large transition matrices. For instance, in Google PageRank, there is a Markov state for every page on the Web!

Instead, we exploit the fact that Equation 3.3 says that the transition matrix has an eigenvector $\nu$ with eigenvalue 1. Due to the typical huge size of the matrix, the power method or a variant is often used.

## 13.11 Your Turn

**Your Turn:** Take a symmetric matrix $A$ of your choice, use R to find its eigenvalues and eigenvectors. Verify that the latter are orthogonal, and that the matrix $P$ formed as in Theorem 13.3 does indeed have its transpose as its inverse. (If you choose a noninvertible $A$, try another.)

**Your Turn:** Show that any Markov transition matrix has an eigenvalue 1, with eigenvector consisting of all 1s.

**Your Turn:** Show that the diagonalizing matrix $P$ for a symmetric matrix must have determinant $\pm 1$.

**Your Turn:** Show that if $x$ is an eigenvector of $M$ with eigenvalue $\lambda \neq 0$, then for any nonzero number $c$, $cx$ will also be an eigenvector with eigenvalue $\lambda$.

**Your Turn:** Show that if a matrix $M$ has a 0 eigenvalue, $M$ must be singular. Also prove the converse. (Hint: Consider the column rank.)

**Your Turn:** Consider a projection matrix $P_W$. Show that the only possible eigenvalues are 0 and 1. Hint: Recall that projection matrices are idempotent.

**Your Turn:** Say $A$ is an invertible matrix with eigenvalue $\lambda$ and eigenvector $v$. Show that $v$ is also an eigenvector of $A^{-1}$, with eigenvalue $1/\lambda$.

**Your Turn:** Show that if $A$ is nonnegative-definite, its eigenvalues must be nonnegative.

**Your Turn:** Theorem 13.3 says that for any symmetric matrix $M$ is diagonalizable: There is an orthogonal matrix $P$ such that $P'MP$ is equal to a diagonal matrix $D$, the latter consisting of the eigenvalues of $M$. Use this to show that

$$x'Mx \leq \lambda_{max}||x||^2$$

where $\lambda_{max}$ is the largest eigenvalue. Hint: First show that

$$x'Mx = (Px)'D(Px)$$

# 14 Principal Components

Goals of this chapter:

It turns out that eigenanalysis can be quite useful for dimension reduction, by determining the *principal components* of our data. This is the focus of the current chapter.

So in the last chapter we had a very brief introduction to Principal Components Analysis (PCA), finding the first component entails finding an eigenvector and eigenvalue. It turns out that this is true for the second, third and all the components. In other words, PCA is basically an eigenvectors and eigenvalues application.

## 14.1 The Second, Third Etc. PCs

Say we have a dataset $Q$ in matrix form, again using the **mtcars** data for concreteness.

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

There are as many principal components as there are columns in $Q$. We derived the first PC in Section 13.2.1. So, how are the second, third and so on components defined and computed?

The key issue is that we want our PCs to be orthogonal vectors, because as we saw in Section 10.6, orthogonal random vectors are statistically uncorrelated, and independent in the multivariate normal case. So if we summarize our data with say, the first few PCs, the fact that they are uncorrelated makes for a neater summary, in a sense having no duplication.

In the sense of random-X regression (Section 5.3.7), what we write as "Cov(X)" below should actually be $\widehat{Cov}(X)$, a sample estimate of a population quantity. However, we omit the "hat" in the interests of simplicity.

So with $u$ and $v$ denoting the first and second PCs, we want $v$ to maximize $v'Cov(Q)v$, so we set the derivative of

$$v'Cov(Q)v + \omega(v'v - 1) + \tau(u'v - 0)$$

with respect to $v$ to 0. (Note the two Lagrange variables, $\omega$ and $\tau$.)

$$0 = 2Cov(Q)v + 2\omega v + \tau u \tag{14.1}$$

Pre-multiply by $u'$:

$$\begin{aligned} 0 &= 2u'Cov(Q)v + 2\omega u'v + \tau u'u \\ &= 2u'Cov(Q)v + \tau \end{aligned} \tag{14.2}$$

Also, since $Cov(Q)$ is symmetric and $u$ is an eigenvector of $Cov(Q)$, say with eigenvalue $\gamma$,

$$u'Cov(Q)v = [Cov(Q)u]'v = \gamma u'v = 0 \tag{14.3}$$

So Equation 14.2 and Equation 14.4 now tell us that $\tau = 0$, and thus Equation 14.1 reduces to

$$Cov(Q)v = -\omega v \tag{14.4}$$

showing that the second PC is again an eigenvector of $Cov(Q)$. Equation 13.2 then says that $-\omega$ is the corresponding variance.

The story is the same for the remaining PCs.

**Theorem 14.1.** *The principal components have the following properties:*

- *They are orthogonal to each other.*
- *They are eigenvectors of $Cov(Q)$.*
- *The eigenvalue corresponding to a PC $z$ is $Var(z'Q)$.*
- *The eigenvalues form a nondecreasing sequence.*

- *Say $Q$ has $m$ columns. Normalize the PCs $u_i$ (i.e. divide a PC by its length, so that we have a vector of length 1) and form the $m \times m$ (partitioned) matrix*

$$P = (u_1|u_2|...|u_m)$$

  *Then*

$$P'Cov(Q)P = D \tag{14.5}$$

  *where $D$ is a diagonal matrix whose entries are $Var(u_i'Q)$, and $P'P = I$.*

- *PC1, i.e. $u_1$, is a linear combination of the original variables, PC2, i.e. $u_2$, is a linear combination of the original variables, and so on. The PCi are our new predictor variables, though we will probably use only the first few.*

- *Let $G$ denote a matrix in the same form at the matrix $A$ in the linear regression model (without a 1s column). In other words, each row contains the data on one sampling unit, e.g. one person. To convert from the original predictor variables to PCs, compute $GP$. If we wish to use just the first few PCs, then compute*

$$G(P_1|...|P_k)$$

  *where $k$ denotes our desired number of PCs. (A topic to be explored in Section 14.5.)*

- *By construction, the covariance matrix of $GP$ is $D$, so the PCs are uncorrelated.*

## 14.2 Review of Goals Achieved

- We hope to achieve *dimension reduction*, meaning a reduction in our number of predictor variables, for two reasons:

  - Parsimony: All else being equal, smaller models are easier to deal with and interpret.
  - Avoidance of overfitting: As the model size $p$ increases relative to fixed data size $n$, there is a point past which the predictive power of a model actually begins to decline. (But see Chapter 16.)

- PCA achieves those goals:
  - We produce new predictors, the PCs, from our original ones, ranked according to variance. (Recall: A variable with low variance is nearly constant, thus rather useless.)
  - We can reduce our number of predictors by using only some of the PCs.
  - The new predictors don't duplicate each other, in that they are uncorrelated.

## 14.3 The PCs as Our New Predictor Variables

So, let's see how all this works. We will start with a simple dataset, our census data.

```
data(svcensus)
head(svcensus)
```

```
       age     educ occ wageinc wkswrkd gender
1 50.30082 zzzOther 102   75000      52 female
2 41.10139 zzzOther 101   12300      20   male
3 24.67374 zzzOther 102   15400      52 female
4 50.19951 zzzOther 100       0      52   male
5 51.18112 zzzOther 100     160       1 female
6 57.70413 zzzOther 100       0       0   male
```

```
svc <- factorsToDummies(svcensus)
# remove our "Y" variable
svc <- svc[,-11]
z <- prcomp(svc)
names(z)
```

```
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
z$sdev
```

```
 [1] 1.460125e+01 1.122240e+01 6.221131e-01 5.967231e-01 5.165748e-01
 [6] 4.730515e-01 3.879820e-01 2.264263e-01 2.211419e-01 1.668888e-01
[11] 2.255119e-15 1.424588e-15 1.270416e-15
```

The entity **z$rotation** is our matrix $P$. Here **sdev** is the square roots of the variances, i.e. of the square roots of the eigenvalues. It appears that there are two main PCs, the rest minor. Here we can see which of the original variables each of these two PCs focuses on.

It's called a rotation because geometrically the change of variables really does cause a rotation of the coordinate system.

```
> z$rotation
                        PC1           PC2          PC3           PC4
age            0.0071442921  0.9999532343  0.000726208 -2.397875e-05
educ.14       -0.0010745227  0.0010525943 -0.321503608  4.907644e-01
educ.16       -0.0004109197  0.0010580494 -0.036350998  2.631539e-02
educ.zzzOther  0.0014854423 -0.0021106436  0.357854606 -5.170798e-01
occ.100        0.0009216241  0.0009575240  0.242783424 -8.554274e-02
occ.101        0.0016559718 -0.0011507204  0.101640265 -1.731946e-01
occ.102       -0.0016720082 -0.0037463757 -0.245946083  3.378269e-01
occ.106       -0.0001386860 -0.0001002262  0.017018077  3.846487e-03
occ.140       -0.0001607628 -0.0003608480 -0.010823679 -1.106524e-02
occ.141       -0.0006061389  0.0044006464 -0.104672004 -7.187076e-02
wkswrkd       -0.9999670218  0.0071416042  0.003457670 -9.540378e-04
gender.female  0.0015140583 -0.0001762090  0.559515249  4.088654e-01
gender.male   -0.0015140583  0.0001762090 -0.559515249 -4.088654e-01
...
```

In that first PC, the largest component by far is **wkswrkd**, trailed substantially by the next-largest, **age**. Those two variables more or less trade places in the second PC.

In other words, if our project is to predict wage income, we might use PCs 1 and 2 as our predictors, as opposed to using all 13 of the original variables.

The numbers suggest that none of the other predictors is very powerful, though this should not be viewed as meaning that the others are collectively useless.

Well, how do we actually use PC1 and PC2? Say we have several new people for which we need to predict wage income. Place their predictor values in a matrix, say $G$, one person per row (i.e. the same format at the matrix $A$ in Equation 5.10, though without a 1s column). Then as stated in the theorem, our matrix of new predictor data is simply the matrix product shown above

```
G %*% P[,1:nPC]
```

So to make the full switch, convert the entire dataset:

```
P <- z$rotation
svcNew <- svc %*% P[,1:2]
head(svcNew)  # our new data!
```

```
             PC1      PC2
[1,] -51.6375939 50.66379
[2,] -19.7040727 41.23922
[3,] -51.8206812 25.03792
[4,] -51.6387521 50.56755
[5,]  -0.6303930 51.18454
[6,]   0.4131482 57.70046
```

```
lmOut <- lm(svcensus$wageinc ~ svcNew)
lmOut
```

```

Call:
lm(formula = svcensus$wageinc ~ svcNew)

Coefficients:
(Intercept)    svcNewPC1    svcNewPC2
   -22101.3      -1385.4        508.4
```

## 14.4 Back to the African Soils Example

So, we remove the "Y'' variable, **pH**, number 3598 as in Section 13.1, and proceed. We will also remove the nonnumeric columns, **PIDN** and **Depth**.

```
library(WackyData)
data(AfricanSoil)
x <- AfricanSoil[,-c(1,3595,3598)]
z <- prcomp(x)
head(z$sdev,25)
```

```
 [1] 8.7178222 3.0908233 2.6418988 2.0517201 1.6983891 1.4748272 1.3980475
 [8] 1.1497188 0.9682689 0.8033391 0.7974972 0.7405673 0.6734597 0.6310006
[15] 0.6054025 0.5912755 0.5096743 0.4511786 0.3871415 0.3533344 0.3345691
[22] 0.3089582 0.2843214 0.2532976 0.2465674
```

Ah, the eigenvalues fall off rapidly after the first few. So we might use, say, the first dozen PCs.

That's quite a feat! We started with over 3,000 predictors, and cut it down to 12.

```
newx <- as.matrix(x) %*% z$rotation[,1:12]
head(newx)
```

```
           PC1         PC2       PC3        PC4       PC5       PC6       PC7
[1,] -46.55433  1.75412328 -27.61468 -0.7715529 -14.72332 -1.181072 -8.763462
[2,] -48.97753 -0.03899486 -28.60920 -0.3140904 -14.19085 -1.831532 -9.773733
[3,] -41.34060  2.79805480 -25.44885 -0.3670190 -15.62468 -1.645295 -8.645823
[4,] -39.34208  1.92317324 -25.78511 -0.2379570 -15.08456 -1.647429 -9.150608
[5,] -41.88505  2.64120041 -27.53284 -1.5812961 -15.23804 -1.191244 -8.990876
[6,] -37.09569  1.24453500 -27.13932 -1.1065947 -14.96667 -1.696422 -9.365870
          PC8           PC9     PC10      PC11       PC12
[1,] 2.860247 -0.2246997473 1.816366 -2.303456 -0.5364818
[2,] 3.001019 -0.0038125862 1.305971 -2.530724 -0.6107867
[3,] 3.037891  0.0686585033 2.673836 -2.017832 -0.6884367
[4,] 2.839273  0.0861274613 2.513475 -2.241782 -0.7523705
[5,] 3.492016 -0.0008967448 1.123003 -2.025351 -0.4733205
[6,] 3.027975 -0.3744700168 1.398285 -2.234916 -0.3414471
```

```r
lmOut <- lm(AfricanSoil[,3598] ~ newx)
lmOut
```

```
Call:
lm(formula = AfricanSoil[, 3598] ~ newx)

Coefficients:
(Intercept)      newxPC1      newxPC2      newxPC3      newxPC4      newxPC5
   1.322005    -0.007453    -0.037351     0.090569    -0.277864    -0.044382
    newxPC6      newxPC7      newxPC8      newxPC9     newxPC10     newxPC11
  -0.255148     0.076345    -0.060700    -0.062136    -0.001227    -0.086023
   newxPC12
  -0.130541
```

## 14.5 How Many Principal Components Should We Use?

There are no formal rules for how many PCs to use. Various "rules of thumb" do exist, but of uncertain real value.

If we are doing prediction, there is a very natural way to choose our number of PCs – do cross-validation (Section 12.6), and use whichever number gives the most accurate prediction.

### 14.5.1 Example: New York City taxi trips

This dataset was introduced in Section 12.12, a version of which is included in the **qeML** package. The object is to predict trip time, given pickup and dropoff locations, trip distance and day of the week.

In the raw form of the data, there are just 5 columns, thus 4 predictors. But the pickup and dropoff locations are R factors, coding numerous locations. As before, these must be decoded to dummy variables (values 1 or 0, coding whether or not, say, a given trip began at pickup location 121). If for instance one calls the R linear regression function **lm**, that function will do the conversion internally, but in our case we will perform the conversion ourselves, using **regtools::factorsToDummies**. (The **regtools** package is included by **qeML**.)

```r
library(qeML)
data(nyctaxi)
dim(nyctaxi)
```

```
[1] 10000     5
```

```r
nyc <- factorsToDummies(nyctaxi,dfOut=T)
dim(nyc)
```

```
[1] 10000   357
```

Wow! That's quite a lot of predictors, and well in excess of the common rule of thumb that one should have no more than $\sqrt{n}$ predictors, 100 in this case.

So, we might try dimension reduction via PCA, then use the PCs as predictors. The function **qeML::qePCA** combines these two operations. Let's see how it works.

```r
args(qePCA)
```

```
function (data, yName, qeName, opts = NULL, pcaProp, holdout = floor(min(1000,
    0.1 * nrow(data))))
NULL
```

Here **qeName** indicates which function is desired for prediction, e.g. **qeLin** for a linear model. (This function wraps **lm**.) But a key argument here is **pcaProp**. Recall that:

- The PCs come in order of decreasing variance.
- We are mainly interested in the first few PCs. The later ones have small variance, which makes them approximately constant and thus of no use to us.

The name ‘pcaProp’ stands for “proportion of total variance.” If we set this to, say, 0.25, we are saying “Give us whatever number of the first few PCs that have a total variance of at least 25% of the total.” Let’s give that a try:

```
qePCA(nyc,'tripTime','qeLin',pcaProp=0.25)$testAcc
[1] 311.9159
```

We asked to predict the column ‘tripTime’ in the dataset **nyc** using the **qeLin** function, based on as many of the PCs that will give us 25% of the total variance. Most **qeML** prediction functions split the data into a training set and a holdout set. The model is fit to the training set, and then applied to prediction of the holdout set. The output value is the mean absolute prediction error (MAPE).

However, since the holdout set is randomly generated, the MAPE value is random, so we should do multiple runs. Some experimentation showed that MAPE here is highly variable, so we decided to perform 500 runs, e.g.

```
mean(sapply(1:500,function(i) qePCA(nyc,'tripTime','qeLin',pcaProp=0.1)$testAcc))
[1] 359.663
```

Table 14.1: Mean Absolute Predictive Error

| pcaProb | MAPE |
|---|---|
| 0.1 | 359.6630 |
| 0.2 | 359.3068 |
| 0.3 | 403.9878 |
| 0.4 | 397.3783 |
| 0.5 | 430.8958 |
| 0.6 | 423.7299 |
| 0.7 | 517.7917 |
| 0.8 | 969.1304 |
| 0.9 | 2275.091 |

Among other things, this shows the dangers of overfitting, in this case using too many PCs in our linear regression model. It seems best here to use only the first 10 or 20% of the PCs.

## 14.6 A "Square Root" Matrix, and MV Normal Simulation

**Theorem 14.2.** *Any covariance matrix $A$ has a "square root" matrix $Q$, i.e.*

$$Q^2 = A$$

*Proof.* From Theorem 14.1, we have $P'AP = D$ for some matrix $P$ and diagonal matrix $D$, with the entries of the latter being the variances of the PCs $\sigma_i^2$. That latter point implies that

$$D_1^2 = D$$

where $D_1 = diag(\sigma_1, \sigma_2, ..., \sigma_n)$. Then setting $Q = PD_1P'$, we have

$$Q^2 = (PD_1P')(PD_1P') = PD_1^2P' = PDP' = A$$

### 14.6.1 Implications for simulation of multivariate normal X

Say we wish to write code to simulate a random vector $Q$ having an m-variate normal distribution with mean vector $\mu$ and covariance matrix $\Sigma$. Here is how it works:

- We start with generating $Z$, a vector of $m$ independent N(0,1) variables. That means $Z$ is m-variate normally distributed, with mean vector consisting of $m$ 0s, and covariance matrix $I$, the $m \times m$ identity matrix.
- We compute $W = \Sigma^{0.5}Z$. By Equation 4.10, $W$ will again be multivariate normally distributed, with mean vector consisting of $m$ 0s, and covariance matrix equal to

$$\Sigma^{0.5} I \Sigma^{0.5} = \Sigma$$

Our solution is then

$$Q = \mu + \Sigma^{0.5} Z$$

□ □

## 14.7 Your Turn

**Your Turn:** Use PCA to do dimension reduction on the **s50** dataset.

**Your Turn:** The reader has likely seen the concept of a *cumulative distribution function* (CDF). For a scalar random variable $X$, this is $F_x(t) = P(X \leq t)$. If $X$ is an $m$-dimensional random vector, the definition is

$$F_X(t_1, ..., t_m) = P(X_1 \leq t_1, ..., X_m \leq t_m)$$

Write an R function with call form

```
multiCDF(mu,Sigma,t,n)
```

that uses simulation to evaluate the multivariate CDF, where $t = (t_1, ..., t_m)$ and $n$ is the number of replications to simulate.

**Your Turn:** Say $X$ has mean vector $\mu$ and covariance matrix $\Sigma$. Show how we can use Theorem 14.2 to find a square, constant matrix $A$ such that $Cov(AX) = I$. If in addition $X$ has a multivariate normal distribution, then $AX$ will then consist of independent random variables with variance 1. Explain why.

**Your Turn:** Modify the code for **qePCA** for the case of linear regression by adding a component **xCoeffs** to its return value. This will give the regression coefficients in terms of the original X predictors.

**Your Turn:** Say the symmetric matrix $A$ has *block diagonal* form

$$A = \begin{pmatrix} A_1 & 0 & 0... \\ 0 & A_2 & 0... \\ ... & & \end{pmatrix}$$

where $A_i$ $(i = 1, ..., r)$

- is symmetric
- is of size $k_i \times k_i$
- has eigenvalues $\gamma_1, ..., \gamma_{k_i}$
- has eigenvectors $u_{1,j}, ..., u_{k,j}$, where $j = 1, ..., k_i$

State the form of the eigenvalues and eigenvectors of $A$.

**Your Turn:** In Section 13.7, there was a statement "$c$ is nonzero with high probability." Why was that important?

**Your Turn:** Write an R function with call form

```
bestPCAPred(data,yName)
```

It will apply PCA to the X portion of **data**, then apply **lm** successivly to the first PC, then the first two, then the first three and so on assessing with cross-validation in each case. It will then return the number of PCs (and the PCs themselves) that predicts best. Try your function on various datasets.

**Your Turn:** Say the symmetric $n \times n$ matrix $A$ has rank $r$. Show that $n - r$ of its eigenvalues are 0s.

# 15 Singular Value Decomposition

Goals of this chapter:

We've seen the notion of eigenanalysis for square matrices. Well, it turns out this can be extended usefully for non-square matrices, via *Singular Value Decomposition* (SVD), the topic of this chapter, with important applications.

## 15.1 Basic Idea

Here is what we are aiming for.

Our target relation:

Given an $m \times n$ matrix $A$, we wish to find orthogonal matrices $U$ and $V$, and a matrix $\Sigma = diag(\sigma_1, ..., \sigma_n)$ whose nonzero elements are positive, such that

$$A = U \Sigma V' \tag{15.1}$$

By convention the ordering of columns is set so that the $\sigma_i$ occur in nonincreasing order.
Note that, by matrix partitioning, Equation 15.1 also says that

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i' \tag{15.2}$$

where $u_i$ and $v_i$ are the $i^{th}$ rows of $U$ and $V$.

Note the dimensions:

- $U$ is $m \times n$
- $V$ is $n \times n$
- $\Sigma$ is $n \times n$

Let $r$ denote the rank of $A$. It will turn out that $\sigma_i = 0$ for $i = r + 1, ..., n$.

## 15.2 Example Applications

Here are some brief overviews of applications.

### 15.2.1 Recommender Systems

Recall Section 2.10, which began with movie ratings data. We would like to predict the rating that some particular user would give to some particular movie. It will turn out that we can neatly set up SVD in such a way that $U$ contains movie data and $V$ contains user data.

### 15.2.2 Text Classification

An oft-used example is that in which we have a collection of newspaper articles that we wish to categorize, say politics, sports, health, finance and so on. Say one of the articles includes the word *bonds*; is it referring to financial instruments, family relations, former baseball star Barry Bonds etc.?

In spite of today's dazzling array of Large Language Models, a simple problem like this may be better solved using straightforward methods, such as SVD. We set up a *document-term matrix*, with element $(i, j)$ being 1 or 0, according to whether document $i$ contains word $j$. Applying SVD to this matrix, we have a setting with similar appeal to the recommender systems example above, in which $U$ contains our data on documents and $V$ does the same for words.

### 15.2.3 Dimension Reduction

Recall that in the matrix $\Sigma$, the eigenvalues are arranged in decreasing order, possibly followed by 0s. This suggests that we can achieve dimension reduction by replacing even some of the smaller eigenvalues by 0s as well, with corresponding adjustments to the columns of $U$ and $V$. We will discuss this below in Section 15.8.

## 15.3 Solution to Our SVD Goal

There are various derivations, e.g. along the lines of Section 13.2.1, but let's go directly to the answer:

1. Say we have a $p \times n$ matrix $A$ (not necessarily from the regression context), so that $A'A$ will be of size $n \times n$. Let $r$ denote the rank of $A'A$, which will be the same as the rank of $A$ from Theorem 7.5.

2. Compute the eigenvalues $\sigma_1, ..., \sigma_n$ and eigenvectors $v_1, ..., v_n$ of $A'A$. Normalize the $v_i$ by dividing by their lengths. (The same eigenvalues will still hold.) Order the $\sigma_i$ from largest to smallest, and use the same ordering for the $v_i$.

3. Since $A'A$ is symmetric and nonnegativre-definite, its eigenvalues will be nonnegative and its eigenvectors $v_1, ..., v_r$ will be orthogonal as long as $\sigma_1, ..., \sigma_r$ are distinct, as is typically the case for numeric data. Since these eigenvectors diagonalize $A'A$ with the $\sigma_i$ on the diagonal (Chapter 14), we will have $\sigma_{r+1} = ... = \sigma_n = 0$.

4. Set $\Sigma$ to $diag(\sigma_1, ..., \sigma_n)$, the nonincreasing list of those eigenvalues. Set the first $r$ columns of $V$ to the corresponding eigenvectors. Use the Gram-Schmidt Method (see Section 10.5) to add $n - r$ more vectors. $V$ will then have orthonormal columns, and thus be an orthogonal matrix.

5. Set

$$u_i = \frac{1}{\sigma_i} A v_i, \ \ i = 1, ..., r \tag{15.3}$$

   The $u_i$ will be orthogonal: For $i \neq j$,

$$u_i' u_j = \frac{1}{\sigma_i \sigma_j} v_i' A' A v_j = v_i' v_j = 0$$

   where we have used the facts that $v_j$ is an eigenvector of $A'A$ and the $v_k$ are orthogonal.

Using Gram-Schmidt, we can compute (if $r < n$ necessitates it) vectors $u_{r+1}, ...u_n$ so that $u_1, ..., u_n$ is an orthonormal basis for $\mathcal{R}^m$. Set $U$, described in partitioning terms: to

$$U = (u_1|...|u_n)$$

## 15.4 Interpretation of $U$ and $V$

Again, say $A$ is of size $m \times n$. Recall that $V$ consists of the eigenvectors of the $n \times n$ matrix $A'A$. $U$ is constructed from vectors as in Equation 15.3, but actually those are eigenvectors of $AA'$:

$$\begin{aligned}
(AA')u_i &= (AA')(\frac{1}{\sigma_i}Av_i) && (15.4)\\
&= \frac{1}{\sigma_i}A[(A'A)\ v_i] && (15.5)\\
&= \frac{1}{\sigma_i}A\sigma_i v_i && (15.6)\\
&= \sigma_i u_i && (15.7)
\end{aligned}$$

So, $u_i$ is an eigenvector of $AA'$, with eigenvalue $\sigma_i$.

## 15.5 SVD as a basis for matrix generalized inverse

Recall the example in Section 7.1. We could not have dummy-variable columns for both male and female, as their sum would be a column of all 1s, in addition to a column of 1s the X data matrix already had. The three columns would then have a nonzero linear combination that evaluates to the 0 vector. Then in Equation 5.10, $A$ (i.e. X) would not be of full rank, nor would $A'A$ (Theorem 7.5), so that $(A'A)^{-1}$ would not exist.

And yet the equation from which that comes,

$$A'Ab = A'S \tag{15.8}$$

is still valid. We could, as in that example, remove one of the gender columns, thus solving the problem of less than full rank, but the use of *generalized inverses* solves the problem directly. If $A$ has many binary variables, the use of generalized inverses may be more convenient.

One of the most famous forms of generalized inverse, is the Moore-Penrose *pseudoinverse*, based on SVD. Given the SVD of a matrix $M$,

$$M = U_M \Sigma_M V_M'$$

its Moore-Penrose inverse, denoted by $M^+$, is

$$M^+ = V_M \Sigma_M^+ U_M'$$

where $\Sigma_M^+$ is the diagonal matrix obtained from $\Sigma_M$ by replacing each nonzero element by its reciprocal.

The Moore-Penrose solution of $Mz = w$ for vectors $z$ and $w$, is

$$z = M^+ w \tag{15.9}$$

The R function **MASS::ginv** performs the necessary computation for us; we need not call **svd()**.

## 15.6 SVD in linear models

Now apply this to our linear model problem (Equation 15.8). We claim that

$$b = A^+ S = V \Sigma^+ U' S \tag{15.10}$$

solves the equation. (Actually if $A$ is of less than full rank, there are infinitely many solutions, a point discussed in detail in Chapter 16.)

Let's check. Before beginning, recall that the orthogonal nature of $U$ and $V$ implies that $U'U = I$ and $V'V = I$. Now substitute Equation 15.10 in the left side of Equation 15.8:

$$
\begin{aligned}
A'Ab &= (U\Sigma V')'(U\Sigma V')(U\Sigma V')^+S && (15.11)\\
&= (V\Sigma U')(U\Sigma V')(V\Sigma^+U'S) && (15.12)\\
&= V\Sigma^2V'(V\Sigma^+U'S) && (15.13)\\
&= V\Sigma U'S && (15.14)\\
&= A'S && (15.15)
\end{aligned}
$$

□

## 15.7 Example: Census Data

```
library(qeML)
data(svcensus)
head(svcensus)
```

```
       age     educ occ wageinc wkswrkd gender
1 50.30082 zzzOther 102   75000      52 female
2 41.10139 zzzOther 101   12300      20   male
3 24.67374 zzzOther 102   15400      52 female
4 50.19951 zzzOther 100       0      52   male
5 51.18112 zzzOther 100     160       1 female
6 57.70413 zzzOther 100       0       0   male
```

```
svc <- svcensus[,-c(2,3)]
# have only 1 categorical/dichotomous variable, for simple example
svc <- factorsToDummies(svc)
head(svc)
```

```
          age wageinc wkswrkd gender.female gender.male
[1,] 50.30082   75000      52             1           0
[2,] 41.10139   12300      20             0           1
[3,] 24.67374   15400      52             1           0
[4,] 50.19951       0      52             0           1
[5,] 51.18112     160       1             1           0
[6,] 57.70413       0       0             0           1
```

```r
x <- cbind(1,svc[,-2])
head(x)
```

```
            age wkswrkd gender.female gender.male
[1,] 1 50.30082      52             1           0
[2,] 1 41.10139      20             0           1
[3,] 1 24.67374      52             1           0
[4,] 1 50.19951      52             0           1
[5,] 1 51.18112       1             1           0
[6,] 1 57.70413       0             0           1
```

```r
xplus <- MASS::ginv(x)
bhat <- xplus %*% svc[,2]
bhat
```

```
           [,1]
[1,] -16022.454
[2,]    496.747
[3,]   1372.756
[4,] -13361.634
[5,]  -2660.821
```

```r
lm(wageinc ~ .,svcensus[,-c(2,3)])$coef
```

```
(Intercept)         age     wkswrkd  gendermale
 -29384.088     496.747    1372.756   10700.813
```

Since the two analyses use different sets of predictors, it is not surprising that the intercept terms differ, but otherwise the two approaches are consistent with each other. This includes the gender variables, since

$$-13361.634 - (-2660.821) = -10700.81$$

## 15.8 Dimension Reduction: SVD as the Best Low-Rank Approximation

This is a very common application of SVD.

### 15.8.1 "Thin" SVD

The SVD of a rank $r$, $q \times n$ matrix $A$ can be partitioned as

$$A = (U_1|U_2) \left( \begin{array}{cc} D_1 & 0 \\ 0 & 0 \end{array} \right) (V_1|V2)'$$

where $U_1$ is of size $q \times r$, $V_1$ is of size $n \times r$, and $D_1$ is $r \times r$. Simplifying, we have

$$A = (U_1 D_1|0) \left( \begin{array}{c} V_1' \\ V_2' \end{array} \right) = U_1 D_1 V_1' \qquad (15.16)$$

So, we've reduced the size of memory needed to store the SVD, by using $U_1$, $D_1$ and $V_1$ instead of the larger $U$, $D$ and $V$. This is called the *thin* SVD.

But there's more:

### 15.8.2 Low-rank approximation

The eigenvalues of a matrix, arranged from largest to smallest, tend to degrade in a gradual manner, as seen for example in Section 14.4. So, in the context of SVD, with the last $n - r$ eigenvalues being 0s, it will typically be the case that the last few eigenvalues among $\sigma_1, ..., \sigma_r$ are *near* 0.

In other words, we probably can get a good approximation to the SVD by treating those near-0 eigenvalues as 0s, and removing the corresponding columns of $U_1$ and $V_1$. Why do this?

- Achieve a further reduction in storage requirements. for instance in settings in which we have many, many images or even better, many, many videos.
- Reduce noise, again for example with images. Small blotches are smoothed out.
- In the spirit of Chapter 12, the low-rank approximation may be viewed as a "shrunken" SVD, a remedy to possible overfitting.

It can be shown that the above recipe produces the "best" low-rank approximation, in terms of *Frobenius* norm. The latter treats an $m \times n$ matrix as a vector of length $mn$ and applies the ordinary $l_2$ norm.

### 15.8.3 Example: Image Compression

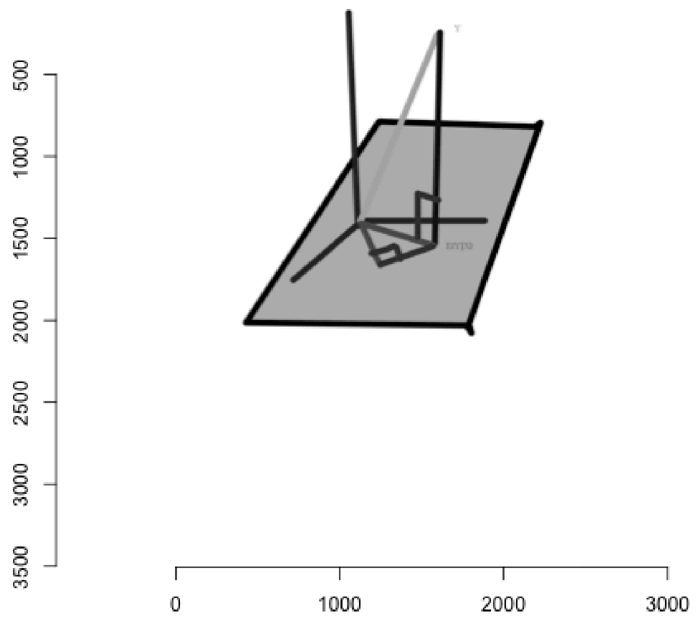
Let's try that idea with the picture on the cover of this book. We will

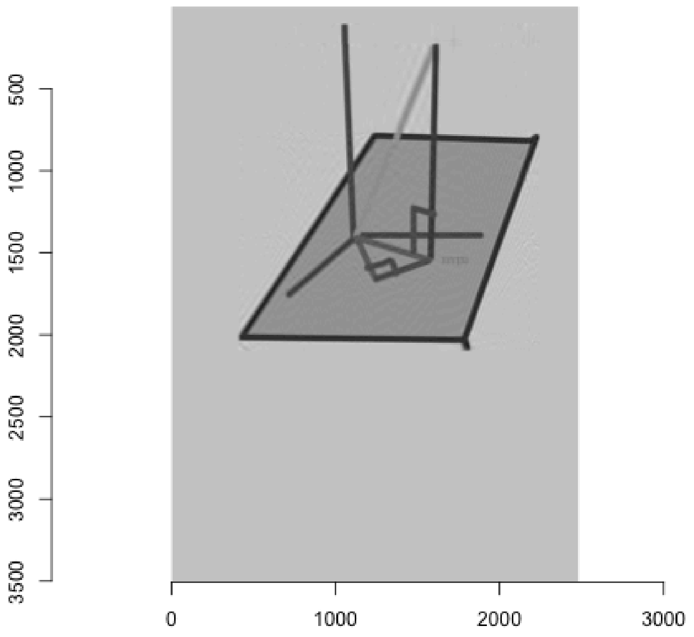
- convert from color to grayscale for simplicity (a color image consists of three matrices, one for each primary color)
- calculate the SVD of the resulting matrix
- retain only the first $k$ eigenvectors and eigenvalues; the smaller $k$ is, the greater the storage savings but the poorer the approximation

Here is the code:

```r
library(imager)
img <- load.image('prj.png')
# grayscale() doesn't work directly on this
# image, due to alpha (transparency) channel
imgNoAlpha <- rm.alpha(img)
imgGrayNoAlpha <- grayscale(imgNoAlpha)
dim(imgGrayNoAlpha)
imgSVD <- svd(imgGrayNoAlpha)
u <- imgSVD$u
v <- imgSVD$v
origImgMat <- u %*% diag(imgSVD$d) %*% t(v)
```

```
plot(as.cimg(origImgMat))
retainedRank <- 50  # keep only the first 50 columns in U and V
newImgMat <- u[,1:retainedRank] %*% diag((imgSVD$d)[1:retainedRank]) %*%
   t(v[,1:retainedRank])
plot(as.cimg(newImgMat))
```





The rank-50 version is almost as sharp as the full image – and achieves a huge reduction in storage space.

### 15.8.4 Example: MNIST optical character recognition

The famous MNIST image dataset is the "Hello World" of image recognition, everyone's introductory example. It consists of 70,000 images, each of size $28 \times 28 = 784$ pixels. images of handwritten digits 0 through 9. The goal of course is to correctly guess the digit ("Y"), based on the 784 pixels ("X").

The situation here is a bit different from other classification applications we've covered so far, in the here "Y" is a vector rather than a scalar. Element $i$ of the vector is either 1 or 0, depending on whether the handwritten digit is $i$ or not.

There are various ways of handling this, but the one we will discuss here is to perform 10 separate classification fits, one for each digit, using the logistic model. In other words, we will call **glm** 10 times, yielding 10 fits. Since fit $i$ gives the probability of digit $i$, given the values of the 784 pixels, we then take our guess for the digit to be the one with highest probability.

In our usual notation, we have $p = 784$, with $n = 70000$. Again, the $p < \sqrt{n}$ rule of thumb is violated, so dimension reduction should be of value. We will find the low-rank approximation for the data. This also has the advantage of speeding up algorithms that are much slower than logit, such as neural networks.

We will only illustrate one digit here, '5', and only the first steps.

```r
library(WackyData)
data(mnistPred5)
dim(mnistPred5)
```

```
[1] 70000   785
```

```r
# final column is y, 1 or 0 for '5' or not
u <- svd(mnistPred5[,-785])
head(u$d,100)
```

```
  [1] 417574.10 141551.44 131066.28 121191.36 112716.38 101716.94  93479.11
  [8]  83716.56  83066.77  75104.59  71476.99  70398.76  64033.75  63983.27
 [15]  61950.04  59850.43  57243.19  55743.22  53440.71  52615.31  50592.90
 [22]  49245.10  48008.36  46777.03  46077.41  44899.62  44143.54  43434.54
 [29]  42160.13  40705.79  39704.84  39368.55  37987.55  37634.68  36891.21
 [36]  36147.17  34811.89  34197.64  33911.45  33513.38  33062.17  32687.66
 [43]  31757.46  30842.74  30375.39  30040.28  29459.00  28953.76  28554.57
 [50]  27749.48  27681.52  27297.31  26680.43  26397.22  26052.00  25471.55
 [57]  25402.46  24833.91  24658.92  24236.31  24012.67  23941.08  23461.93
 [64]  23033.74  22621.82  22294.68  22071.39  21657.40  21441.10  21279.71
 [71]  21188.01  20887.84  20602.49  20404.25  19894.73  19804.88  19687.80
 [78]  19254.52  18792.30  18478.41  18412.93  18349.67  18308.91  18021.62
 [85]  17834.89  17796.95  17639.25  17339.31  17177.86  17012.03  16728.90
 [92]  16568.33  16454.85  16242.15  16133.17  16042.94  15780.31  15761.63
 [99]  15545.61  15493.61
```

In order to maximize classification accuracy, we would probably need a larger value of $k$, large than the 100 value we tried here. As usual, we could use cross-validation to determine a "big enough" $k$.

## 15.9 Matrix Factorization in Recommender Systems

Now, in the recommender systems context, take $A$ to be the matrix whose row $i$, column $j$ element is the rating user $i$ gives to movie $j$. (We take $A$ to be fully known for now.)

- The columns of $U$ are basically eigenvectors of the matrix $AA'$, which in turn has one row and one column for each moviegoer.
- The columns of $V$ are basically eigenvectors of the matrix $A'A$, which in turn has one row and one column for each movie.
- Thus the SVD $A = U\Sigma V'$ expresses the ratings matrix $A$ in terms of a moviegoers data factor $U$ and a movies data factor $V$. To write it as a genuine product of two factors, write

  $$A = U\Sigma V' = (U\Sigma^{0.5})(\Sigma^{0.5}V') = WH' \tag{15.17}$$

  where $\Sigma^{0.5}$ is the diagonal matrix with elements $\sqrt{\sigma_i}$.

In the last section, our goal was to "thin out" a matrix In this section, we hope to fill in a sparse matrix.

Let $A$ denote the ratings matrix, so that the element in row $i$, column $j$ is the rating user $i$ gives to item $j$. Note that most elements of $A$ are unknown, and we hope to predict them with some reasonable amount of accuracy.

Again, since we do not know all of $A$, we do not know any of the matrices that make up its generalized inverse. We will return to this problem shortly, but for now pretend $A$ and the matrices are known.

Write the SVD, as in Equation 15.17:

$$A = U\Sigma V' = (U\Sigma^{0.5})(\Sigma^{0.5}V') = WH'$$

Say $A$ is $u \times m$, for $u$ users and $m$ movies. Again, the the point is that we have factored $A$ into the product of a matrix $W$ containing information about the users' ratings and a matrix $H'$ that does the same for items.

Note that the row $i$, column $j$ element of $A$ is equal to $w_i h_j$, where $w_i$ is row $i$ of $W$ and $v_j$ is column $j$ of $H'$.

This suggests that the following iterative process may work:

1. Replace the unknown elements of $A$ by some temporary values. This could be all 0s, say, or maybe replacing all unknown values in column $j$ by the mean of the intact values in that column.
2. Find $W$ and $H'$ for that temporary version of $A$, our guess.
3. Use the formula $w_i h_j$ to calculate (new guesses for) the unknown elements, updating accordingly to a new guess for $A$.
4. If convergence not yet reached, go to Step 2.

Optionally in Step 2, we can first convert the SVD to a low-rank approximation before computing $W$ and $H$.

And though SVD provided the motivation for the model $A = WH'$, we can use the model more generally, i.e. without assuming $W = U\Sigma^{0.5}$ and $H' = \Sigma^{0.5}V'$.

For example, one approach is to minimize

$$||A - WH'||^2 + \lambda ||W||^2 + \mu ||H||^2 \tag{15.18}$$

again in the spirit of Chapter 12, with shrinkage parameters $\lambda$ and $\mu$. The computation is made much easier via an *alternating least squares* scheme. One first holds $H$ fixed, minimizing Equation 15.18 with respect to $W$. In effect, this is ridge regresssion, with the matrix $W$ playing an analogous role to the

Note that the "Y" variable here, i.e. the quantity being predicted in the least-squares analysis, is $A$, a matrix rather than the usual vector. The conditional mean is then given by Equation 4.12 if one assumes a multivariate normal settings. That assumption is not an issue, though, as recommender systems methods tend to be *ad hoc* rather that formal.

unknown coefficients vector in ridge. Then in the second iteration, the roles are reversed, with $H'$ now taken as the unknown coefficients vector, and so on.

## 15.10 Using SVD to Gain Insight into Ridge Regression

Consider once again Equation 15.8, or better, Equation 12.2. It turns out that SVD gives us more detailed information of how ridge regression shrinks estimated regression coefficients.

See Ryan Tibshirani, *High-Dimensional Regression: Ridge* and T. Hastie, *Ridge Regularization: an Essential Concept in Data Science.*

Writing as usual $A = U \Sigma V'$, with $A$ of size $n \times p$ and with $\Sigma = diag(\sigma_1, ..., \sigma_p)$, some algebraic manipulation yields the following equation for the estimated coefficients vector:

$$b_\lambda = \Sigma_{\sigma_j > 0} v_j \frac{\sigma_j^2}{\sigma_j^2 + \lambda^2} u_j' S \tag{15.19}$$

where $u_j$ and $v_j$ are column $j$ of $U$ and $V$, as well as an expression for the vector of fitted "Y" values,

$$Ab = \Sigma_{\sigma_j > 0} \frac{\sigma_j^2}{\sigma_j^2 + \lambda^2} u_j u_j' S \tag{15.20}$$

These two equations form great examples of the power of SVD:

- Equation 15.19 gives an explicit formula for the vector of estimated coefficients $b$ as a function of $\lambda$. Without it, we would need to do the matrix inversion in Equation 12.2, once for each value of $\lambda$, potentially a very slow process.
- Ridge does not shrink the coefficients uniformly. Equation 15.20 tells us how it works. The quantity

$$\frac{\sigma_j^2}{\sigma_j^2 + \lambda^2} \leq 1$$

gives us a proportional shrinkage factor. Of course, non-ridge corresponds to $\lambda = 0$, so the shrinkage factor is in comparison to non-ridge. Smaller values of $\sigma_j$ give more shrinkage.

But Equation 15.20 not only gives us insight as to the *amount* of shrinkage but also its *direction*. Essentially, the $u_i$ are the Principal Components of our data, as follows.

A Your Turn problem in Chapter 4 has the reader derive the relation

$$Cov(X) = E(XX') - (EX)(EX)'$$

If we center and scale our variables, this becomes

$$Cov(X) = E(XX')$$

The sample-data based estimate is

$$\frac{1}{n} AA'$$

But we found earlier that the eigenvectors of $AA'$ are the $u_j$!

So, we have:

> Ridge shrinks the vector of fitted "Y" values more in the directions of the Principal Components of our data.

## 15.11 SVD As a Foundation for the Four Fundamental Subspaces

Recall the four subspaces discussed in Chapter 11. We can quickly obtain much information about them from the SVD.

Let $u_i$ and $v_i$ denote the colums of $U$ and $V$. Then:

- $u_1, ..., u_r$ is an orthonormal basis for $\mathcal{C}(A)$

- $u_{r+1}, ..., u_m$ is an orthonormal basis for $\mathcal{R}(A)$
- $v_1, ..., v_r$ is an orthonormal basis for $\mathcal{R}(A)$
- $v_{r+1}, ..., v_n$ is an orthonormal basis for $\mathcal{N}(A)$

## 15.12 Your Turn

**Your Turn:** Write an R function will call form

```
getLowRank(A,lowrank)
```

that uses SVD to find the best approximation to the matrix **A** of rank **lowrank**.

**Your Turn:** Our equation Equation 3.6 resulted from replacing one row (the last, but could have been any) by all 1s, with a corresponding change on the right-hand side. But now that we have pseudoinverses at our disposal, we could *add* a row rather than *replacing* a row:

$$\begin{pmatrix} P \\ 1 \end{pmatrix} \nu = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

where 1 on the left side means a row of 1s, 0 on the right side is a column of 0s, and 1 on the right side means the scalar 1.

Write an R function with call form

```
findNuPseudoInv(p)
```

to implement this approach. Here **p** is a Markov transition matrix. Try your code on a couple of examples.

**Your Turn:** Fill in the missing algebraic steps in Equation 15.20. Hints: Use the fact that $U$ and $V$ are orthogonal matrices, and the fact that $(CD)^{-1} = D^{-1}C^{-1}$ for invertible matrices $C$ and $D$.

**Your Turn:** Consider a diagonal matrix $G = diag(c_1, ..., c_s)$, with the $c_i$ being positive numbers. Show that

$$G^+ = diag(c_1, ..., c_s) \tag{15.21}$$

i.e. that in this case $U$ and $V$ are identity matrices.

# 16 A Deeper Look at Overfitting

Goals of this chapter:

We have mentioned the concept of “overfitting” occasionally in previous chapters. It has always been an issue in statistics, but in recent years it has become an acute problem as datasets with very large number of predictors $p$, and old assumptions are now being questioned. This chapter presents an up-to-date view of the topic, and as usual, linear algebra plays a key role.

## 16.1 Motivating Example

Classically, statistics and ML books have stated that, for fixed number of data points $n$, the graph of predictive accuracy against number of predictor variables $p$ roughly has a U-shape, concave up. Starting at $p = 1$ and then increasing $p$, we achieve better and better predictive power until we reach the minimum point of the U; then adding further predictors degrades performance.

We wish to explore this notion on the Million Song dataset from Section 12.2. Here “Y” is the year of release of a song, stored in the first column with label **V1**. “X” is a set of 90 audio measurements. In the version of the dataset used here, we have 5000 rows.

We ran code like this:

```
sapply(seq(10,90,10),function(p)
   mean(replicate(500,
   qeML::qeLin(yr5000[1:150,1:p],'V1')$testAcc
   )))
```

Here we are running a linear model on the first 150 rows of the dataset (i.e. $n = 150$), and the first $p$ columns, so that $p = 10, 20, ..., 90$. Since **qeML** functions automatically form holdout sets, we obtain the accuracy value, Mean Absolute Prediction Error. And since the holdout set is randomly chosen, we do this 500 times. We graph the result using **qePlotCurves** (which does smoothing).

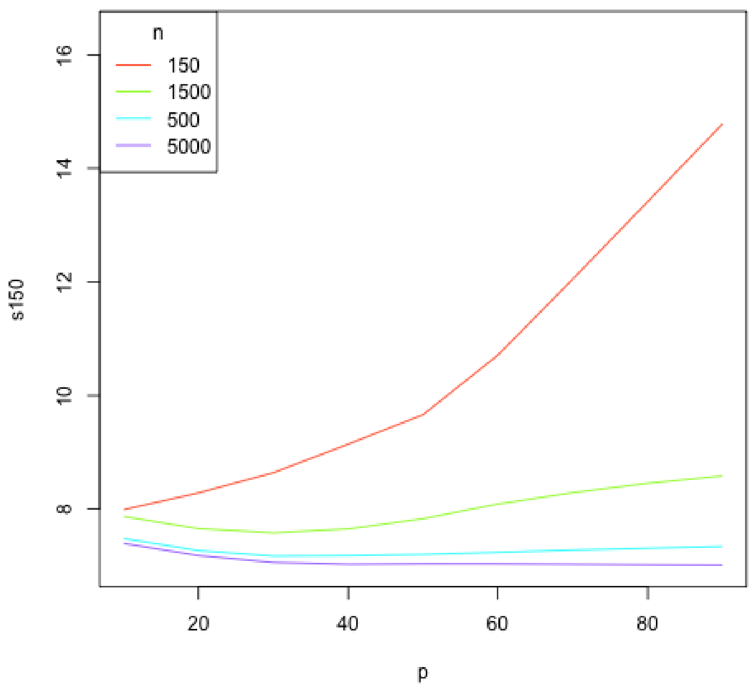


Figure 16.1: U shapes

What do we see here?

- The graphs are indeed U-shaped, albeit often shallower than the typical pictures shown in books.
- The larger the value of $n$,
  - the lower the curve (better prediction accuracy), and
  - the larger the value of the best $p$ (optimal to use more predictors)
- Some are half-U's, indicating it's best to use all variables (large $n$) or one or two (small $n$).

## 16.2 Interpolation Point

Consider a linear regression model, for convenience say with no $\beta_0$ term. As $p$ increases, eventually it becomes equal to $n$, the number of rows. When that happens, we get a "perfect" fit: Our line, plane or hyperplane will pass through every data point, and Equation 5.7 will be 0. Just think of the case $n = 2$, $p = 1$, now with a $\beta_0$ term: Two points in $R^2$, through which we can draw a line passing exactly through them.

This value of $p$ is thus called the *interpolation point*, viewed classically as follows.

- Of course, setting $p$ to the interpolation point would be a terrible fit, unable to predict new data well; we are at the right-hand end of the U.
- And since we already have reached a "perfect" fit, there would be no point in increasing $p$ any further (using a pseudoinverse).

So, pictures of overfititng in books had $p$ stop at $n$. But wow, were they wrong!

## 16.3 Double Descent

Though the prevailing wisdom used to be that, as noted above, there is no point in trying values of $p$ even near $n$, let alone beyond $n$, it turns out that in some cases, the best value of $p$ is in fact larger than $n$.

As a first step to seeing this, consider the following code:

```
overfit <- function(nreps,n,maxP)
{

   require(qeML)
   load('YearData.save')  # 500K Million Song data
   nas <- rep(NA,nreps*(maxP-1))
   # record p, Mean Absolute Prediction Error
   outdf <- data.frame(p=nas,mape=nas)
   rownum <- 0
   for (i in 1:nreps) {
      idxs <- sample(1:nrow(yr),n)
      trn <- yr[idxs,]
      tst <- yr[-idxs,]
      for (p in 2:maxP) {
         rownum <- rownum + 1
         out<-qePolyLin(trn[,1:(p+1)],
            'V1',2,holdout=NULL)
         preds <- predict(out,tst[,-1])
         mape <- mean(abs(preds - tst[,1]))
         outdf[rownum,1] <- p
         outdf[rownum,2] <- mape
         print(outdf[rownum,])
      }
   }
   outdf
}

z <- overfit(10,250,30)
w <- tapply(z$mape,z$p,mean)
plot(w)
```

This code tries values of **p** up to **maxP** but one must note that we are fitting quadratic models. For any given value of **p**, the number of columns after polynomial expansion will be much larger. Consider this code:

```
> u22 <- qePolyLin(yr[1:100,1:23],'V1',deg=2)
P > N. With polynomial terms and interactions, P is 265.
> u22 <- qePolyLin(yr[1:100,1:22],'V1',deg=2)
P > N. With polynomial terms and interactions, P is 246.
```

So $p = 21$ will upon polynomial expansion produce 246 columns, while $p = 22$ will produce 265. We won't hit 250 exactly, but the interpolation point will be in this range.

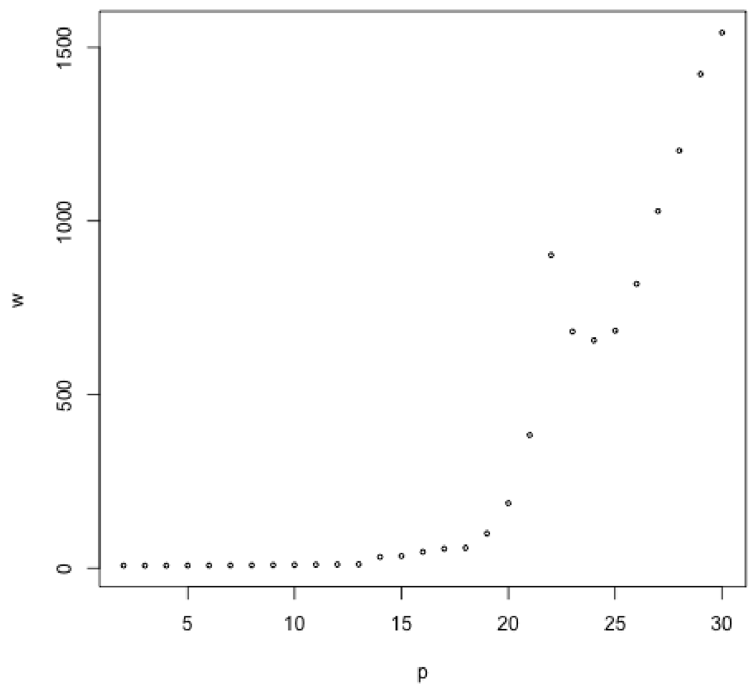


Figure 16.2: U shapes

After the interpolation point, we see another U! Hence the name *double descent.* This introduces the concept, but the big news is:

> The minimum of the second U is sometimes lower than that of the first. In other words, it can pay to overfit!

Let's see how linear algebra plays a role in explaining how double descent can occur..

## 16.4 Plausibility of Double Descent: the Minimum Norm Solution

In an overdetermined linear system ($p > n$) such as Equation 15.8, there are many solutions. Our **qePolyLin** function calls **regtools::penrosePoly**. An advantage of Moore-Penrose is that it gives us the *minimum norm* solution. We'll discuss the significance of this for Double Descent shortly, but let's prove it first.

### 16.4.1 Proof of the minimum-norm property

We will need this:

**Theorem 16.1** (Multiplication by an Orthogonal Matrix Preserves Norm)**.** *For an orthogonal matrix $M$ and a vector $w$, $||Mw|| = ||w||$.*

*Proof.* See the Your Turn problem below. □

**Theorem 16.2** (The Moore-Penrose Solution Is Min-Norm)**.** *Consider an $m \times n$ matrix $B$ and vector $q$ of length $m$. Of all solutions $x$ (of length $n$) to*

$$Bx = q \tag{16.1}$$

*the Moore-Penrose solution minimizes $||x||$.*

In our linear regression context Equation 15.8, $B = A'A$, $x = b$ and $q = A'S$.

*Proof.* Again, write

Adapted in part from a derivation by Jose' C. Feriero.

$$B = U\Sigma V',$$

where $U$ is $m \times n$ and $\Sigma$ and $V$ are $n \times n$. (For our discussion of overfitting, we will primarily be interested in the case $n > m$.)

Consider the residual sum of squares

$$||Bx - q||^2 = ||U\Sigma V'x - q||^2 \tag{16.2}$$

so

$$||Bx - q||^2 = ||U(\Sigma V'x - U'q)||^2 \tag{16.3}$$

since $UU' = I$.

From Theorem 16.1 we can remove the factor $U$ in Equation 16.3, yielding

$$||Bx - q||^2 = ||\Sigma V'x - U'q||^2 \tag{16.4}$$

Rename $V'x$ to $w$ and $U'q$ to $s$:

$$||Bx - q||^2 = ||\Sigma w - s||^2 \tag{16.5}$$

So, instead of finding $x$ to minimize the residual sum of squares $||Bx - q||^2$, we've (temporarily) changed the problem to that of finding $w$ to minimize

$$||\Sigma w - s||^2 \tag{16.6}$$

This version of the original problem is more easily solved, since we are dealing with a diagonal matrix. Setting the derivative of Equation 16.6 with respect to $w$ to 0, we have

$$\Sigma^2 w = \Sigma s$$

Thus (illustrating for $B$ of rank $r = 2$)

$$\left( \begin{array}{cccc} \sigma_1 & 0 & 0 & ... \\ 0 & \sigma_2 & 0 & ... \\ 0 & 0 & 0 & ... \\ & ... & & \end{array} \right) \left( \begin{array}{c} w_1 \\ w_2 \\ 0 \\ ... \\ 0 \end{array} \right) = \left( \begin{array}{c} s_1 \\ s_2 \\ 0 \\ ... \\ 0 \end{array} \right)$$

(Due to the structure of $\Sigma$ here, it doesn't matter what $w_i, i > r$ is; let's make them 0s.)

So we see that

$$w_i = \frac{1}{\sigma_i}, i = 1, .., r$$

In other words, making use of Equation 15.21,

$$w = \Sigma^+ s$$

Recalling that $V'x = w$ and $U'q = s$, we have

$$\begin{aligned} x &= Vw && (16.7) \\ &= V\Sigma^+ s && (16.8) \\ &= V\Sigma^+ U'q && (16.9) \\ & && (16.10) \end{aligned}$$

Wrapping up, we have

$$x = V\Sigma^+ U'q$$

Exactly the Moore-Penrose inverse!

□ □

### 16.4.2 Connection to Double Descent

First, consider the minimum norm solution $x_{min}$ in our last section, with $p > n$, i.e. post-interpolation. Intuitively, shorter solutions have less variability, i.e. smaller variance. Moreover, all the solutions have a 0 residual sum of squares, so why not take the one with smaller variance? It is thus plausible that $x_{min}$ might do better than the value of $x$ we obtain at interpolation, hence a downward trend in the curve of predictive accuracy against $p$. On the other hand, as $p$ grows even further,

$x$ has more and more components, and likely that minimum-norm $x$ will grow in length at some point, causing an increase in variance, hence the typical U-shape – as our *second* U.

In addition, it can be shown that if *gradient descent* is used to minimize the least-squares problem, the tendency is that the result will be of minimum norm.

## 16.5 Plausibility of Double Descent: the Condition Number

We briefly mentioned the *condition number* of a square matrix $A$ back in Section 13.7, the ratio of the maximal and minimal eigenvalues. High values suggest problems of multicollinearity.

If $A$ is nonsquare, as in our setting in this chapter, one can apply the same criterion to $AA'$. Then we can go through the same exercise as above, starting with $p = 1$, i.e. using only the first column of $A$, then adding more and more columns. It turns out that if we graph condition number against $p$, we again see Double Descent behavior! And again, the peak will occur at the interpolation point.

As before, once $p$ passes the interpolation point, we need to use the pseudoinverse or equivalent. Also, now the denominator will be the smallest *nonzero* eigenvalue

Here is the code, run on the Million Song data:

```
cn <-
function(a)
{
   aap <- a %*% t(a)
   evs <- eigen(aap)$values
   max(evs) / min(evs[evs > 1.0e-02])
}

u <- seq(55,85,1)
z <- sapply(u,function(p) cn(yr[1:75,1:p]))
plot(u,z,cex=0.5)
```

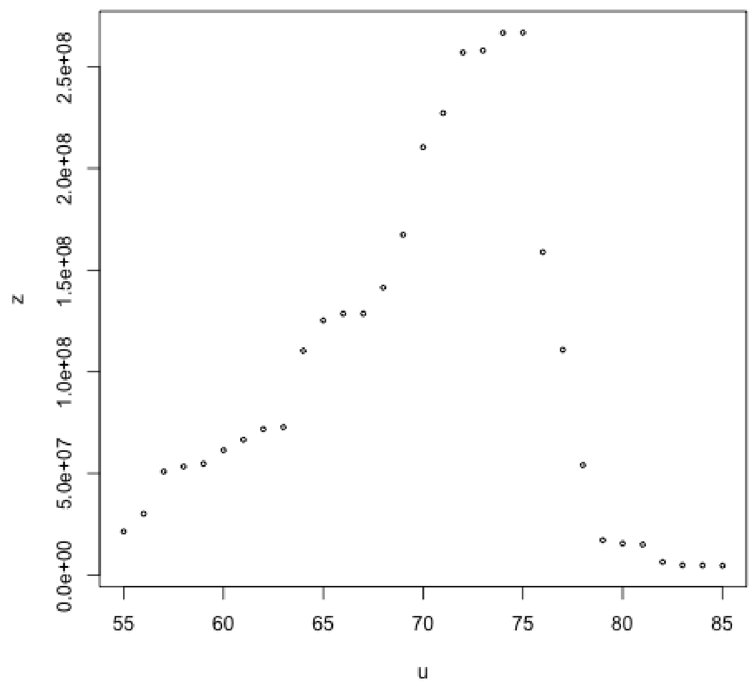


Figure 16.3: Condition number

There it is, a peak at $n = p$.

Now, how might this relate to the original (accuracy vs. $p$ Double Descent problem? Recall that a large condition number indicates multicollinearity, which in turn causes high variance in $\hat{\beta}$, thus impaired predictive ability.

## 16.6 Your Turn

**Your Turn:** Show that for an orthogonal matrix $M$ and a vector $w$, $||Mw|| = ||w||$.

**Your Turn:** Using properties of matrix rank, show that if $p + 1 > n$ in Equation 5.12, the inverse will not exist.

# 17 Attention

Goals of this chapter:

The emergence of Large Language Models (LLMs) really startled the general public, leading to an explosion of public attention to the field. In turn, the pivotal research paper behind the success of LLMs was *Attention Is All You Need*, by Ashish Vaswani *et al*, 2017. Actually, the mathematical operation called *attention* is rather simple in its foundation. This approach is in turn related to material that has arisen here in earlier chapters. Here we explore the basics of this idea.

## 17.1 Dot Product as a Measure of Similarity

As discussed in Section 12.1.1, the row $i$, column $j$ element of the matrix $A'A$ in Equation 5.10 is a measure of the strength of relation between predictor variables $i$ and $j$. And since that matrix element is the dot product between rows $i$ and $j$, we see that dot products serve as measures of similarity.

Recall the key role of dot products in Section 12.14, with kernels acting like generalized dot products. One of the reasons dot products are so important is that they serve as measures of similarity.

This point also arises in recommender systems. Recall Section 2.10:

> In order to assess interuser similarity of the nature described above, we might form a matrix , as follows. There would be 943 rows, one for each user, and 1682 columns, one for each movie. The element in row , column would be the rating user gave to movie . Most of the matrix would be 0s.
>
> Determining the similarity of users becomes a matter of measuring similarity of rows of the matrix. This paves the way to exploiting the wealth of matrix-centric methodology we have developed in this book.

And *how* might this similarity between users be measured? By dot product, of course!

## 17.2 Revisiting the Linear Model

Before going to the LLM usage of attention, it will be instructive to view the familiar linear regression model in terms of attention (aside from the point regarding $A'A$ above). Here we follow *Ordinary Least Squares as an Attention Mechanism*, by Philippe Goulet Coulombe, 2025.

One again, consider the model

$$E(S|A) = A\beta$$

whose least-squares solution is

$$\hat{\beta} = (A'A)^{-1}A'S$$

where

- $A$, $n \times p$, is our predictor variable data, column $i$ containing predictor $i$, "X",
- $S$, $n \times 1$, is our response variable data, "Y", and
- $\beta$, $p \times 1$, is a population vector, with sample estimate as above.

Alluding to the fact that fitting a model is often referred to as "training" the model, below we will use the notation $A_{train}$ and $Y_{train}$ for $A$ and $S$.

After computing $\hat{\beta}$, we are ready to predict new cases! Say we have $m$ of them, stored in the $m \times p$ matrix $A_{new}$. Our predictions are

$$Y_{preds} = A_{new}\hat{\beta} = A_{new}(A'A)^{-1}A'_{train}Y_{train}$$

As in Equation 14.5, write

$$A'A = PDP'$$

At this point, we turn to the notion of the "square root" of a symmetric matrix, introduced in Section 14.6. Write

$$Y_{preds} = (A_{new}PD^{-0.5})\ (D^{-0.5}P'A'_{train})\ Y_{train}$$

so

$$Y_{preds} = (F_{new}F'_{train})\ Y_{train} \tag{17.1}$$

where $F_{new}$ and $F_{train}$ are $m \times p$ and $p \times n$.

Things to note about Equation 17.1:

- The matrix $F_{new}F'_{train}$ transforms our training "Y" values into predicted "Y" values.
- Each predicted "Y" value is a linear combination of the training "Y" values.
- Thus the predicted values are *weighted* sums of the training values.
- The weights matirx $F_{new}F'_{train}$ factors neatly into a product of a matrix involving the new data and a matrix involving the training data.

These last two bullet points share the foundation of the attention concept.

## 17.3 Attention

Though the concept of attention is simple, the application can be quite complex. Our presentation here will merely provide an overview.

## 17.3.1 Basic structures

- There is a *query* matrix $Q$, analogous to $F_{new}$.
- There is a *keys* matrix $K$ analogous to $F_{train}$.
- There is a *values* matrix $V$ analogous to $Y_{train}$.

Major differences from the last section:

- The data in $Q$ and $K$ are *ordered sequences*, such as words within a sentence, genes within a chromosome or daily stock market prices in a time series.
- Rather than predicting a numerical quantity, these applications are driven by probabilities of sentences or other sequences.

The fundamental computations consist of dot products of rows of $Q$ and $K$.

## 17.3.2 Iterative computation

The word "attention" alludes to the weighting. Heavier weights are given to more important parts of the input sequence.

Recall that in motivating the least-squares approach to estimation for the linear regression model in Section 5.1.1, we said, "Pretend for a moment that we don't know, say, $C_{28}$," the latter being one of the "Y" values in our training set. We predict that value with our model, and evaluate our error.

The idea is the same with LLMs, though as with most machine learning algorithms, the computation is iterative. We take a query from our training set, and predict "Y" value using our current weights. At each iteration, the current guesses for probabilities in $V$ are updated.

(As noted, this involves clever updating algorithms, using tremendous amounts of computation, whose details are beyond the scope of this book.)

### 17.3.3 Next-token prediction

The items in a sequence, say words in a sentence or genes in a chromosome, are called *tokens.* During training, we step through a query sequence, one token at a time. At each step, we predict the next token according to which would make the sentence most probable, given our current guess for $V$. After building up sequences in this manner, we build up a candidate full sequence. After doing this step for all sequences in $Q$, we check to see how well we predicted. We then use this to update all in our next iteration.

The idea of next-token prediction is easiest to understand in an application like translating English test to, say, French. Here we input a sentence in English, and output a sentence in French; our output is the one determined to have the highest probability.