# From Linear Models to Machine Learning

## a Modern View of Statistical Regression and Classification

**Norman Matloff**

**University of California, Davis**

# Contents

# Preface

Regression analysis is both one of the oldest branches of statistics, with *least-squares* analysis having been first proposed way back in 1805. But ironically, it is also one of the newest areas, in the form of the *machine learning* techniques being vigorously researched today. Not surprisingly, then, there is a vast literature on the subject.

Well, then, why write yet another regression book? Many books are out there already, with titles using words like *regression*, *classification*, *predictive analytics*, *machine learning* and so on. They are written by authors whom I greatly admire, and whose work I myself have found useful. Yet, I did not feel that any existing books covered the material in a manner that sufficiently provided insight for the practicing data analyst. Too many equations, too few explanations.

Merely including examples with real data is not enough to truly tell the story in a way that will be useful in practice. Few if any books go much beyond presenting the formulas and techniques, and thus the hapless practitioner is largely left to his/her own devices. Too little is said in terms of what the concepts really mean in a practical sense, what can be done with regard to the inevitable imperfections of our models, which techniques are too much the subject of "hype," and so on.

This book aims to remedy this gaping deficit. It develops the material in a manner that is mathematically precise yet always maintains as its top priority — borrowing from a book title of the late Leo Breiman — "a view toward applications."

In other words:

> The philosophy of this book is to not only prepare the analyst to know *how* to do something, but also to understand *what* she is doing. For successful application of data science techniques,

the latter is just as important as the former.

**Examples of what is different here:**

How is this book is different from all other regression books? Here are a few examples:

- *A recurring interplay between parametric and nonparametric methods.*

  On the one hand, the book explains why parametric methods can be much more powerful than their nonparametric cousins if a reasonable model can be developed, but on the other hand it shows how to use nonparametric methods effectively in the absence of a good parametric model. The book also shows how nonparametric analysis can help in parametric model assessment. In the chapter on selection of predictor variables (Chapter 9, Dimension Reduction), the relation of number of predictors to sample size is discussed in both parametric and nonparametric realms.

- *In-depth treatment of the Description aspect of regression analysis.*

  A well-known point, made in many books, is that in addition to the vital Prediction goal of regression analysis, there is an equally-important Description goal. This book devotes an entire chapter to the latter (Chapter 7, Disaggregating Factor Effects). After an in-depth discussion of the interpretation of coefficients in parametric regression models, and a detailed analysis (and even a resolution) of Simpson's Paradox, the chapter then turns to the problem of comparing groups in the presence of covariates — updating the old *analysis of covariance*. In addition, novel regression-based methods for Small Area Estimation and Propensity Matching are presented.

- *Special issues in classification settings.*

  Classification methods are discussed throughout the book, intertwined with general regression methodology. This reflects the fact that classification can be viewed as a special case of regression. The conditional mean — regression function – simply becomes the conditional probability of Class 1 in a two-class classification problem.

  However, a number of issues arise that are specifici to multi-class settings. So here again there is an entire chapter on this topic (Chapter 5, Multiclass Classification Problems). For instance, the One vs. All and All vs. All approaches are compared, as well as the issue of "unbalanced" class data. The treatment is both parametric and nonparametric.

- *To use Method X or not use it — that is the question.*

  A number of sections in the book are titled, "The Verdict," suggesting to the practitioner which among various competing methods might be the most useful. Consider for instance the issue of *heteroscedasticity*, in which the variance of the response variable is nonconstant across covariate values. After showing that the effects on statistical inference are perhaps more severe than many realize, the book presents various solutions: Weighted least squares (including nonparametric estimation of weights); the Eicker-White method; and variance-stabilizing transformations. The section titled "The Verdict" then argues for opting for the Eicker-White model if the goal is Description, and ignoring the problem if the goal is Prediction.

  A related issue, variance-stabilizing transformations, epitomizes the philosophy of the book. While virtually every regression book presents this method, few discuss whether it is a good idea; we do so here.

- *A general aim toward a unified, modern approach to the field.*

  Note too that the book aims to take a unified approach to the various aspects — regression and classification, parametric and nonparametric approaches, methodology developed in both the statistics and machine learning communities, and so on. The aforementioned use of nonparametrics to help assess fit in parametric models exemplifies this.

- *Treatment of Big Data settings.*

  These days there is much talk about Big Data. Though it is far from the case that most data these days is Big Data, on the other hand it is true that things today are indeed quite different from the days of "your father's regression book."

  Perhaps the most dramatic of these changes is the emergence of data sets with very large numbers of predictor variables $p$, as a fraction of $n$, the number of observations. Indeed, for some data sets $p >> n$, an extremely challenging situation. Chapter 9, Dimension Reduction, covers not only "ordinary" issues of variable selection, but also this important newer type of problem, for which many solutions have been proposed.

**A comment on the field of machine learning:**

Mention should be made of the fact that this book's title includes both the word *regression* and the phrase *machine learning*.

When China's Deng Xiaoping was challenged on his then-controversial policy of introducing capitalist ideas to China's economy, he famously said, "Black cat, white cat, it doesn't matter as long as it catches mice." Statisticians and machine learning users should take heed, and this book draws upon both fields, which at core are not really different from each other anyway.

My own view is that machine learning (ML) consists of the development of regression models with the Prediction goal. Typically nonparametric methods are used. Classification models are more common than those for predicting continuous variables, and it is common that more than two classes are involved, sometimes a great many classes. All in all, though, it's still regression analysis, involving the conditional mean of $Y$ given $X$ (reducing to $P(Y = 1|X)$ in the classification context).

One often-claimed distinction between statistics and ML is that the former is based on the notion of a sample from a population whereas the latter is concerned only with the content of the data itself. But this difference is more perceived than real. The idea of cross-validation is central to ML methods, and since that approach is intended to measure how well one's model generalizes beyond our own data, it is clear that ML people do think in terms of samples after all.

So, at the end of the day, we all are doing regression analysis, and this book takes this viewpoint.

**Intended audience:**

This book is aimed at both practicing professionals and use in the classroom. It aims to be both accessible and valuable to such diversity of readership.

Minimal background: The reader must of course be familiar with terms like *confidence interval*, *significance test* and *normal distribution*, and is assumed to have knowledge of basic matrix algebra, along with some experience with R. Many readers will have had at least some prior exposure to regression analysis, but this is not assumed, and the subject is developed from the beginning. Very elementary notions of expected value are assumed, but math stat is needed only for readers who wish to pursue the Mathematical Complements sections at the end of most chapters. Many chapters also have Computational Complements sections, which present advanced details of the usage of R and other computational issues.

The book can be used as a text at either the undergraduate or graduate level. At the undergraduate level, a basic course might cover Chapters 1, 2,

3, 5, 6 and 7, plus further topics as time permits. A graduate-level course with an undergraduate prerequisite might focus on the remaining chapters, plus the Mathematical Complements sections of all of the chapters.

Those who wish to use the book as a course text should find that all their favorite topics are here, just organized differently and presented in a fresh, modern point of view.

There is little material on Bayesian methods (meaning subjective priors, as opposed to empirical Bayes). This is partly due to author interest, but also because the vast majority of R packages for regression and classification do not take a Bayesian approach. However, armed with the solid general insights into predictive statistics that this book hopes to provide, the reader would find it easy to go Bayesian in this area.

**Code and Software:**

Code is displayed explicitly wherever feasible, as this makes concepts more concrete, and facilitates more "hands-on" learning.

In most cases, data wrangling/data cleaning code is shown, not only for the purpose of 'hands-on" learning, but also to highlight the importance of those topics.

The book also makes use of some of my research results and associated software. The latter is in my package **regtools**, available from GitHub.

In many cases, code is also displayed within the text, so as to make clear exactly what the algorithms are doing.

**Thanks**

Conversations with a number of people have directly or indirectly enhanced the quality of this book, among them Charles Abromaitis, Stuart Ambler, Doug Bates, Oleksiy Budilovsky, Yongtao Cao, Frank Harrell, Benjamin Hofner, Jiming Jiang, Michael Kane, Hyunseung Kang, Erin McGinnis, John Mount, Art Owen, Ariel Shin, Yu Wu, Yingkang Xie, Achim Zeileis and Jiaping Zhang.

Thanks go to my editor, John Kimmel, for his encouragement and patience, and to the internal reviewers, David Giles and ... Of course, I cannot put into words how much I owe to my wonderful wife Gamis and my daughter Laura, both of whom inspire all that I do, including this book project.

**A final comment:**

My career has evolved quite a bit over the years. I wrote my dissertation in

abstract probability theory, but turned my attention to applied statistics soon afterward. I was one of the founders of the Department of Statistics at UC Davis, but a few years later transferred into the new Computer Science Department. Yet my interest in regression has remained constant throughout those decades.

I published my first research papers on regression methodology way back in the 1980s, and the subject has captivated me ever since. My long-held wish has been to write a regression book, and thus one can say this work is 30 years in the making. I hope you find its goals both worthy and attained. Above all, I simply hope you find it an interesting read.

# Chapter 1

# Setting the Stage

This chapter will set the stage for the book, previewing many of the major concepts to be presented in later chapters. The material here will be referenced repeatedly throughout the book.

## 1.1 Example: Predicting Bike-Sharing Activity

Let's start with a well-known dataset, **Bike Sharing**, from the Machine Learning Repository at the University of California, Irvine.[1] Here we have daily/hourly data on the number of riders, weather conditions, day-of-week, month and so on. Regression analysis, which relates the mean of one variable to the values of one or more other variables, may turn out to be useful to us in at least two ways:

- **Prediction:**

  The managers of the bike-sharing system may wish to predict ridership, say for the following question:

  > Tomorrow, Sunday, is expected to be sunny and cool, say 62 degrees Fahrenheit. We may wish to predict the number of riders, so that we can get some idea as to how many bikes will need repair. We may try to predict ridership, given the

[1] Available at `https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset`.

weather conditions, day of the week, time of year and so on.

- **Description:**

  We may be interested in determining what factors affect ridership. How much effect, for instance, does wind speed have in influencing whether people wish to borrow a bike?

These twin goals, Prediction and Description, will arise frequently in this book. Choice of methodology will often depend on the goal in the given application.

## 1.2 Example of the Prediction Goal: Bodyfat

*Prediction is difficult, especially about the future* — baseball great, Yogi Berra

The great baseball player, Yogi Berra was often given to malapropisms, one of which was supposedly was the quote above. But there is more than a grain of truth to this, because indeed we may wish to "predict" the present or even the past.

For example, consiser the **bodyfat** data set, available in the R package, **mfp**, available on CRAN. (See Section 1.20.1 for information on CRAN packages, a numbr of which will be used in this book.) Body fat is expensive and unwieldy to measure directly, as it involves underwater weighing. Thus it would be highly desirable to "predict" that quantity from easily measurable variables such as height, age, weight, abdomen circumference and so on.

In scientific studies of ancient times, there may be similar situations in which we "predict" unknown quantities from known ones.

## 1.3 Example of the Description Goal: Who Clicks Web Ads?

One of the most common applications of machine learning methods is in marketing. Sellers wish to learn which types of people might be interested

in a given product. The reader is probably familiar with Amazon's *recommender system*, in which the viewer who indicates interest in a given book, say, is shown a list of similar books.[2]

We will discuss recommender systems at several points in this book, beginning with Section 3.2.4. A more general issue is the *click-through rate* (CTR), meaning the proportion of viewers of a Web page who click on a particular ad on the page. A simple but very engaging example was discussed online (R-statistics Blog, 2010). The data consist of one observation per state of the U.S. There was one predictor, the proportion of college graduates in the state, and a response variable, the *click-through rate*.

One approach to learning what relation, if any, educational level has to CTR would be to use regression analysis. We will see how to do so in Section 1.8.

## 1.4 Optimal Prediction

Even without any knowledge of statistics, many people would find it reasonable to predict via subpopulation means. In the above bike-sharing example, say, this would work as follows.

Think of the "population" of all days, past, present and future, and their associated values of number of riders, weather variables and so on.[3] Our data set is considered a sample from this population. Now consider the subpopulation consisting of all days with the given conditions: Sundays, sunny skies and 62-degree-temperatures.

It is intuitive that:

> A reasonable prediction for tomorrow's ridership would be the mean ridership among all days in the subpopulation of Sundays with sunny skies and 62-degree-temperatures.

In fact, such a strategy is optimal, in the sense that it minimizes our expected squared prediction error. We will defer the proof to Section 1.19.3 in the Mathematical Complements section at the end of this chapter, but

[2] As a consumer, I used to ignore these, but not with the sharp decline in the number of bricks-and-mortar bookstores which I could browse, I now often find Amazon's suggestions useful.

[3] This is a somewhat slippery notion, because there may be systemic differences from the present and the distant past and distant future, but let's suppose we've resolved that by limiting our time range.

what is important for now is to note that in the above prediction rule, we are dealing with a conditional mean: Mean ridership, given day of the week is Sunday, skies are sunny, and temperature is 62.

## 1.5 A Note About E(), Samples and Populations

To make this more mathematically precise, keep in mind that in this book, as with many other books, the *expected value* functional $E()$ refers to population mean. Say we are studying personal income, $I$, for some population, and we choose a person at random from that population. Then $E(I)$ is not only the mean of that random variable, but much more importantly, it is the mean income of all people in that population.

Similarly, we can define condition means, i.e., means of subpopulations. Say $G$ is gender. Then the conditional expected value, $E(I \mid G = \text{ male})$ is the mean income of all men in the population.

To illustrate this in the bike-sharing context, let's define some variables:

- $R$, the number of riders
- $W$, the day of the week
- $S$, the sky conditions, e.g. sunny
- $T$, the temperature

We would like our prediction $\widehat{R}$ to be[4] the conditional mean,

$$\widehat{R} = E(R \mid W = \text{ Sunday}, S = \text{ sunny}, T = 62) \tag{1.1}$$

There is one major problem, though: We don't know the value of the right-hand side of (1.1). All we know is what is in our sample data, whereas the right-side of (1.1) is a population value, and thus unknown.

**The difference between sample and population is of course at the very core of statistics.** In an election opinion survey, for instance, we wish to know $p$, the proportion of people in the population who plan to vote for Candidate Jones. But typically only 1200 people are sampled, and

[4]Note that the "hat" notation ˆ is the traditional one for "estimate of."

we calculate the proportion of Jones supporters among them, $\widehat{p}$, using that as our estimate of $p$. This is why the news reports on these polls always report the *margin of error*,[5]

Similarly, though we would like to know the value of $E(R \mid W = \text{ Sunday}, S = \text{ sunny}, T = 62)$, **it is an unknown population value, and thus must be estimated from our sample data**, which we'll do later in this chapter.

**Readers will greatly profit from constantly keeping in mind this distinction between populations and samples.**

Before going on, a bit of terminology: We will refer to the quantity to be predicted, e.g. $R$ above, as the *response variable*, and the quantities used in prediction, e.g. $W$, $S$ and $T$ above, as the *predictor variables*. (By the way, the machine learning community uses the term *features* rather than *predictors*.)

## 1.6 Example: Do Baseball Players Gain Weight As They Age?

Though the bike-sharing data set is the main example in this chapter, it is rather sophisticated for introductory material. Thus we will set it aside temporarily, and bring in a simpler data set for now. We'll return to the bike-sharing example in Section 1.13.

This new dataset involves 1015 major league baseball players, courtesy of the UCLA Statistics Department. You can obtain the data either from the UCLA Web page, or as the data set **mlb** in **freqparcoord**, a CRAN package authored by Yingkang Xie and myself. The variables of interest to us here are player weight $W$, height $H$ and age $A$, especially the first two.

Here are the first few records:

```
> library(freqparcoord)
> data(mlb)
> head(mlb)
                 Name Team         Position Height
1    Adam_Donachie  BAL          Catcher     74
2        Paul_Bako  BAL          Catcher     74
3 Ramon_Hernandez  BAL          Catcher     72
4     Kevin_Millar  BAL  First_Baseman     72
```

[5] Actually the radius of a 95% confidence interval for $p$.

```
5      Chris_Gomez   BAL   First_Baseman      73
6    Brian_Roberts   BAL  Second_Baseman      69
  Weight    Age PosCategory
1    180  22.99      Catcher
2    215  34.69      Catcher
3    210  30.78      Catcher
4    210  35.43    Infielder
5    188  35.71    Infielder
6    176  29.39    Infielder
```

### 1.6.1 Prediction vs. Description

Recall the Prediction and Description goals of regression analysis, discussed in Section 1.1. With the baseball player data, we may be more interested in the Description goal, such as:

> Ahtletes strive to keep physically fit. Yet even they may gain weight over time, as do people in the general population. To what degree does this occur with the baseball players? This question can be answered by performing a regression analysis of weight against height and age, which we'll do in Section 1.9.1.2.[6]

On the other hand, there doesn't seem to be much of a Prediction goal here. It is hard to imagine much need to predict a player's weight. One example of such is working with missing data, in which we wish to predict any value that might be unavailable.

However, for the purposes of explaining the concepts, we will often phrase things in a Prediction context. In the baseball player example, it will turn out that by trying to predict weight, we can deduce effects of height and age. In particular, we can answer the question posed above concerning weight gain over time.

So, suppose we will have a continuing stream of players for whom we only know height (we'll bring in the age variable later), and need to predict their weights. Again, we will use the conditional mean to do so. For a player of height 72 inches, for example, our prediction might be

$$\widehat{W} = E(W \mid H = 72) \tag{1.2}$$

[6]The phrasing here, "regression analysis of ... against ...," is commonly used in this field. The quantity before "against" is the response variable, and the ones following are the predictors.

Again, though, this is a population value, and all we have is sample data. How will we estimate $E(W \mid H = 72)$ from that data?

First, some important notation: Recalling that $\mu$ is the traditional Greek letter to use for a population mean, let's now use it to denote a function that gives us subpopulation means:

> For any height $t$, define
>
> $$\mu(t) = E(W \mid H = t) \tag{1.3}$$
>
> which is the mean weight of all people in the population who are of height $t$.
>
> Since we can vary $t$, this is indeed a function, and it is known as *the regression function of W on H*.

So, $\mu(72.12)$ is the mean population weight of all players of height 72.12, $\mu(73.88)$ is the mean population weight of all players of height 73.88, and so on. These means are population values and thus unknown, but they do exist.

So, to predict the weight of a 71.6-inch tall player, we would use $\mu(71.6)$ — if we knew that value, which we don't, since once again this is a population value while we only have sample data. So, we need to estimate that value from the (height, weight) pairs in our sample data, which we will denote by $(H_1, W_1), ...(H_{1015}, W_{1015})$. How might we do that? In the next two sections, we will explore ways to form our estimate, $\widehat{\mu}(t)$.

### 1.6.2 A First Estimator, Using a Nonparametric Approach

Our height data is only measured to the nearest inch, so instead of estimating values like $\mu(71.6)$, we'll settle for $\mu(72)$ and so on. A very natural estimate for $\mu(72)$, again using the "hat" symbol to indicate "estimate of," is the mean weight among all players in our sample for whom height is 72, i.e.

$$\widehat{\mu}(72) = \text{ mean of all } W_i \text{ such that } H_i = 72 \tag{1.4}$$

R's **tapply()** can give us all the $\widehat{\mu}(t)$ at once:

```
> library(freqparcoord)
> data(mlb)
> muhats <- tapply(mlb$Weight,mlb$Height,mean)
> muhats
      67       68       69       70       71       72
172.5000 173.8571 179.9474 183.0980 190.3596 192.5600
      73       74       75       76       77       78
196.7716 202.4566 208.7161 214.1386 216.7273 220.4444
      79       80       81       82       83
218.0714 237.4000 245.0000 240.5000 260.0000
```

In case you are not familiar with **tapply()**, here is what just happened. We asked R to partition the Weight variable into groups according to values of the Height variable, and then compute the mean weight in each group. So, the mean weight of people of height 72 in our sample was 192.5600. In other words, we would set $\widehat{\mu}(72) = 192.5600$, $\widehat{\mu}(74) = 202.4566$, and so on. (More detail on **tapply()** is given in the Computational Complements section at the end of this chapter.)

Since we are simply performing the elementary statistics operation of estimating population means from samples, we can form confidence intervals (CIs). For this, we'll need the "n" and sample standard deviation for each height group:

```
> tapply(mlb$Weight,mlb$Height,length)
 67  68  69  70  71  72  73  74  75  76  77  78
  2   7  19  51  89 150 162 173 155 101  55  27
 79  80  81  82  83
 14   5   2   2   1
> tapply(mlb$Weight,mlb$Height,sd)
      67       68       69       70       71       72
10.60660 22.08641 15.32055 13.54143 16.43461 17.56349
      73       74       75       76       77       78
16.41249 18.10418 18.27451 19.98151 18.48669 14.44974
      79       80       81       82       83
28.17108 10.89954 21.21320 13.43503       NA
```

Here is how that first call to **tapply()** worked. Recall that this function partitions the data by the Height variables, resulting in a weight vector for each height value. We need to specify a function to apply to each of those vectors, which in this case we choose to be R's **length()** function. The latter then gives us the count of weights for each height value, the "n" that we need to form a CI.

An approximate 95% CI for $\mu(72)$, for example, is then

$$190.3596 \pm 1.96 \frac{17.56349}{\sqrt{150}} \tag{1.5}$$

or about (187.6,193.2).

The above analysis takes what is called a *nonparametric* approach. To see why, let's proceed to a parametric one, in the next section.

### 1.6.3 A Possibly Better Estimator, Using a Linear Model

*All models are wrong, but some are useful* — famed statistician George Box

So far, we have assumed nothing about the shape that $\mu(t)$ would have, if it were plotted on a graph. Again, it is unknown, but the function does exist, and thus it does correspond to *some* curve. But we might consider making an assumption on the shape of this unknown curve. That might seem odd, but you'll see below that this is a very powerful, intuitively reasonable idea.

Toward this end, let's plot those values of $\widehat{\mu}(t)$ we found above. We run

```
> plot(67:83,muhats)
```

producing Figure 1.1.

Interestingly, the points in this plot seem to be near a straight line, suggesting that our unknown function $\widehat{\mu}(t)$ has a linear form, i.e. that

$$\mu(t) = c + dt \tag{1.6}$$

for some constants $c$ and $d$, over the range of $t$ appropriate to human heights. Or, in English,

$$\text{mean weight} = c + d \times \text{ height} \tag{1.7}$$

Don't forget the word *mean* here! We are assuming that the mean weights in the various height subpopulations have the form (1.6), NOT that weight itself is this function of height, which can't be true.

This is called a *parametric* model for $\mu(t)$, with parameters $c$ and $d$. We will use this below to estimate $\mu(t)$. Our earlier estimation approach, in

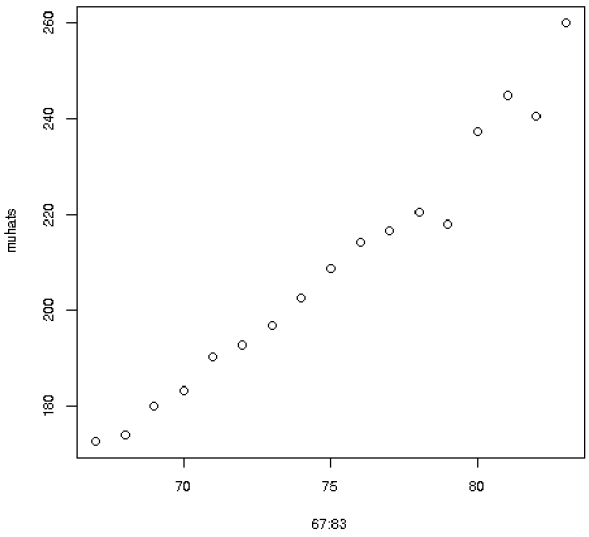


Figure 1.1: Plotted $\widehat{\mu}(t)$

Section 1.6.2, is called *nonparametric*. It is also called *assumption-free*, since it made no assumption at all about the shape of the $\mu(t)$ curve.

Note the following carefully:

- Figure 1.1 suggests that our straight-line model for $\widehat{\mu}(t)$ may be less accurate at very small and very large values of $t$. This is hard to say, though, since we have rather few data points in those two regions, as seen in our earlier R calculations; there is only one person of height 83, for instance.

  But again, in this chapter we are simply exploring, so let's assume for now that the straight-line model for $\widehat{\mu}(t)$ is reasonably accurate. We will discuss in Chapter 6 how to assess the validity of this model.

- Since $\mu(t)$ is a population function, the constants $c$ and $d$ are population values, thus unknown. However, we can estimate them from our sample data. We do so using R's **lm()** ("linear model") function:[7]

[7] Details on how the estimation is done will be given in Chapter 2.

```
> lmout <- lm(mlb$Weight ~ mlb$Height)
> lmout
Call:
lm(formula = mlb$Weight ~ mlb$Height)

Coefficients:
(Intercept)    mlb$Height
    -151.133         4.783
```

This gives $\widehat{c} = -151.133$ and $\widehat{d} = 4.783$.

We would then set, for instance (using the caret instead of the hat, so as to distinguish from our previous estimator)

$$\check{\mu}(72) = -151.133 + 4.783 \times 72 = 193.2666 \tag{1.8}$$

We need not type this expression into R by hand. Here is why: Writing the expression in matrix-multiply form, it is

$$(-151.133, 4.783) \begin{pmatrix} 1 \\ 72 \end{pmatrix} \tag{1.9}$$

Be sure to see the need for that 1 in the second factor; it is used to multiply the -151.133. Now let's use that matrix form to show how we can conveniently compute that value in R:[8]

The key is that we can exploit the fact that R's **coef()** function fetches the coefficients $c$ and $d$ for us:

```
> coef(lmout)
(Intercept)    mlb$Height
-151.133291      4.783332
```

Recalling that the matrix-times-matrix operation in R is specified via the %*% operator, we can now obtain our estimated value of $\mu(72)$ as

```
> coef(lmout) %*% c(1,72)
          [,1]
[1,] 193.2666
```

[8] In order to gain a more solid understanding of the concepts, we will refrain from using R's **predict()** function for now. It will be introduced later, though, in Section 4.4.4.

So, using this model, we would predict a slightly heavier weight than our earlier prediction.

We can form a confidence interval from this too, which for the 95% level will be

$$\widehat{\mu}(72) \pm 1.96 \text{ s.e.}[(\widehat{\mu}(72)] \tag{1.10}$$

where *s.e.*[] signifies *standard error*, the estimated standard deviation of an estimator. Here $\widehat{\mu}(72)$, being based on our random sample data is itself random, i.e. it will vary from sample to sample. It thus has a standard deviation, which we call the standard error. We will see later that s.e.$[(\widehat{\mu}(72)]$ is to be obtainable using the R **vcov()** function:

```
> tmp <- c(1,72)
> sqrt(tmp %*% vcov(lmout) %*% tmp)
           [,1]
[1,] 0.6859655
> 193.2666 + 1.96 * 0.6859655
[1] 194.6111
> 193.2666 - 1.96 * 0.6859655
[1] 191.9221
```

(More detail on **vcov()** and **coef()** as R functions is presented in Section 1.20.3 in the Computational Complements section at the end of this chapter.)

So, an approximate 95% CI for $\mu(72)$ under this model would be about (191.9,194.6).

## 1.7 Parametric vs. Nonparametric Models

Now here is a major point: The CI we obtained from our linear model, (191.9,194.6), was narrower than the nonparametric approach gave us, (187.6,193.2); the former has width of about 2.7, while the latter's is 5.6. In other words:

> A parametric model is — if it is (approximately) valid — more powerful than a nonparametric one, yielding estimates of a regression function that tend to be more accurate than what the nonparametric approach gives us. This should translate to more accurate prediction as well.

Why should the linear model be more effective? Here is some intuition, say for estimating $\mu(72)$: As will be seen in Chapter 2, the **lm()** function uses *all* of the data to estimate the regression coefficients. In our case here, all 1015 data points played a role in the computation of $\check{\mu}(72)$, whereas only 150 of our observations were used in calculating our nonparametric estimate $\widehat{\mu}(72)$. The former, being based on much more data, should tend to be more accurate.[9]

On the other hand, in some settings it may be difficult to find a valid parametric model, in which case a nonparametric approach may be much more effective. *This interplay between parametric and nonparametric models will be a recurring theme in this book.*

## 1.8 Example: Click-Through Rate

Let's try a linear regression model on the CTR data in Section 1.3:

```
> ctr <-
   read.table('State_CTR_Date.txt',header=T,sep='\t')
> lm(ctr$CTR ~ ctr$College_Grad)
...
Coefficients:
     (Intercept)   ctr$College_Grad
         0.01412           -0.01373
...
```

We can put this in perspective by considering the standard deviation of **College_Grad**:

```
> sd(ctr$College_Grad)
[1] 0.04749804
```

So, a "typical" difference between one state and another is something like 0.05. Multiplying by the -0.01373 figure above, this translates to a difference in click-through rate of about 0.0005. This is certainly not enough to have any practical meaning.

So, putting aside such issues as whether our data constitute a sample from some "population" of potential states, the data suggest that there is really no substantial relation between educational level and CTR.

[9] Note the phrase *tend to* here. As you know, in statistics one usually cannot say that one estimator is always better than another, because anomalous samples do have some nonzero probability of occurring.

## 1.9 Several Predictor Variables

Now let's predict weight from height and age. We first need some notation.

Say we are predicting a response variable $Y$ from variables $X^{(1)}, ..., X^{(k)}$. The regression function is now defined to be

$$\mu(t_1, ..., t_k) = E(Y \mid X^{(1)} = t_1, ..., X^{(k)} = t_k) \tag{1.11}$$

In other words, $\mu(t_1, ..., t_k)$ is the mean $Y$ among all units (people, cars, whatever) in the population for which $X^{(1)} = t_1, ..., X^{(k)} = t_k$.

In our baseball data, $Y$, $X^{(1)}$ and $X^{(2)}$ might be weight, height and age, respectively. Then $\mu(72, 25)$ would be the population mean weight among all players of height 72 and age 25.

We will often use a vector notation

$$\mu(t) = E(Y \mid X = t) \tag{1.12}$$

with $t = (t_1, ..., t_k)'$ and $X = (X^{(1)}, ..., X^{(k)})'$, where $'$ denotes matrix transpose.[10]

### 1.9.1 Multipredictor Linear Models

Let's consider a parametric model for the baseball data,

$$\text{mean weight} = c + d \times \text{ height} + e \times \text{ age} \tag{1.14}$$

#### 1.9.1.1 Estimation of Coefficients

We can again use **lm()** to obtain sample estimates of $c$, $d$ and $e$:

[10] Our vectors in this book are column vectors. However, since they occupy a lot of space on a page, we will often show them as transposes of rows. For instance, we will often write $(5, 12, 13)'$ instead of

$$\begin{pmatrix} 5 \\ 12 \\ 13 \end{pmatrix} \tag{1.13}$$

```
> lm(mlb$Weight ~ mlb$Height + mlb$Age)
...
Coefficients:
(Intercept)    mlb$Height        mlb$Age
   -187.6382        4.9236         0.9115
```

Note that the notation mlb$Weight ~mlb$Height + mlb$Age simply means "predict weight from height and age." The variable to be predicted is specified to the left of the tilde, and the predictor variables are written to the right of it. The + does not mean addition.

A shorter formulation is

```
> lm(Weight ~ Height + Age,data=mlb)
```

and, shorter still,

```
> lm(Weight ~ .,data=mlb[,4:6])
```

where the period means "all the other variables."

So, the output shows us the estimated coefficientsis, e.g. $\widehat{d} = 4.9236$. So our estimated regression function is

$$\widehat{\mu}(t_1, t_2) = -187.6382 + 4.9236\ t_1 + 0.9115\ t_2 \tag{1.15}$$

where $t_1$ and $t_2$ are height and age, respectively.

Setting $t_1 = 72$ and $t_2 = 25$, we find that

$$\widehat{\mu}(72, 25) = 189.6485 \tag{1.16}$$

and we would predict the weight of a 72-inch tall, age 25 player to be about 190 pounds.

#### 1.9.1.2 The Description Goal

It was mentioned in Section 1.1 that regression analysis generally has one or both of two goals, Prediction and Description. In light of the latter, some brief comments on the magnitudes of the estimated coefficientsis would be useful at this point:

- We estimate that, on average (a key qualifier), each extra inch in height corresponds to almost 5 pounds of additional weight.
- We estimate that, on average, each extra year of age corresponds to almost a pound in extra weight.

That second item is an example of the Description goal in regression analysis, We may be interested in whether baseball players gain weight as they age, like "normal" people do. Athletes generally make great efforts to stay fit, but we may ask how well they succeed in this. The data here seem to indicate that baseball players indeed are prone to some degree of "weight creep" over time.

### 1.9.2 Nonparametric Regression Estimation: k-NN

Now let's drop the linear model assumption (1.14), and estimate our regression function "from scratch," as we did in Section 1.6.2. But here we will need to broaden our approach, as follows.

#### 1.9.2.1 Looking at Nearby Points

Again say we wish to estimate, using our data, the value of $\mu(72, 25)$. A potential problem is that there likely will not be any data points in our sample that exactly match those numbers, quite unlike the situation in (1.4), where $\widehat{\mu}(72)$ was based on 150 data points. Let's check:

```
> z <- mlb[mlb$Height == 72 & mlb$Age == 25,]
> z
[1] Name        Team        Position
[4] Height      Weight      Age
[7] PosCategory
<0 rows> (or 0-length row.names)
```

So, indeed there were no data points matching the 72 and 25 numbers. Since the ages are recorded to the nearest 0.01 year, this result is not surprising. But at any rate we thus cannot set $\widehat{\mu}(72, 25)$ to be the mean weight among our sample data points satisfying those conditions, as we did in Section 1.6.2. And even if we had had a few data points of that nature, that would not have been enough to obtain an accurate estimate $\widehat{\mu}(72, 25)$.

Instead, what is done is use data points that are *close* to the desired prediction point. Again taking the weight/height/age case as a first example,

this means that we would estimate $\mu(72, 25)$ by the average weight in our sample data among those data points for which height is *near* 72 and age is *near* 25.

### 1.9.3 Measures of Nearness

Nearness is generally defined as *Euclidean distance*:

$$\text{distance}[(s_1, s_2, ..., s_k), (t_1, t_2, ..., t_k)] = \sqrt{((s_1 - t_1)^2 + ... + (s_k - t_k)^2} \tag{1.17}$$

For instance, the distance from a player in our sample of height 72.5 and age 24.2 to the point (72,25) would be

$$\sqrt{(72.5 - 72)^2 + (24.2 - 25)^2} = 0.9434 \tag{1.18}$$

Note that the Euclidean distance between $s = (s_1, ..., s_k$ and $t = (t_1, ..., t_k$ is simply the Euclidean norm of the difference $s - t$ (Section A.1(.

#### 1.9.3.1 The k-NN Method

The *k-Nearest Neighbor* (k-NN) method for estimating regression functions is simple: Find the $k$ data points in our sample that are closest to the desired prediction point, and average their $Y$ values.

A question arises as to how to choose the value of $k$. Too large a value means we are including "nonrepresentative" data points, but too small a value gives too us few points to average for a good estimate. We will return to this question later, but will note that due to this nature of $k$, will call $k$ a *tuning* parameter. Various tuning parameters will come up in this book.

### 1.9.4 The Code

Here is code to perform k-NN regression estimation:

```
knnest <- function(xydata,regestpts,k) {
   require(FNN)
   ycol <- ncol(xydata)
   x <- xydata[,-ycol,drop = F]
```

```
    y <- xydata[,ycol]
    if (is.vector(regestpts)) {
       regestpts <- matrix(regestpts,nrow=1)
       colnames(regestpts) <- colnames(x)
    }
    tmp <- rbind(x,regestpts)
    tmp <- scale(tmp)
    x <- tmp[1:nrow(x),]
    regestpts <- tmp[(nrow(x)+1):nrow(tmp),]
    if (!is.matrix(regestpts))
       regestpts <- matrix(regestpts,nrow=1)
    tmp <- get.knnx(data=x, query=regestpts, k=k)
    idx <- tmp$nn.index
    meannear <- function(idxrow) mean(y[idxrow])
    apply(idx,1,meannear)
}
```

Each row of **regestpts** is a point at which we wish to estimate the regression function. For example, let's estimate $\mu(72, 25)$, based on the 20 nearest neighbors at each point:

```
> knnest(mlb[,c(4,6,5)],c(72,25),20,scalefirst=TRUE)
[1] 188.9
```

So we would predict the weight of a 72-inches tall, age 25 player to be about 189 pounds, not much different — in this instance — from what we obtained earlier with the linear model.

The call to the built-in R function **scale()** is useful if our predictor variables are of widely different magnitudes. In such a setting, the larger-magnitude variables are in essence being given heavier weightings in the distance computation. However, rerunning the above analysis without scaling (not shown) produces the same result.

## 1.10 After Fitting a Model, How Do We Use It for Prediction?

As noted, our goal in regression analysis could be either Prediction or Description (or both). How specifically does the former case work?

### 1.10.1 Parametric Settings

The parametric case is the simpler one. We fit our data, write down the result, and then use that result in the future whenever we are called upon to do a prediction.

Recall Section 1.9.1.1. It was mentioned there that in that setting, we probably are not interested in the Prediction goal, but just as an illustration, suppose we do wish to predict. We fit our model to our data — called our *training data* — resulting in our estimated regression function, (1.15). From now on, whenever we need to predict a player's weight, given his height and age, we simply plug those values into (1.15).

### 1.10.2 Nonparametric Settings

The nonparametric case is a little more involved, because we have no explicit equation like (1.15). Nevertheless, we use our training data in the same way. For instance, say we need to predict the weight of a player whose height and age are 73.2 and 26.5, respectively. Our predicted value will be then $\widehat{\mu}(73.2, 26.5)$. To obtain that, we go back to our training data, find the $k$ nearest points to (73.2,26.5), and average the weights of those $k$ players. We would go through this process each time we are called upon to perform a prediction.

**A variation:**

A slightly different approach, which we will use here, is as follows. Denote our training set data as $(X_1, Y_1), ..., (X_n, Y_n)$, where again the $X_i$ are typically vectors, e.g. (height,age). We estimate our regression function at each of the points $X_i$, forming $\widehat{\mu}(X_i), i = 1, ..., n$. Then, when faced with a new case $(X, Y)$ for which $Y$ is unknown, we find the *single* closest $X_i$ to $X$, and guess $Y$ to be 1 or 0, depending on whether $\widehat{\mu}(X_i) > 0.5$.

## 1.11 Underfitting, Overfitting, Bias and Variance

One major concern in model development is *overfitting*, meaning to fit such an elaborate model that it "captures the noise rather than the signal." This description is often heard these days, but it is vague and potentially misleading. We will discuss it in detail in Chapter 9, but it is of such

importance that we should introduce it here in this prologue chapter.

The point is that, after fitting our model, we are concerned that it may fit our training data well but not predict well on new data in the future.[11] Let's look into this further:

### 1.11.1 Intuition

To see how overfitting may occur, consider the famous *bias-variance trade-off*, illustrated in the following example. Again, keep in mind that the treatment will at this point just be intuitive, not mathematical.

Long ago, when I was just finishing my doctoral study, I had my first experience in statistical consulting. A chain of hospitals was interested in comparing the levels of quality of care given to heart attack patients at its various locations. A problem was noticed by the chain regarding straight comparison of raw survival rates: One of the locations served a largely elderly population, and since this demographic presumably has more difficulty surviving a heart attack, this particular hospital may misleadingly appear to be giving inferior care.

An analyst who may not realize the age issue here would thus be biasing the results. The term "bias" here doesn't mean deliberate distortion of the analysis, just that one is using a less accurate model than one should, actually "skewed" in the common vernacular. And it is permanent bias, in the sense that it won't disappear, no matter how large a sample we take.

Such a situation, in which an important variable is not included in the analysis, is said to be *underfitted*. By adding more predictor variables in a regression model, in this case age, we are reducing bias.

Or, suppose we use a regression model which is linear in our predictors, but the true regression function is nonlinear. This is bias too, and again it won't go away even if we make the sample size huge.

On the other hand, we must keep in mind that our data is a sample from a population. In the hospital example, for instance, the patients on which we have data can be considered a sample from the (somewhat conceptual) population of all patients at this hospital, past, present and future. A different sample would produce different regression coefficient estimates. In other words, there is variability in those coefficients from one sample to another, i.e. variance. We hope that that variance is small, which gives us

[11] Note that this assumes that nothing changes in the system under study between the time we collect our training data and the time we do future predictions.

confidence that the sample we have is representative.

But the more predictor variables we have, the more collective variability there is in the inputs to our regression calculations, and thus the larger the variances of the estimated coefficients.[12] If those variances are large enough, the benefit of using a lot of predictors may be overwhelmed by the increased variability of the results. This is called *overfitting*.

In other words:

> In deciding how many (and which) predictors to use, we have a tradeoff. The richer our model, the less bias, but the more variance.

In Section 1.19.2 it is shown that for any statistical estimator $\widehat{\theta}$ (that has finite variance),

> mean squred error = squared bias + variance

Our estimator here is $\widehat{\mu}(t)$. This shows the tradeoff: Adding variables, such as age in the hospital example, reduces squared bias but increases variance. Or, equivalently, removing variables reduces variance but exacerbates bias. It may for example be beneficial to accept a little bias in exchange for a sizable reduction in variance, which we may achieve by removing some predictors from our model..

The trick is somehow to find a "happy medium," easier said than done. Chapter 9 will cover this in depth, but for now, we introduce a common method for approaching the problem:

### 1.11.2 Cross-Validation

Toward that end, it is common to artificially create a set of "new" data and try things out. Instead of using all of our collected data as our training set, we set aside part of it to serve as simulated "new" data. This is called the *validation set* or *test set*. The remainder will be our actual training data. In other words, we randomly partition our original data, taking one part as our training set and the other part to play the role of new data. We fit our model, or models, to the training set, then do prediction on the test set, pretending its response variable values are unknown. We then compare to

[12] I wish to thank Ariel Shin for this interpretation.

the real values. This will give us an idea of how well our models will predict in the future. This is called *cross-validation*.

The above description is a little vague, and since there is nothing like code to clarify the meaning of an algorithm, let's develop some. Here first is code to do the random partitioning of **data**, with a proportion **p** to go to the training set:

```
xvalpart <- function(data,p) {
   n <- nrow(data)
   ntrain <- round(p*n)
   trainidxs <- sample(1:n,ntrain,replace=FALSE)
   valididxs <- setdiff(1:n,trainidxs)
   list(train=data[trainidxs,], valid=data[valididxs,])
}
```

R's **setdiff()** function does set differencing, e.g.

```
> a <- c(5,2,8,12)
> b <- c(8,5,88)
# everything in a but not b
> setdiff(a,b)
[1]  2 12
```

This sets up the partitioning of the data into training and test sets.

Now to perform cross-validation, we'll consider the parametric and non-parametric cases separately, in the next two sections.

### 1.11.3 Linear Model Case

To do cross-validation for linear models, we could use this code.[13]

#### 1.11.3.1 The Code

```
# arguments:
#
#    data:  full data
#    ycol:  column number of resp. var.
#    predvars:  column numbers of predictors
```

[13]There are sophisticated packages on CRAN for this, such as **cvTools**. But to keep things simple, and to better understand the concepts, we will write our own code. Similarly, as mentioned, we will not use R's **predict()** function for the time being.

```
#    p:  prop. for training set
#    meanabs:  see 'value' below

# value:  if meanabs is TRUE, the mean absolute
#         prediction error; otherwise, an R list
#         containing pred., real Y

xvallm <- function(data,ycol,predvars,p,meanabs=TRUE){
   tmp <- xvalpart(data,p)
   train <- tmp$train
   valid <- tmp$valid
   # fit model to training data
   trainy <- train[,ycol]
   trainpreds <- train[,predvars]
   # we'll be using matrices, e.g. in lm()
   trainpreds <- as.matrix(trainpreds)
   lmout <- lm(trainy ~ trainpreds)
   # apply fitted model to validation data
   validpreds <- as.matrix(valid[,predvars])
   predy <- cbind(1,validpreds)%*% coef(lmout)
   realy <- valid[,ycol]
   if (meanabs) return(mean(abs(predy - realy)))
   list(predy = predy, realy = realy)
}
```

#### 1.11.3.2 Matrix Partitioning

Note that in the line

```
predy <- cbind(1,validpreds)%*% coef(lmout)
```

we have exploited the same matrix multiplication property as in (1.9). Here, though, we have applied it at the matrix level. Such operations will become common in some parts of this book, so a brief digression will be worthwhile. For a concrete numerical example, consider the vector

$$\begin{pmatrix} (-1,2)(3,8)' \\ (2,5)(3,8)' \end{pmatrix} = \begin{pmatrix} 13 \\ 46 \end{pmatrix} \tag{1.19}$$

The reader should verify that "distributing out" that common $(3,8)'$ factor is valid algebra:

$$\begin{pmatrix} -1 & 2 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 3 \\ 8 \end{pmatrix} = \begin{pmatrix} 13 \\ 46 \end{pmatrix} \tag{1.20}$$

#### 1.11.3.3 Applying the Code

Let's try cross-validtion on the weight/height/age data, using mean absolute prediction error as our criterion for prediction accuracy:

```
> xvallm(mlb,5,c(4,6),2/3)
[1] 12.94553
```

So, on average we would be off by about 13 pounds. We might improve upon this by using the data's Position variable, but we'll leave that for later.

### 1.11.4 k-NN Case

Here is the code for performing cross-validation for k-NN:

```
# arguments:
#
#    data:  full data
#    ycol:  column number of resp. var.
#    predvars:  column numbers of predictors
#    k:  number of nearest neighbors
#    p:  prop. for training set
#    meanabs:  see 'value' below

# value:  if meanabs is TRUE, the mean absolute
#         prediction error; otherwise, an R list
#         containing pred., real Y

xvalknn <-
      function(data,ycol,predvars,k,p,meanabs=TRUE){
   tmp <- xvalpart(data,p)
   train <- tmp$train
   valid <- tmp$valid
   trainxy <- data[,c(predvars,ycol)]
   validx <- valid[,predvars]
```

```
    validx <- as.matrix(validx)
    predy <- knnest(trainxy,validx,k)
    realy <- valid[,ycol]
    if (meanabs) return(mean(abs(predy - realy)))
    list(predy = predy, realy = realy)
}
```

So, how well does k-NN predict?

```
> xvallm(mlb,5,c(4,6),2/3)
[1] 12.94553
```

The two methods gave similar results. However, this depended on choosing a value of 20 for **k**, the number of nearest neighbors. We could have tried other values of **k**, and in fact could have used cross-validation to choose the "best" value.

### 1.11.5 Choosing the Partition Sizes

One other problem, of course, is that we did have a random partition of our data. A different one might have given substantially different results.

In addition, there is the matter of choosing the sizes of the training and validation sets (e.g. via the argument **p** in **xvalpart()**). We have a classical tradeoff at work here: Let $k$ be the size of our training set. If we make $k$ too large, the validation set will be too small for an accurate measure of prediction accuracy. We won't have that problem if we set $k$ to a smaller size, but then we are masuring the predictive ability of only $k$ observations, whereas in the end we will be using all $n$ observations for predicting new data.

The *Leaving One-Out Method* solves this problem, albeit at the expense of much more computation. It will be presented in Section 2.9.5.

## 1.12 Rough Rule of Thumb

The issue of how many predictors to use to simultaneously avoid overfitting and still produce a good model is nuanced, and in fact this is still not fully resolved. Chapter 9 will be devoted to this complex matter.

Until then, though it is worth using the following:

**Rough Rule of Thumb (Tukey):** For a data set consisting of n observations, use fewer than $\sqrt{(n)}$ predictors.

## 1.13 Example: Bike-Sharing Data

We now return to the bike-sharing data. Our little excursion to the simpler data set, involving baseball player weights and heights, helped introduce the concepts in a less complex setting. The bike-sharing data set is more complicated in several ways:

- **Complication (a):** It has more potential predictor variables.
- **Complication (b):** It includes some *nominal* (or *categorical*) variables, such as Day of Week. The latter is technically numeric, 0 through 6, but those codes are just names. Hence the term *nominal*. In R, by the way, the formal term for such variables is *factors*.

  The problem is that there is no reason, for instance, tha Sunday, Thursday and Friday should have an ordinal relation in terms of ridership just because $0 < 4 < 5$.
- **Complication (c):** It has some potentially nonlinear relations. For instance, people don't like to ride bikes in freezing weather, but they are not keen on riding on really hot days either. Thus we might suspect that the relation of ridership to temperature rises at first, eventually reaching a peak, but declines somewhat as the temperature increases further.

Now that we know some of the basic issues from analyzing the baseball data, we can treat this more complicated data set.

Let's read in the bike-sharing data. We'll restrict attention to the first year,[14] and since we will focus on the registered riders, let's shorten the name for convenience:

```
> shar <- read.csv("day.csv",header=T)
> shar <- shar[1:365,]
> names(shar)[15] <- "reg"
```

[14] There appears to have been some systemic change in the second year, and while this could be modeled, we'll keep things simple by considering only the first year.

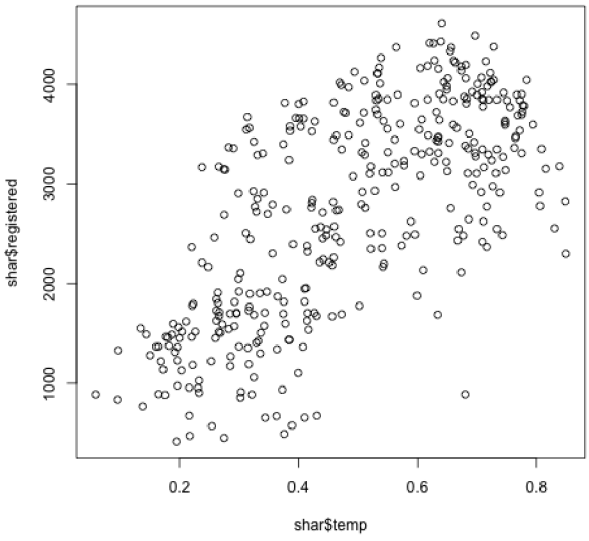


Figure 1.2:
Ridership vs. Temperature

### 1.13.1 Linear Modeling of $\mu(t)$

In view of Complication (c) above, the inclusion of the word *linear* in the title of our current section might seem contradictory. But one must look carefully at *what* is linear or not, and we will see shortly that, yes, we can use linear models to analyze nonlinear relations.

Let's first check whether the ridership/temperature relation seems nonlinear, as we have speculated:

```
plot(shar$temp,shar$reg)
```

The result is shown in Figure 1.2.

There seem to be some interesting groupings among the data, likely due to the other variables, but putting those aside for now, the plot does seem to suggest that ridership is slightly associated with temperature in the "first up, then later down" form as we had guessed.

Thus a linear model of the form

$$\text{mean ridership} = c + d \times \text{ temperature} \tag{1.21}$$

would seem inappropriate. But don't give up so quickly! A model like

$$\text{mean ridership} = c + d \times \text{ temperature} + e \times \text{ temperature}^2 \tag{1.22}$$

i.e., with a temperature-squared term added, might work fine. A negative value for $e$ would give us the "first up, then later down" behavior we want our model to have.

And there is good news — the model (1.22) is actually linear! We say that the expression is *linear in the parameters*, even though it is nonlinear with respect to the temperature variable. This means that if we multiply each of $c$, $d$ and $e$ by, say, 8, then the values of the left and right sides of the equation both increase eightfold.

Anotber way to see this is that in calling **lm()**, we can simply regard squared temperature as a new variable:

```
> shar$temp2 <- shar$temp^2
> lm(shar$reg ~ shar$temp + shar$temp2)

Call:
lm(formula = shar$reg ~ shar$temp + shar$temp2)

Coefficients:
(Intercept)     shar$temp    shar$temp2
      -1058         16133        -11756
```

And note that, sure enough, the coefficient of the squared term, $\widehat{e} = -11756$, did indeed turn out to be negative.

Of course, we want to predict from many variables, not just temperature, so let's now turn to Complication (b) cited earlier, the presence of nominal data. This is not much of a problem either.

Such situations are generally handled by setting up what are called *indicator variables* or *dummy variables*. The former term alludes to the fact that our variable will *indicate* whether a certain condition holds or not, with 1 coding the yes case and 0 indicating no.

We could, for instance, set up such a variable for Tuesday data:

```
> shar$tues <- as.integer(shar$weekday == 2)
```

Indeed, we could define six variables like this, one for each of the days Monday through Saturday. Note that Sunday would then be indicated indirectly, via the other variables all having the value 0. A direct Sunday variable would be redundant, and in fact would present mathematical problems, as we'll see in Chapter 8. (Actually, R's **lm()** function can deal with factor variables directly, as shown in Section 9.7.5.1. But we take the more basic route here, in order to make sure the underlying principles are clear.)

However, let's opt for a simpler analysis, in which we distinguish only between weekend days and week days, i.e. define a dummy variable that is 1 for Monday through Friday, and 0 for the other days. Actually, those who assembled the data set already defined such a variable, which they named **workingday**.[15]

We incorporate this into our linear model:

$$\text{mean reg} = c + d \times \text{ temp} + e \times \text{ temp}^2 + f \text{ workingday} \tag{1.23}$$

There are several other dummy variables that we could add to our model, but for this introductory example let's define just one more:

```
> shar$clearday <- as.integer(shar$weathersit == 1)
```

So, our regression model will be

$$\begin{aligned} \text{mean reg} &= \beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\ &+ \beta_3 \text{ workingday} + \beta_4 \text{ clearday} \end{aligned} \tag{1.24}$$

As is traditional, here we have used subscripted versions of the Greek letter $\beta$ to denote our equation coefficients, rather than $c$, $d$ and so on.

So, let's run this through **lm()**:

```
> lmout <- lm(reg ~ temp+temp2+workingday+clearday,
    data = shar[test,])
```

(The use of the **data** argument saved multiple typing of the data set name **shar** and clutter.)

[15] More specifically, a value of 1 for this variable indicates that the day is in the Monday-Friday range *and* it is not a holiday.

The return value of **lm()**, assigned here to **lmout**, is a very complicated R object, of class **"lm"**. We shouldn't inspect it in detail now, but let's at least print the object, which in R's interactive mode can be done simply by typing the name, which automatically calls **print()** on the object:[16]

```
> lmout
...
...
Coefficients:
(Intercept)          temp         temp2
    -2310.3       17342.2      -13434.7
  workingday     clearday
      988.5         760.3
```

Remember, the population function $\mu(t)$ is unnown, so the $\beta_i$ are unknown. The above coefficients are merely sample-based estimates. For example, using our usual "hat" notation to mean "estimate of," we have that

$$\widehat{\beta}_3 = 988.5 \tag{1.25}$$

In other words, estimated regression function is

$$\widehat{\mu}(t_1, t_2, t_3, t_4) = -2310.3 + 17342.2 t_1 - 13434.7 t_2 + 988.5 t_3 + 760.3 t_4 \tag{1.26}$$

where $t_2 = t_1^2$.

So, what should we predict for number of riders on the type of day described at the outset of this chapter — Sunday, sunny, 62 degrees Fahrenheit? First, note that the designers of the data set have scaled the **temp** variable to [0,1], as

$$\frac{\text{Celsius temperature} - \text{minimum}}{\text{maximum} = \text{minimum}} \tag{1.27}$$

where the minimum and maximum here were -8 and 39, respectively. This form may be easier to understand, as it is expressed in terms of where the given temperature fits on the normal range of temperatures. A Fahrenheit temperature of 62 degrees corresponds to a scaled value of 0.525. So, our

[16] See more detail on this in Section 1.20.3.

predicted number of riders is

$$-2310.3 + 17342.2 \times 0.525 - 13434.7 \times 0.525^2 + 988.5 \times 0 + 760.3 \times 1 \quad (1.28)$$

which as before we can conveniently evaluate as

```
> coef(lmout) %*% c(1,0.525,0.525^2,0,1)
          [,1]
[1,] 3851.673
```

So, our predicted number of riders for sunny, 62-degree Sundays will be about 3852.

As noted earlier, one can also form confidence intervals and perform significance tests on the $\beta_i$. We'll go into this in Chapter 2, but some brief comments on the magnitudes and signs of the $\widehat{\beta}_i$ is useful at this point:

- As noted, the estimated coefficient of **temp2** is negative, consistent with our intuition. Note, though, that it is actually more negative than when we predicted **reg** from only temperature and its square. This is typical, and will be discussed in detail in Chapter 7.
- The estimated coefficient for **workingday** is positive. This too matches our intuition, as presumably many of the registered riders use the bikes to commute to work. The value of the estimate here, 988.5, indicates that, for fixed temperature and weather conditions, weekdays tend to have close to 1000 more registered riders than weekends.
- Similarly, the coefficient of **clearday** suggests that for fixed temperature and day of the week, there are about 760 more riders on clear days than on other days.

### 1.13.2 Nonparametric Analysis

Let's see what k-NN gives us as our predicted value for sunny, 62-degree Sundays, say with values of 20 and 50 for **k**:

```
> knnest(shar[,c(10,8,17,15)],matrix(c(0.525,0,1),
    nrow=1),20)
[1] 2881.8
> knnest(shar[,c(10,8,17,15)],matrix(c(0.525,0,1),
    nrow=1),10)
[1] 3049.7
```

This is quite different from what the linear model gave us. Let's see how the two approaches compare in cross-validation:

```
> xvallm(shar,15,c(10,18,8,17),2/3)
[1] 519.8701
> xvalknn(shar,15,c(10,8,17),20,2/3)
[1] 461.2426
> xvalknn(shar,15,c(10,8,17),10,2/3)
[1] 498.3115
```

The nonparametric approach did substantially better, possibly indicating that our linear model was not valid. Of course, there still is the problems of not knowing what value to use for **k**, the fact that our partition was random and so on. These issues will be discussed in detail in succeeding chapters.

## 1.14 Interaction Terms

Let's take another look at (1.24), specifically the term involving the variable **workingday**, a dummy indicating a nonholiday Monday through Friday. Our estimate for $\beta_3$ turned out to be 988.5, meaning that, holding temperature and the other variables fixed, there are 988.5 additional riders on workingdays.

But implicit in this model is that the workingday effect is the same on low-temprerature days as on warmer days. For a broader model that does not make this assumption, we could add an *interaction term*, consisting of a product of **workingday** and **temp**:

$$\begin{aligned} \text{mean reg} &= \beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\ &+ \beta_3 \text{ workingday } + \beta_4 \text{ clearday} \qquad (1.29) \\ &+ \beta_5 \text{ temp } \times \text{ workingday} \qquad (1.30) \end{aligned}$$

How does this model work? Let's illustrate it with a new data set.

### 1.14.1 Example: Salaries of Female Programmers and Engineers

This data is from the 2000 U.S. Census, consisting of 20,090 programmers and engineers in the Silicon Valley area. The data set is included in the **freqparcoord** package on CRAN. Suppose we are working toward a Description goal, specifically the effects of gender on wage income.

As with our bike-sharing data, we'll add a quadratic term, in this case on the age variable, reflecting the fact that many older programmers and engineers encounter trouble finding work after age 35 or so. Let's restrict our analysis to workers having at least a Bachelor's degree, and look at the variables **age**, **age2**, **sex** (coded 1 for male, 2 for female), **wkswrked** (number of weeks worked), **ms**, **phd** and **wageinc**:

```
> library(freqparcoord)
> data(prgeng)
> prgeng$age2 <- prgeng$age^2
> edu <- prgeng$educ
> prgeng$ms <- as.integer(edu == 14)
> prgeng$phd <- as.integer(edu == 16)
> prgeng$fem <- prgeng$sex - 1
> tmp <- prgeng[edu >= 13,]
> pe <- tmp[,c(1,12,9,13,14,15,8)]
> pe <- as.matrix(pe)
```

Our model is

$$\begin{aligned} \text{mean wageinc} &= \beta_0 + \beta_1 \text{ age} + \beta_2 \text{ age}^2 + \beta_3 \text{ wkswrkd} \\ &+ \beta_4 \text{ ms} + \beta_5 \text{ phd} \\ &+ \beta_6 \text{ fem} \end{aligned} \tag{1.31}$$

We find the following:

```
> summary(lm(pe[,7] ~ pe[,-7]))
...
Coefficients:
                  Estimate Std. Error t value
(Intercept)     -87162.556   4716.088 -18.482
pe[, -7]age       4189.304    234.335  17.877
pe[, -7]age2       -43.479      2.668 -16.293
pe[, -7]wkswrkd   1312.234     29.325  44.748
```

```
pe[, -7]ms          9845.719     843.128   11.678
pe[, -7]phd        17403.523    1762.154    9.876
pe[, -7]fem       -11176.740     912.206  -12.252
                  Pr(>|t|)
(Intercept)         <2e-16 ***
pe[, -7]age         <2e-16 ***
pe[, -7]age2        <2e-16 ***
pe[, -7]wkswrkd     <2e-16 ***
pe[, -7]ms          <2e-16 ***
pe[, -7]phd         <2e-16 ***
pe[, -7]fem         <2e-16 ***
...
```

The results are striking in terms of gender: With age, education and so on held constant, women are estimated to have incomes about $11,177 lower than comparable men.

But this analysis implicitly assumes that the female wage deficit is, for instance, uniform across educational levels. To see this, consider (1.31). Being female makes a $\beta_6$ difference, no matter what the values of **ms** and **phd** are. To generalize our model in this regard, let's define two interaction variables:[17]

```
> msfem <- pe[,4] * pe[,6]
> phdfem <- pe[,5] * pe[,6]
> pe <- cbind(pe,msfem,phdfem)
```

Our model is now

$$\begin{aligned} \text{mean wageinc} &= \beta_0 + \beta_1 \text{ age} + \beta_2 \text{ age}^2 + \beta_3 \text{ wkswrkd} \\ &+ \beta_4 \text{ ms} + \beta_5 \text{ phd} \\ &+ \beta_6 \text{ fem} + \beta_7 \text{ msfem} + \beta_8 \text{ phdfem} \end{aligned} \tag{1.32}$$

So, now instead of there being a single number for the "female effect," $\beta_6$, we how have two:

- Female effect for Master's degree holders: $\beta_6 + \beta_7$

[17] Rather than creating the interaction terms "manually" as is done here, one can use R colon operator, which automates the process. This is not done here, so as to ensure that the reader fully understands the meaning of interaction terms. For information on the colon operator, type **?formula** at the R prompt.

- Female effect for PhD degree holders $\beta_6 + \beta_8$

So, let's rerun the regression analysis:

```
> summary(lm(pe[,7] ~ pe[,-7]))
...
Coefficients:
                    Estimate Std. Error t value
(Intercept)       -87499.793   4715.343 -18.556
pe[, -7]age         4183.402    234.244  17.859
pe[, -7]age2         -43.439      2.667 -16.285
pe[, -7]wkswrkd     1312.160     29.313  44.763
pe[, -7]ms         11060.653    965.016  11.462
pe[, -7]phd        19726.664   1907.382  10.342
pe[, -7]fem        -9091.230   1121.816  -8.104
pe[, -7]msfem      -5088.779   1975.841  -2.575
pe[, -7]phdfem    -14831.582   4957.859  -2.992
                  Pr(>|t|)
(Intercept)        < 2e-16 ***
pe[, -7]age        < 2e-16 ***
pe[, -7]age2       < 2e-16 ***
pe[, -7]wkswrkd    < 2e-16 ***
pe[, -7]ms         < 2e-16 ***
pe[, -7]phd        < 2e-16 ***
pe[, -7]fem       5.75e-16 ***
pe[, -7]msfem      0.01002 *
pe[, -7]phdfem     0.00278 **
...
```

The estimated values of the two female effects are -9091.230 -5088.779 = -14180.01, and 9091.230 -14831.582 = -23922.81. A few points jump out here:

- Once one factors in educational level, the gender gap is seen to be even worse than before.
- The gap is worse at the PhD level than the Master's, likely because of the generally higher wages for the latter.

Thus we still have many questions to answer, especially since we haven't consider other types of interactions yet. This story is not over yet, and will be pursued in detail in Chapter 7.

## 1.15 Classification Techniques

Recall the hospital example in Section 1.11.1. There the response variable is nominal, represented by a dummy variable taking the values 1 and 0, depending on whether the patient survives or not. This is referred to as a *classification problem*, because we are trying to predict which class the population unit belongs to — in this case, whether the patient will belong to the survival or nonsurvival class. We could set up dummy variables for each of the hospital branches, and use these to assess whether some were doing a better job than others, while correcting for variations in age distribution from one branch to another. (Thus our goal here is Description rather than directly Prediction itself.)

This will be explained in detail in Chapter 4, but the point is that we are predicting a 1-0 variable. In a marketing context, we might be predicting which customers are more likely to purchase a certain product. In a computer vision context, we may want to predict whether an image contains a certain object. In the future, if we are fortunate enough to develop relevant data, we might even try our hand at predicting earthquakes.

Classification applications are extremely common. And in many cases there are more than two classes, such as in indentifying many different printed characters in computer vision.

In a number of applications, it is desirable to actually convert a problem with a numeric response variable into a classification problem. For instance, there may be some legal or contractural aspect that comes into play when our variable $V$ is above a certain level $\mathbf{c}$, and we are only interested in whether the requirement is satisfied. We could replace $V$ with a new variable

$$Y = \begin{cases} 1, & \text{if } V > c \\ 0, & \text{if } V \leq c \end{cases} \tag{1.33}$$

Classification methods will play a major role in this book.

### 1.15.1 It's a Regression Problem!

Recall that the regression function is the conditional mean:

$$\mu(t) = E(Y \mid X = t) \tag{1.34}$$

(As usual, $X$ and $t$ may be vector-valued.) In the classification case, $Y$ is an indicator variable, which implies that we know its mean is the probability that $Y = 1$ (Section 1.19.1). In other words,

$$\mu(t) = P(Y = 1 \mid X = t) \tag{1.35}$$

The great implication of this is that *the extensive knowledge about regression analysis developed over the years can be applied to the classification problem.*

An intuitive strategy — but, as we'll see, NOT the only appropriate one — would be to guess that $Y = 1$ if the conditional probability of 1 is greater than 0.5, and guess 0 otherwise. In other words,

$$\text{guess for } Y = \begin{cases} 1, & \text{if } \mu(X) > 0.5 \\ 0, & \text{if } \mu(X) \leq 0.5 \end{cases} \tag{1.36}$$

It turns out that this strategy is optimal, in that it minimizes the overall misclassification error rate (see Section 1.19.4 in the Mathematical Complements portion of this chapter). However, it should be noted that this is not the only possible criterion that might be used. We'll return to this issue in Chapter 5.

As before, note that (1.35) is a population quantity. We'll need to estimate it from our sample data.

### 1.15.2 Example: Bike-Sharing Data

Let's take as our example the situation in which ridership is above 3,500 bikes, which we will call HighUsage:

```
> shar$highuse <- as.integer(shar$reg > 3500)
```

We'll try to predict that variable. Let's again use our earlier example, of a Sunday, clear weather, 62 degrees. Should we guess that this will be a High Usage day?

We can use our k-NN approach just as before:

```
> knnest(shar[,c(10,8,18,19)],c(0.525,0,1),20)
[1] 0.1
```

We estimat that there is a 10% chance of that day having HighUsage.

The parametric case is a little more involved. A model like

$$\begin{aligned} \text{probability of HighUsage} &= \beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\ &+ \beta_3 \text{ workingday } + \beta_4 \text{ clearday} \end{aligned} \quad (1.37)$$

could be used, but would not be very satisfying. The left-hand side of (1.37), as a probability, should be in [0,1], but the right-hand side could in principle fall far outside that range.

Instead, the most common model for conditional probability is *logistic regression*:

$$\begin{aligned} \text{probability of HighUsage} &= \ell(\beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\ &+ \beta_3 \text{ workingday } + \beta_4 \text{ clearday}) \end{aligned} \quad (1.38)$$

where $\ell(s)$ is the *logistic function*,

$$\ell(s) = \frac{1}{1+e^{-s}} \quad (1.39)$$

Our model, then is

$$\mu(t_1,t_2,t_3,t_4) = \frac{1}{1+e^{-(\beta_0+\beta_1 t_1+\beta_2 t_2+\beta_3 t_3+\beta_4 t_4)}} \quad (1.40)$$

where $t_1$ is temperature, $t_2$ is the square of temperature, and so on. We wish to estimate $\mu(62, 62^2, 0, 1)$.

Note the form of the curve, shown in Figure 1.3 The appeal of this model is clear at a glance: First, the logistic function produces a value in [0,1], as appropriate for modeling a probability. Second, it is a monotone increasing function in each of the variables in (1.38), just as was the case in (1.24) for predicting our numeric variable, **reg**. Other motivations for using the logistic model will be discussed in Chapter 4.

R provides the **glm()** ("generalized linear model") function for several nonlinear model families, including the logistic,[18], which is designated via **family = binomial**:

[18] Often called "logit," by the way.

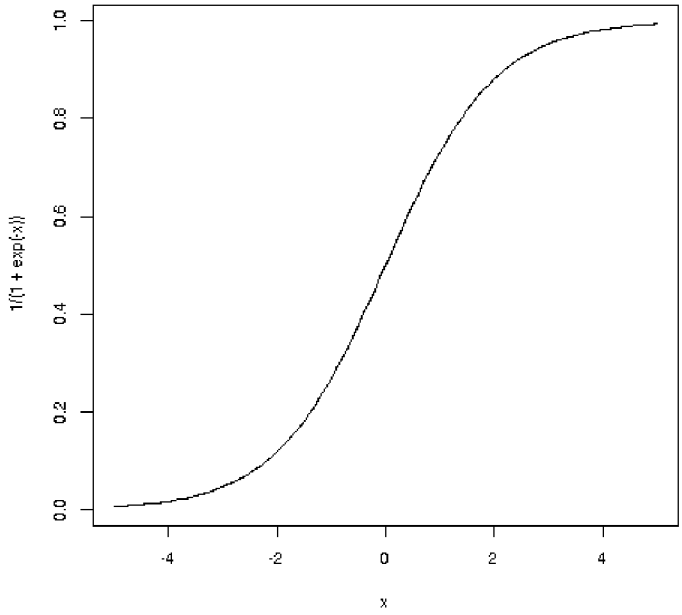


Figure 1.3: Logistic Function

```
> glmout <- glm(highuse ~
      temp+temp2+workingday+clearday,
      data=shar,family=binomial)
> glmout
...
Coefficients:
(Intercept)          temp         temp2
    -18.263        45.909       -36.231
 workingday      clearday
      3.570         1.961
...
> tmp <- coef(glmout) %*% c(1,0.525,0.525^2,0,1)
> 1/(1+exp(-tmp))
            [,1]
[1,] 0.1010449
```

So, our parametric model gives an almost identical result here to the one arising from k-NN, about a 10% probability of HighUsage.

We can perform cross-validation too, and will do so in later chapters. For now, note that our accuracy criterion should change, say to the proportion of misclassified data points.

## 1.16 Crucial Advice: Don't Automate, Participate!

Data science should not be a "spectator sport"; the methodology is effective only if the users *participate*. Avoid ceding the decision making to the computer output. For example:

- Statistical significance does not imply practical importance, and conversely.
- A model is just that — just an approximation to reality, hopefully useful but never exact.
- Don't rely solely on variable selection algorithms to choose your model (Chapter 9).
- "Read directions before use" — make sure you understand what a method really does before employing it.

## 1.17 Informal Use of Prediction

Though statisticians tend to view things through the lens of exact models, proper computation of standard errors and so on, much usage of regression models is much less formal. Former Chinese leader Deng Xiaoping's famous line to convince his peers to accept capitalistt ways comes to mind: "Black cat, white cat, it doesn't matter as long as it catches mice." This more relaxed view is common in the machine learning community, for example.[19]

### 1.17.1 Example: Nonnegative Matrix Factorization

Nonnegative matrix factorization (NMF) is a popular tool in many applications, such as image and text recognition. Here is an overview:

Given an $u \times v$ matrix $A$ with nonnegative elements, we wish to find nonnegative, rank-$k$ matrices[20] $W$ $(u \times k)$ and $H$ $(k \times v)$ such that

$$A \approx WH \tag{1.41}$$

The larger the rank, the better our approximation in (1.41). But we typically hope that a good approximation can be achieved with

$$k \ll \text{ rank}(A) \tag{1.42}$$

The matrices $W$ and $H$ are calculated iteratively, with one of the major methods being regression. Here is how:

We make initial guesses for $W$ and $H$, say with random numbers. Now consider an odd-numbered iteration. Suppose just for a moment that we know the exact value of $W$, with $H$ unknown. Then for each $j$ we could "predict" column $j$ of $A$ from the columns of $W$. The coefficient vector returned by $lm()$ will become column $j$ of $H$. We do this for $j = 1, 2, ..., v$.

In even-numbered iterations, suppose we know $H$ but not $W$. We could take transposes,

$$A' = H'W' \tag{1.43}$$

[19] There of course is the question of whether the cat really is catching mice, a topic that will arise frequently in the coming chapters.

[20] Recall that the rank of a matrix is the maximal number of linearly independent rows or columns.

Figure 1.4: Mt. Rushmore

and then just interchange the roles of $W$ and $H$ above. Here a call to **lm()** gives us a row of $W$, and we do this for all rows.

R's **NMF** package for NMF computation is quite versatile, with many, many options. In its simplest form, though, it is quite easy to use. For a matrix **a** and desired rank **k**, we simply run

```
> nout <- nmf(a,k)
```

The factors are then in **nout@fit@W** and **nout@fit@H**.

Let's illustrate it in an image context, using the image in Figure 1.4.

Though typically NMF is used for image classification, with input data consisting of many images, here we have only one image, and we'll use NMF to compress it, not do classification. We first obtain $A$:

```
> library(pixmap)
> mtr <- read.pnm('Images/MtRush.pgm')
> a <- mtr@grey
```

Now, perform NMF, find the approximation to $A$, and display it, as seen

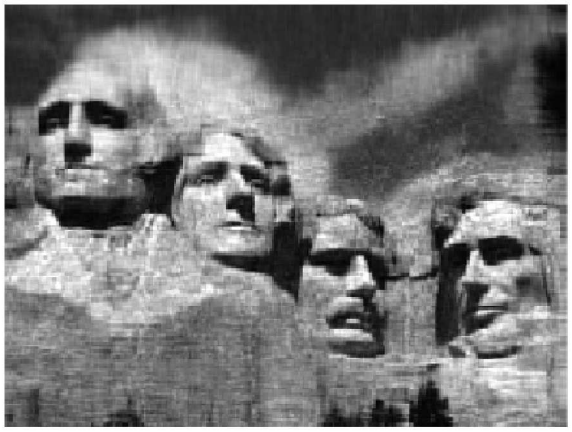

Figure 1.5: Mt. Rushmore, Compressed Image

in Figure 1.5:

```
> aout <- nmf(a,50)
> w <- aout@fit@W
> h <- aout@fit@H
> approxa <- w %*% h
# brightness values must be in [0,1]
> approxa <- pmin(approxa,1)
> mtrnew <- mtr
> mtrnew@grey <- approxa
> plot(mtrnew)
```

This is understandably blurry. The original matrix has dimension $194 \times 259$, and thus presumably has rank 194.[21] We've approximated the matrix by

[21] This is confirmed by running the function **rankMatrix()** in the **Matrix** package.

one of rank only 50, with a 75% storage savings. This is not important for one small p;cture, but possibly worthwhile if we have many large ones. The approximation is not bad in that light, and may be good enough for image recognition or other applications.

Indeed, in many if not most applications of NMF, we need to worry about overfitting. As you will see later, overfitting in this context amounts to using too high a value for our rank, something to be avoided.

## 1.18 Some Properties of Conditional Expectation

This section will be just a wee bit mathematical. Usually this book aims to relegate such material to the Mathematical Complements sections, so that nonmathematical readers can skip such material. But in this section, we ask the reader's brief indulgence, and promise to focus on intuition.

Since the regression function is defined as a conditional expected value, as in (1.3), for mathematical analysis we'll need some properties. First, a definition.

### 1.18.1 Conditional Expectation As a Random Variable

For any random variables $U$ and $V$ with defined expectation, either of which could be vector-valued, define a new random variable $W$, as follows. First note that the conditional expectation of $V$ given $U = t$ is a function of $t$,

$$\mu(t) = E(V \mid U = t) \tag{1.44}$$

This is an ordinary function, just like, say, $\sqrt{t}$ or $\sin(t)$. But we can turn it into a random variable by plugging in a random variable, say $Q$, in for $t$: $R = \sqrt{Q}$ is a random variable. Thinking along these lines, we define the *random variable* version of conditional expectation accordingly:

$$W = E(V|U) = \mu(U) \tag{1.45}$$

As a simple example, say we choose a number $U$ at random from the numbers 1 through 5. We then randomly choose a second number $V$, from the

numbers 1 through $U$. Then

$$\mu(t) = E(V \mid U = t) = \frac{1+t}{2} \tag{1.46}$$

We now form a new random variable $W = (1 + U)/2$.

And, since $W$ is a random variable, we can talk of *its* expected value, which turns out to be an elegant result:

### 1.18.2 The Law of Total Expectation

A property of conditional expected value, proven in many undergraduate probability texts, is

$$E(V) = EW = E[E(V \mid U)] \tag{1.47}$$

The foreboding appearance of this equation belies the fact that it is actually quite intuitive, as follows. Say you want to compute the mean height of all people in the U.S., and you already have available the mean heights in each of the 50 states. You cannot simply take the straight average of those state mean heights, because you need to give more weight to the more populous states. In other words, the national mean height is a *weighted* average of the state means, with the weight for each state being is its proportion of the national population.

In (1.47), this corresponds to having $V$ as height and $U$ as state. State is an integer-valued random variable, ranging from 1 to 50, so we have

$$\begin{aligned} EV &= E[E(V \mid U)] && (1.48)\\ &= EW && (1.49)\\ &= \sum_{i=1}^{50} P(U = i)\ E(V \mid U = i) && (1.50) \end{aligned}$$

The left-hand side, $EV$, is the overall mean height in the nation; $E(V \mid U = i)$ is the mean height in state $i$; and the weights in the weighted average are the proportions of the national population in each state, $P(U = i)$.

Not only can we look at the mean of $W$, but also its variance. By using the

various familiar properties of mean and variance, one can derive a similar relation for variance:

### 1.18.3 Law of Total Variance

For scalar $V$,

$$Var(V) = E[Var(V|U)] + Var[E(V|U)] \tag{1.51}$$

One might initially guess that we only need the first term. To obtain the national variance in height, we would take the weighted average of the state variances. But this would not take into account that the mean heights vary from state to state, thus also contributing to the national variance in height, hence the second term.

This is proven in Section 2.12.10.3.

### 1.18.4 Tower Property

Now consider conditioning on two variables, say $U_1$ and $U_2$. One can show that

$$E\left[E(V|U_1,U_2) \mid U_1\right] = E(V \mid U_1) \tag{1.52}$$

In the height example above, take $U_1$ and $U_2$ to be height and gender, respectively, so that $E(V|U_1,U_2)$ is the mean height of all people in a certain state and of a certain gender. If we take the mean of all these values for a certain state — i.e. take the average of the two gender-specific means in the state — we get the mean height in the state without regard to gender.

Again, note that we take the straight average of the two gender-specific means, because the two genders have equal proportions. If, say, $U_2$ were race instead of gender, we would need to compute a *weighted* average of the race-specific means, with the weights being the proportions of the various races in the given state.

This is proven in Section 7.8.1.

### 1.18.5 Geometric View

There is an elegant way to view all of this in terms of abstract vector spaces — (1.47) becomes the Pythagorean Theorem! — which we will address later in Mathematical Complements Sections 2.12.10 and 7.8.1.

## 1.19 Mathematical Complements

### 1.19.1 Indicator Random Variables

A random variable $W$ is an indicator variable, if it is equal to 1 or 0, depending on whether a certain event $Q$ occurs or not. Two simple properties are very useful:

- $EW = P(Q)$
  This follows from

$$EW = 1 \cdot P(Q) + 0 \cdot P(\text{not } Q) = P(Q) \tag{1.53}$$

- $Var(W) = P(Q) \cdot [1 - P(Q)]$
  True because

$$Var(W) = E(W^2) - (EW)^2 = E(W) - E(W^2) = EW(1 - EW) \tag{1.54}$$

  where the second equality stems from $W^2 = W$ (remember, $W$ is either 1 or 0). Then use the first bullet above!

### 1.19.2 Mean Squared Error of an Estimator

Say we are estimating some unknown population value $\theta$, using an estimator $\widehat{\theta}$ based on our sample data. Then a natural measure of the accuracy of our estimator is the *Mean Squared Error* (MSE),

$$E[(\widehat{\theta} - \theta)^2] \tag{1.55}$$

This is the squared distance from our estimator to the true value, averaged over all possible samples.

Let's rewrite the quantity on which we are taking the expected value:

$$\left(\widehat{\theta} - \theta\right)^2 = \left(\widehat{\theta} - E\widehat{\theta} + E\widehat{\theta} - \theta\right)^2 = (\widehat{\theta} - E\widehat{\theta})^2 + (E\widehat{\theta} - \theta)^2 + 2(\widehat{\theta} - E\widehat{\theta})(E\widehat{\theta} - \theta) \tag{1.56}$$

Look at the three terms on the far right of (1.56). The expected of the first is $Var(\widehat{\theta})$, by definition of variance.

As to the second term, $\widehat{E\theta - \theta}$ is the *bias* of $\widehat{\theta}$, the tendency of $\widehat{\theta}$ to over- or underestimate $\theta$ over all possible samples.

What about the third term? Note first that $E\widehat{\theta} - \theta$ is a constant, thus factoring out of the expectation. But for what remains,

$$E(\widehat{\theta} - E\widehat{\theta}) = 0 \tag{1.57}$$

Taking the expected value of both sides of (1.56), taking the above remarks into account, we have

$$\begin{aligned} \text{MSE}(\widehat{\theta}) &= Var(\widehat{\theta}) + (E\widehat{\theta} - \theta)^2 && (1.58) \\ &= \text{variance} + \text{bias}^2 && (1.59) \end{aligned}$$

In other words:

> The MSE of $\widehat{\theta}$ is equal to the variance of $\widehat{\theta}$ plus squared bias of $\widehat{\theta}$.

### 1.19.3 $\mu$(t) Minimizes Mean Squared Prediction Error

**Claim:** *Consider all the functions f() with which we might predict Y from X, i.e., $\widehat{Y} = f(X)$. The one that minimizes mean squared prediction error, $E[(Y - f(X))^2]$, is the regression function, $\mu(t) = E(Y \mid X = t)$.*

(Note that the above involves population quantities, not samples. Consider the quantity $E[(Y - f(X))^2]$, for instance. It is the mean squared prediction error (MSPE) over all $(X, Y)$ pairs in the population.)

To derive this, first ask, for any (finite-variance) random variable $W$, what number $c$ that minimizes the quantity $E[(W - c)^2]$? The answer is $c = EW$.

To see this, write

$$E[(W-c)^2] = E(W^2 - 2cW + c^2] = E(W^2) - 2cEW + c^2 \tag{1.60}$$

Setting to 0 the derivative of the right-hand side with respect to $c$, we find that indeed, $c = EW$.

Now use the Law of Total Expectation (Section 1.47):

$$\text{MSPE} = E[(Y - f(X))^2] = E\left[E((Y - f(X))^2|X)\right] \tag{1.61}$$

In the inner expectation, $X$ is a constant, and from the statement following (1.60) we know that the minimizing value of $f(X)$ is "EW," in this case $E(Y|X)$, i.e. $\mu(X)$. Since that minimizes the inner expectation for any $\mathbf{X}$, the overall expectation is minimized too.

### 1.19.4 $\mu(t)$ Minimizes the Misclassification Rate

This result concerns the classification context. It shows that if we know the population distribution — we don't, but are going through this exercise to guide our intuition — the conditional mean provides the optimal action in the classification context.

Remember, in this context, $\mu(t) = P(Y \mid X = t)$, i.e. the conditional mean reduces to the conditional probability. Now plug in $X$ for $t$, and we have the following.

**Claim:** *Consider all rules based on $X$ that produce a guess $\widehat{Y}$, taking on values 0 and 1. The one that minimizes the overall misclassification rate $P(\widehat{Y} \neq Y)$ is*

$$\widehat{Y} = \begin{cases} 1, & \text{if } \mu(X) > 0.5 \\ 0, & \text{if } \mu(X) \leq 0.5 \end{cases} \tag{1.62}$$

The claim is completely intuitive, almost trivial: After observing $X$, how should we guess $Y$? If conditionally $Y$ has a greater than 50% chance of being 1, then guess it to be 1!

(Note: In some settings, a "false positive" may be worse than a "false negative," or *vice versa*. The reader should ponder how to modify the material here for such a situation. We'll return to this issue in Chapter 5.)

Think of this simple situation: There is a biased coin, with known probability of heads p. The coin will be tossed once, and we are supposed to guess the outcome.

Let's name your guess $g$ (a nonrandom constant), and let C denote the as-yet-unknown outcome of the toss (1 for heads, 0 for tails). Then the reader should check that, no matter whether we choose 0 or 1 for $g$, the probability that we guess correctly is

$$\begin{aligned} P(C = g) &= P(C = 1)g + P(C = 0)(1 - g) && (1.63) \\ &= pg + (1 - p)(1 - g) && (1.64) \\ &= [2p - 1]g + 1 - p && (1.65) \end{aligned}$$

Now remember, $p$ is known. How should we choose $g$, 0 or 1, in order to maximize (1.65), the probability that our guess is correct? Inspecting (1.65) shows that maximizing that expression will depend on whether $2p-1$ is positive or negative, i.e., whether $p > 0.5$ or not. In the former case we should choose $g = 1$, while in the latter case $g$ should be chosen to be 0.

The above reasoning gives us the very intuitive — actually trivial, when expressed in English — result:

> If the coin is biased toward heads, we should guess heads. If the coin is biased toward tails, we should guess tails.

Now to show the original claim, we use *iterated expectation*, a term alluding to the following. Now returning to our original claim, write

$$P(\widehat{Y} = Y) = E\left[P(\widehat{Y} = Y \mid X)\right] \tag{1.66}$$

In that inner probability, "p" is

$$P(Y = 1 \mid X) = \mu(X) \tag{1.67}$$

which completes the proof.

## 1.20 Computational Complements

### 1.20.1 CRAN Packages

There are thousands of useful contributed R packages available on CRAN, the Comprehensive R Archive Network, `https://cran.r-project.org`. The easiest way to install them is from R's interactive mode, e.g.

```
> install.packages('freqparcoord','~/R')
```

¿

Here I have instructed R to download the **freqparcoord** package, installing it in **~/R**, the directory where I like to store my packages.

Official R parlance is *package*, not *library*, even though ironically one loads a package using the **library()** function! For instance,

```
> library(freqparcoord)
```

One can learn about the package in various ways. After loading it, for instance, you can list its objects, such as

```
> ls('package:freqparcoord')
[1] "freqparcoord" "knndens"      "knnreg"       "posjitter"
"regdiag"
[6] "regdiagbas"   "rmixmvnorm"   "smoothz"      "smoothzpred"
```

where we see objects (functions here) **knndens()** and so on. There is the **help()** function, e.g.

```
> help(package=freqparcoord)

                Information on package  freqparcoord

Description:

Package:        freqparcoord
Version:        1.1.0
Author:         Norm Matloff <normmatloff@gmail.com> and Yingkang Xie
                <yingkang.xie@gmail.com>
Maintainer:     Norm Matloff <normmatloff@gmail.com>
...
```

Some packages have *vignettes*, extended tutorials. Type

```
> vignette()
```

to see what's available.

### 1.20.2 The Functions tapply() and Its Cousins

In Section 1.6.2 we had occasion to use R's `tapply()`, a highly useful feature of the language. To explain it, let's start with useful function, **split()**.

Consider this tiny data frame:

```
> x
  gender height
1      m     66
2      f     67
3      m     72
4      f     63
```

Now let's split by gender:

```
> xs <- split(x,x$gender)
> xs
$f
  gender height
2      f     67
4      f     63
5      f     63

$m
  gender height
1      m     66
3      m     72
```

Note the types of the objects:

- **xs** is an R list
- **xs$f** and **xs$m** are data frames, the male and female subsets of **x**

We *could* then find the mean heights in each gender:

```
> mean(xs$f$height)
[1] 64.33333
```

```
> mean(xs$m$height)
[1] 69
```

But with **tapply()**, we can combine the two operations:

```
> tapply(x$height,x$gender,mean)
       f        m
64.33333 69.00000
```

The first argument of `tapply()` must be a vector, but the function that is applied can be vector-valued. Say we want to find not only the mean but also the standard deviation. We can do this:

```
> tapply(x$height,x$gender,function(w) c(mean(w),sd(w)))
$f
[1] 64.333333  2.309401

$m
[1] 69.000000  4.242641
```

Here, our function (which we defined "on the spot," within our call to **tapply()**, produces a vector of two components. We asked **tapply() to call that function on our vector of heights, doing so separately for each gender.**

### 1.20.3 Function Dispatch

The return value from a call to **lm()** is an object of R's S3 class structure; the class, not surprisingly, is named **"lm"**. It turns out that the functions **coef()** and **vcov()** mentioned in this chapter are actually related to this class, as follows.

Recall our usage, on the baseball player data:

```
> lmout <- lm(mlb$Weight ~ mlb$Height)
> coef(lmout) %*% c(1,72)
          [,1]
[1,] 193.2666
```

The call to **coef** extracted the vector of estimated regression coefficents (which we also could have obtained as **lmout$coefficents**). But here is what happened behind the scenes:

The R function **coef()** is a *generic function*, which means it's just a placeholder, not a "real" function. When we call it, the R interpreter says,

> This is a generic function, so I need to relay this call to the one associated with this class, **"lm"**. That means I need to check whether we have a function **coef.lm()**. Oh, yes we do, so let's call that.

That relaying action is referred to in R terminology as the original call being *dispatched* to **coef.lm()**.

This is a nice convenience. Consider another generic R function, **plot()**. No matter what object we are working with, the odds are that some kind of plotting function has been written for it. We can just call **plot()** on the given object, and leave it to R to find the proper call. (This includes the **"lm"** class; try it on our **lmout** above!)

Similarly, there are a number of R classes on which **coef()** is defined, and the same is true for **vcov()**.

One generic function we will use quite often, and indeed have already used in this chapter, is **summary()**. As its name implies, it summarizes (what the function's author believes) are the most important characteristics of the object. So, when this generic function is called on an **"lm"** object, the call is dispatched to **summary.lm()**, yielding estimated coefficients, standard errors and so on.

## 1.21 Further Exploration: Data, Code and Math Problems

**1**. Consider the **bodyfat** data mentioned in Section 1.2. Use **lm()** to form a prediction equation for **density** from the other variables (skipping the first three), and comment on whether use of indirect methods in this way seems feasible.

**2**. Suppose the joint density of $(X, Y)$ is $3s^2e^{-st},\ 1 < s < 2, 0 < t < -\infty$. Find the regression function $\mu(s) = E(Y|X = s)$.

**3**. For $(X, Y)$ in the notation of Section 1.19.3, show that the predicted value $\mu(X)$ and the predicton error $Y - \mu(X)$ are uncorrelated.

**4**. Suppose $X$ is a scalar random variable with density $g$. We are interested in the nearest neighbors to a point $t$, based on a random sample $X_1, ..., X_n$ from $g$. Find $L_k$ denote the cumulative distribution function of the distance of the $k^{th}$-nearest neighbor to $t$.

**5**. In Section 1.11.2, we used the R function **setdiff()** to form the training/test set partitioning. Show how we could use R's negative-index capability to do this as an alternate approach.