From Linear Models to Machine Learning

Regression and Classification, with R Examples

Norman Matloff

University of California, Davis

This is a draft of the first half of a book to be published in 2017 under the Chapman & Hall imprint. Corrections and suggestions are highly encouraged!

Contents

5 Multiclass Classification Problems 123

Preface

Regression analysis is both one of the oldest branches of statistics, with *least-squares* analysis having been first proposed way back in 1805, and 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.

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 precisely-stated yet always maintains as its top priority — borrowing from a book title of the late Leo Breiman — "a view toward applications."

Examples of what is different here:

One of the many ways in which this book is different from all other regression books is its 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.

Another example of how this book takes different paths than do others is its treatment of the well-known point that in addition to the vital Prediction goal of regression analysis, there is an equally-important Description goal. The book devotes an entire chapter to the latter (Chapter 7, Measuring 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*. Again, both parametric and nonparametric regression approaches are presesnted.

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 Eickert-White method; and variance-stabilizing transformations. The section titled "The Verdict" then argues for opting for the Eickert-White model if the goal is Description (and ignoring the problem if the goal is Prediction).

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.

Big Data:

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. Some minimal background is required (see below), but some readers will have some background in some aspects of the coverage of the book. The book aims to both accessible and valuable to such diversity of readership, following the old advice of Samuel Johnson that an author "should make the new familiar and the familiar new."[1]

Minimal background: The reader must of course be familiar with terms

[1] Cited in N. Schenker, "Why Your Involvement Matters, *JASA*, April 2015.

like *confidence interval*, *significance test* and *normal distribution*, and is assumed to have knowledge of basic matrix algebra, along with some experience with R. Most 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. Math stat is needed only for readers who wish to pursue the Mathematical Complements sections at the end of most chapters. Appendices provide brief introductions to R, matrices, math stat and statistical miscellania (e.g. the notion of a *standard error*).

The book can be used as a text at either the undergraduate or graduate level. For the latter, the Mathematical Complements sections would likely be included, whereas for undergraduates they may be either covered lightly or skipped, depending on whether the students have some math stat background.

Chapter outline:

Chapter 1: Setting the Stage: Regression as the conditional mean; parametric and nonparametric prediction models; Prediction and Description goals; classification as a special case of regression; parametric/nonparametric tradeoff; the need for cross-validation analysis.

Chapter 2, The Linear Regression Model: Least-squares estimation; statistical properties; inference methods, including for linear combinations of β; meaning and reliability of R^2; departures from the normality and homoscedasticity assumptions.

Chapter 4, Nonlinear Regression Models: Nonlinear modeling and computation; Generalized Linear Model; iteratively reweighted least squares; logistic model, motivations and interpretations; Poisson regression (including overdispersion and application to log-linear model); others.

Chapter 8: Shrinkage Methods: Multicollinearity in linear and nonlinear models; overview of James-Stein concepts; relation to non-full rank models; ridge regression and Tychonov regularization; LASSO and variants.

Chapter 10, Smoothing-Based Nonparametric Estimation: Estimation via k-nearest neighbor; kernel smoothing; choice of smoothing parameter; bias near support boundaries.

Chapter 6, Model Fit Assessment: Checking propriety of both the regression model and ancillary aspects such as homoscedasticity; residual analysis; nonparametric methods as "helpers"; a parallel-coordinates approach.

Chapter 9, Dimension Reduction: Precise discussion of *overfitting*; relation of the number of variables p to n, the number of data points; extent to which the Curse of Dimensionality is a practical issue; PCA and newer variants; clustering; classical variable-selection techniques, and new ones such as sparsity-based models; possible approaches with very large p.

Chapter 7, Measuring Factor Effects: Description as a goal of regression separate from Prediction; interpretation of coefficients in a linear model, in the presence (and lack of same) of other predictors; Simpson's Paradox, and a framework for avoiding falling victim to the problem; measurement of treatment effects, for instance those in a hospital quality-of-care example presented in Chapter 1; brief discussion of instrumental variables.

Chapter 11, Boundary-Based Classification Methods: Major nonparametric classification methodologies that essentially boil down to estimating the geometric boundary in X space between predicting Yes ($Y = 1$) and No ($Y = 0$); includes methods developed by statisticians (Fisher linear discriminant analysis, CART, random forests), and some developed in the machine learning community, such as support vector machines and neural networks; and brief discussion of bagging and boosting.

Chapter **??***: Outlier-Resistant Methods:* Leverage; quantile regression; robust regression.

Chapter 13: Miscellaneous Topics: Missing values; multiple inference; etc.

Appendices: Reviews of/quick intros to R and matrix algebra; odds and ends from probability modeling, e.g. iterated expectation and properties of covariance matrices; modeling of samples from populations, standard errors, delta method, etc.

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.

Software:

The book also makes use of some of my research results and associated software. The latter is in my package **regtools**, available from CRAN, GitHub and `http://heather.cs.ucdavis.edu/regress.html`. Errata lists, suggested data projects and so on may also be obtained there.

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 Stuart Ambler, Doug Bates, Frank Harrell, Benjamin Hofner, Michael Kane, Hyunseung Kang, John Mount, Art Owen, Yingkang Xie and Achim Zeileis.

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

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

- **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: Bodyfat Prediction

The great baseball player, Yogi Berra was often given to malapropisms, one of which was supposedly his comment, “Prediction is difficult, especially about the future.” 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**. 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 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.[2] 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.

[2] 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.

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 1.13.1 in the Mathematical Complements section at the end of this chapter, but what is important for now is to note that in the above prediction rule, we are dealing with conditional means: This is mean ridership, given day of the week is Sunday, sky conditions are sunny, and temperature is 62.

1.4 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[3] the conditional mean,

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

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

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 we calculate the proportion of Jones supporters among them, $\widehat{p}$, and then use that as our estimate of p.

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*.)

1.5 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 1.10.

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     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
```

1.5.1 Prediction vs. Description

Recall the Prediction and Description goals of regression analysis, discussed in Section 1.12.2. 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 1.7.1.2.

On the other hand, there doesn't seem to be much of a Prediction goal here. It is hard to imagine a need to predict a player's weight. However, for the purposes of explaining the concepts, we will often phrase things in a Prediction context. This is somewhat artificial, but it will serve our purpose of introducing the basic concepts in the very familiar setting of human characteristics.

So, suppose we will have a continuing stream of players for whom we only know height, 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

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

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 μ 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, $\widehat{\mu}(t)$.

1.5.2 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 190.3596. In other words, we would set $\widehat{\mu}(72) = 190.3596$, $\widehat{\mu}(74) = 202.4566$, and so on. (More detail on **tapply()** is given in the Code 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
```

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).

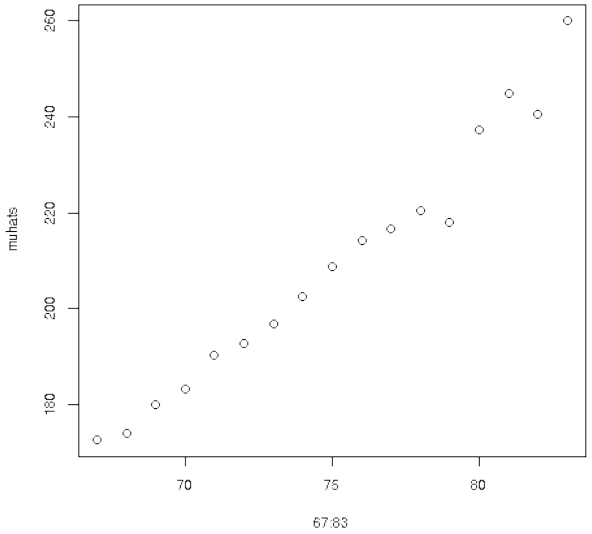

Figure 1.1: Plotted $\widehat{\mu}(t)$

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

1.5.3 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 of $\mu(t)$ 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 has 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 Section 1.5.2, 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:[4]

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

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

```
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)

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

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

$$(-151.133, 4.783) \left(\begin{array}{c} 1 \\ 72 \end{array} \right) \tag{1.9}$$

Be sure to see the need for that 1 in the second factor; it is used to multiply the -151.133.

Or, conveniently in R,[5] we can exploit the fact that R's **coef()** function fetches the coefficients c and d for us:

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

We can form a confidence interval from this too. The *standard error* (Appendix **??**) of $\breve{\mu}(72)$ will be shown later 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
```

[5] 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.

(More detail on **vcov()** and **coef()** is presented in the Code 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).

1.6 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 the 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.[6]

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.*

1.7 Several Predictor Variables

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

[6] 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.

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

$$\mu(t_1, ..., t_k) = E(Y \mid X^{(1)} = t_1, ..., X^{(k)} = t_k) \tag{1.10}$$

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

$$\mu(t) = E(Y \mid X = t) \tag{1.11}$$

with $t = (t_1, ..., t_k)'$ and $X = (X^{(1)}, ..., X^{(k)})'$, where $'$ denotes matrix transpose.[7]

1.7.1 Multipredictor Linear Models

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

$$\text{mean weight} = c + d \times \text{ height} + e \times \text{ age} \tag{1.13}$$

1.7.1.1 Estimation of Coefficients

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

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

[7] 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

$$\begin{pmatrix} 5 \\ 12 \\ 13 \end{pmatrix} \tag{1.12}$$

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.

For example, $\widehat{d} = 4.9236$. Our estimated regression function is

$$\widehat{\mu}(t_1, t_2) = -187.6382 + 4.9236\ t_1 + 0.9115\ t_2 \tag{1.14}$$

where t_1 and t_2 are height and age, respectively.

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

$$\widehat{\mu}(72, 25) = 189.6485 \tag{1.15}$$

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

1.7.1.2 The Description Goal

It was mentioned in Section 1.12.2 that regression analysis generally has one or both of two goals, Prediction and Description. In light of the lstter, 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.

1.7.2 Nonparametric Regression Estimation: k-NN

Now let's drop the linear model assumption (1.13), and estimate our regression function "from scratch," as we did in Section 1.5.2. But here we will need to broaden our approach, as follows.

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 suprising. But at any rate we thus we cannot set $\widehat{\mu}(72, 25)$ to be the mean weight among our sample data points satisfying those conditions, as we did in Section 1.5.2. 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.

1.7.3 Measures of Nearness

Nearness is generally defined as *Euclidean distance*:

$$\text{distance}[(s_1, s_2, ..., s_k), (t_1, t_2, ..., t_k)] = \sqrt{((s_1 - t_1)^2 + ... + (s_k - t_k)^2} \tag{1.16}$$

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

$$\sqrt{(72.5 - 72)^2 + (24.2 - 25)^2} = 0.9434 \tag{1.17}$$

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.

1.7.4 The Code

Here is code to perform k-NN regression estimation:

```
# arguments:
#
#  xydata: matrix or data frame of full (X,Y) data,
#          Y in last column
#  regestpts:  matrix or data frame of X vectors
#          at which to estimate the regression ftn
#   k:  number of nearest neighbors
#   scalefirst: call scale() on the data first
#
# value: estimated reg. ftn. at the given X values

knnest <-
      function(xydata,regestpts,k,scalefirst=FALSE) {
   require(FNN)
   if (is.vector(regestpts))
      regestpts <- matrix(regestpts,nrow=1)
   ycol <- ncol(xydata)
   x <- xydata[,-ycol,drop = F]
   if (scalefirst) {
      tmp <- rbind(x,regestpts)
      tmp <- scale(tmp)
      x <- tmp[1:nrow(x),]
      regestpts <- tmp[(nrow(x)+1):nrow(tmp),]
   }
   colnames(regestpts) <- colnames(x)
   y <- xydata[,ycol]
   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.

1.8 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?

1.8.1 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 1.7.1.1. 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, (1.14). From now on, whenever we need to predict a player's weight, given his height and age, we simply plug those values into (1.14).

1.8.2 Nonparametric Settings

The nonparametric case is a little more involved, because we have no explicit equation like (1.14). 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 (73.2,25.5), 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 $(X_1, Y_1), ..., (X_n, Y_m)$, 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 $\widehat{\mu}(X_i) > 0.5$.

1.9 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 vague description is not satisfactory, and it will be discussed in a precise manner in Chapter 9. But for now 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.[8]

1.9.1 Intuition

To see how overfitting may occur, consider the famous *bias-variance tradeoff*, 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.

[8] 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.

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 then 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. Accordingly, by adding more predictor variables in a regression model, 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 consists 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 confidence that the sample we have is representative.

But the more predictor variables we have, the more variability there is in the inputs to our regression calculations, and thus the larger the variances of the estimated coefficients.[9]

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.

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:

1.9.2 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:

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

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

1.9.3 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 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,])
}
```

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

1.9.4 Linear Model Case

To do cross-validation for linear models, we could use this code.[10]

1.9.4.1 The Code

[10] 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.

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

1.9.4.2 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

$$\left(\begin{array}{r} (-1,2)(3,8)' \\ (2,5)(3,8)' \end{array} \right) = \left(\begin{array}{c} 13 \\ 46 \end{array} \right) \tag{1.18}$$

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

$$\left(\begin{array}{rr} -1 & 2 \\ 2 & 5 \end{array} \right) \left(\begin{array}{r} 3 \\ 8 \end{array} \right) = \left(\begin{array}{r} 13 \\ 46 \end{array} \right) \tag{1.19}$$

1.9.4.3 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.

1.9.5 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.

1.9.6 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 2.7.5.

1.10 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* variables, such as Day of Week. The latter is technically numeric, 0 through 6, but those codes are just names.[11] There is no reason, for instance, that 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,[12] 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"
```

1.10.1 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

[11] Hence the term *nominal*.

[12] 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.

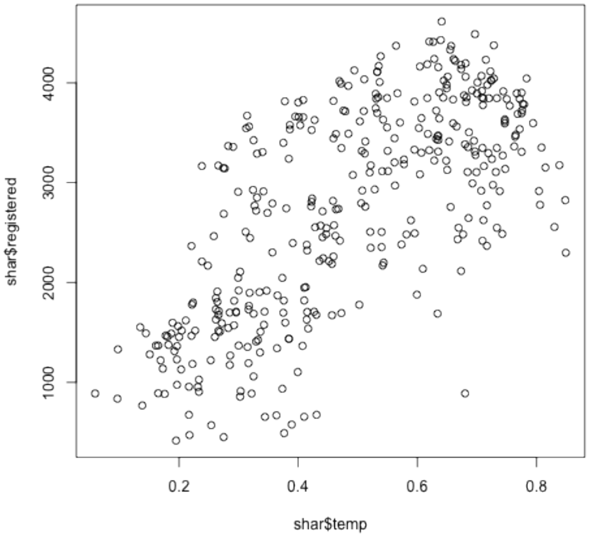

Figure 1.2:
Ridership vs. Temperature

to suggest that ridership is somewhat associated with temperature in the "first up, then later down" form as we had guessed.

Thus a linear model of the form

$$\text{mean ridership} = c + d \times \text{ temperature} \tag{1.20}$$

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

$$\text{mean ridership} = c + d \times \text{ temperature} + e \times \text{ temperature}^2 \tag{1.21}$$

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 (1.21) 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.

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**.[13]

[13] 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.

We incorporate this into our linear model:

$$\text{mean reg} = c + d \times \text{ temp} + e \times \text{ temp}^2 + f \text{ workingday} \tag{1.22}$$

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

$$\begin{aligned} \text{mean reg} &= \beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\ &+ \beta_3 \text{ workingday } + \beta_4 \text{ clearday} \end{aligned} \tag{1.23}$$

As is traditional, here we have used subscripted versions of the Greek letter β 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 typing of the data set name **shar** and clutter.)

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:[14]

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

[14] If you know about *dispatch* in R, invoking **print()** will cause a class-specfic function to be run, in this case **print.lm()**.

Remember, the population function $\mu(t)$ is unnown, so the β_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

$$\widehat{\beta}_3 = 988.5 \tag{1.24}$$

In other words, estimated regression function is

$$\widehat{\mu}(t_1, t_2, t_3, t_4) = -2310.3 + 17342.2 t_1 - 13434.7 t_2 + 988.5 t_3 + 760.3 t_4 \tag{1.25}$$

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 **temp** variable is scaled to [0,1], as

$$\frac{\text{Celsius temperature} - \text{minimum}}{\text{maximum} = \text{minimum}} \tag{1.26}$$

where the minimum and maximum here were -8 and 39, respectively. A Fahrenheit temperature of 62 degrees corresponds to a scaled value of 0.525. So, our predicted number of riders is

$$-2310.3 + 17342.2 \times 0.525 - 13434.7 \times 0.525^2 + 988.5 \times 0 + 760.3 \times 1 \tag{1.27}$$

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.

One can also form confidence intervals and perform significance tests on the β_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.

1.10.2 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.

1.11 Interaction Terms

Let's take another look at (1.23), specifically the term involving the variable **workingday**, a dummy indicating a nonholiday Monday through Friday. Our estimate for β_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**:

$$\begin{aligned}
\text{mean reg} &= \beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\
&+ \beta_3 \text{ workingday } + \beta_4 \text{ clearday} \qquad (1.28) \\
&+ \beta_5 \text{ temp } \times \text{ workingday} \qquad (1.29)
\end{aligned}$$

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

1.11.1 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

$$
\begin{aligned}
\text{mean wageinc} &= \beta_0 + \beta_1 \text{ age} + \beta_2 \text{ age}^2 + \beta_3 \text{ wkswrkd} \\
&+ \beta_4 \text{ ms} + \beta_5 \text{ phd} \\
&+ \beta_6 \text{ fem}
\end{aligned}
\tag{1.30}
$$

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 (1.30). Being female makes a β_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:[15]

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

Our model is now

$$\begin{aligned} \text{mean wageinc} &= \beta_0 + \beta_1 \text{ age} + \beta_2 \text{ age}^2 + \beta_3 \text{ wkswrkd} \\ &+ \beta_4 \text{ ms} + \beta_5 \text{ phd} \\ &+ \beta_6 \text{ fem} + \beta_7 \text{ msfem} + \beta_8 \text{ phdfem} \qquad (1.31) \end{aligned}$$

So, now instead of there being a single number for the "female effect," β_6, we how have two:

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

[15] 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.

```
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.

1.12 Classification Techniques

Recall the hospital example in Section 1.9.1. 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 **c**, and we are only interested in whether the requirement is satisfied. We could replace V with a new variable

$$Y = \begin{cases} 1, & \text{if } V > c \\ 0, & \text{if } V \leq c \end{cases} \tag{1.32}$$

Classification methods will play a major role in this book.

1.12.1 It's a Regression Problem!

Recall that the regression function is the conditional mean:

$$\mu(t) = E(Y \mid X = t) \tag{1.33}$$

(As usual, X and t may be vector-valued.) In the classification case, Y is an indicator variable, so from Appendx **??**, we know its mean is the probability that $Y = 1$. In other words,

$$\mu(t) = P(Y = 1 \mid X = t) \tag{1.34}$$

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,

$$\text{guess for } Y = \begin{cases} 1, & \text{if } \mu(X) > 0.5 \\ 0, & \text{if } \mu(X) \leq 0.5 \end{cases} \tag{1.35}$$

It turns out that this strategy is optimal, in that it minimizes the overall misclassification error rate (see Section 1.13.2 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 (1.34) is a population quantity. We'll need to estimate it from our sample data.

1.12.2 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

$$\begin{aligned}\text{probability of HighUsage} &= \beta_0 + \beta_1 \text{ temp} + \beta_2 \text{ temp}^2 \\ &+ \beta_3 \text{ workingday } + \beta_4 \text{ clearday} \quad (1.36)\end{aligned}$$

could be used, but would not be very satisfying. The left-hand side of (1.36), 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*:

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

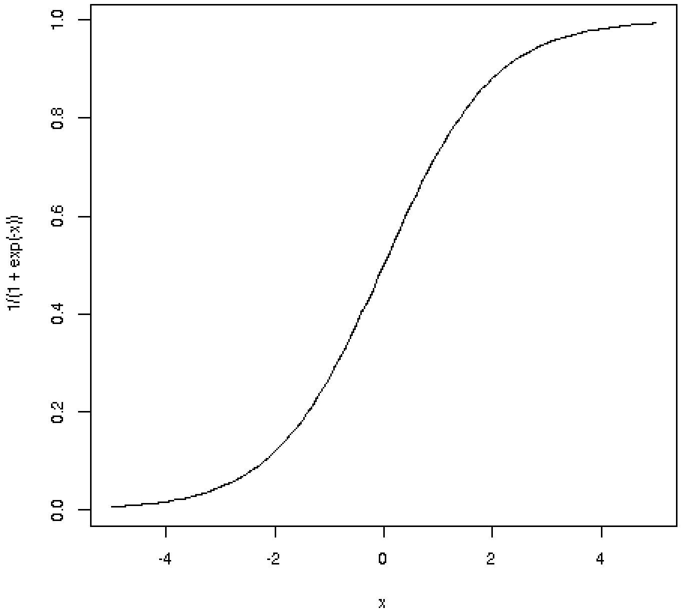

Figure 1.3: Logistic Function

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

$$\ell(s) = \frac{1}{1 + e^{-s}} \tag{1.38}$$

Our model, then is

$$\mu(t_1, t_2, t_3, t_4) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 t_1 + \beta_2 t_2 + \beta_3 t_3 + \beta_4 t_4)}} \tag{1.39}$$

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 (1.37), just as was the case in (1.23) 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,[16], which is designated via **family = binomial**:

```
> 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.

1.13 Mathematical Complements

Certain claims of optimality were made in this chapter. Let's prove them.

1.13.1 $\mu(t)$ 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, $\mu(t) = E(Y \mid X = t)$.*

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

(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 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

$$E[(W-c)^2] = E(W^2 - 2cW + c^2] = E(W^2) - 2cEW + c^2 \tag{1.40}$$

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

Now to show the original claim, use iterated expectation (Appendix **??**) to write

$$E[(Y - f(X))^2] = E\left[E((Y - f(X))^2|X)\right] \tag{1.41}$$

In the inner expectation, X is a constant, and from the above 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.

1.13.2 $\mu(t)$ 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*

$$\widehat{Y} = \begin{cases} 1, & \text{if } \mu(X) > 0.5 \\ 0, & \text{if } \mu(X) \leq 0.5 \end{cases} \tag{1.42}$$

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

$$\begin{aligned} P(C = g) &= P(C = 1)g + P(C = 0)(1 - g) && (1.43) \\ &= pg + (1 - p)(1 - g) && (1.44) \\ &= [2p - 1]g + 1 - p && (1.45) \end{aligned}$$

Now remember, p is known. How should we choose g, 0 or 1, in order to maximize (1.45), the probability that our guess is correct? Inspecting (1.45) 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 returning to our original claim, write

$$P(\widehat{Y} = Y) = E\left[P(\widehat{Y} = Y \mid X)\right] \tag{1.46}$$

In that inner probability, "p" is

$$P(Y = 1 \mid X) = \mu(X) \tag{1.47}$$

which completes the proof.

1.14 Code Complements

1.14.1 The Functions tapply() and Its Cousins

In Section 1.5.2 we had occasion to use R's `tapply()`, a hihgly 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.**

1.15 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(\lambda$Weight ~ \lambda$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()**.

Exercises

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 1.13.1, 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.

Chapter 2

Linear Regression Models

In this chapter we go into the details of linear models. Let's first set some notation, to be used here and in the succeeding chapters.

2.1 Notation

Let Y be our response variable, and let $X = (X^{(1)}, X^{(2)}, ..., X^{(p)})'$ denote the vector of our p predictor variables. Using our weight/height/age baseball player example from Chapter 1 as our running example here, we would have $p = 2$, and Y, $X^{(1)}$ and $X^{(2)}$ would be weight, height and age, respectively.

Our sample consists of n data points, $X_1, X_2, ..., X_n$, each a p-element predictor vector, and $Y_1, Y_2, ..., Y_n$, associated scalars. In the baseball example, n was 1015. Also, the third player had height 72, was of age 30.78, and weighed 210. So,

$$X_3 = \begin{pmatrix} 72 \\ 30.78 \end{pmatrix} \tag{2.1}$$

and

$$Y_3 = 210 \tag{2.2}$$

Write the X_i in terms of their components:

$$X_i = (X_i^{(1)}, ..., X_i^{(p)})' \tag{2.3}$$

$$X = (X^{(1)}, ..., X^{(p)})' \tag{2.4}$$

So, again using the baseball player example, the height, age and weight of the third player would be $X_3^{(1)}$, $X_3^{(2)}$ and Y_3, respectively.

And just one more piece of notation: We sometimes will need to augment a vector with a 1 element at the top, such as we did in (1.9). Our notation for this will consist of a tilde above the symbol,

For instance,

$$\widetilde{X_3} = \begin{pmatrix} 1 \\ 72 \\ 30.78 \end{pmatrix} \tag{2.5}$$

So, our linear model is, for a p-element vector $t = (t_1, ..., t_p)'$,

$$\mu(t) = \beta_0 + \beta_1 \ t_1 + ... + \beta_p \ t_p = \widetilde{t} \ ' \ \beta \tag{2.6}$$

In the baseball example, with both height and weight as predictors:

$$\begin{aligned} \mu((\text{height,age}) &= \beta_0 + \beta_1 \text{ height} + \beta_2 \text{ age} \qquad (2.7) \\ &= (1, \text{height}, \text{age})' \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} \qquad (2.8) \end{aligned}$$

2.2 Random- vs. Fixed-X Cases

We will usually consider the X_i and Y_i to be random samples from some population, so that $(X_1, Y_1), ..., (X_n, Y_n)$ are independent and identically distributed (i.i.d.) according to the population. This is a *random-X* setting, meaning that both the X_i and Y_i are random. There are some situations in which the X-values are fixed by design, known as a *fixed-X* setting. This might be the case in chemistry research, in which we decide in advance to perform experiments at specific levels of concentration of some chemicals.

2.3 Least-Squares Estimation

Linear regression analysis is sometimes called *least-squares estimation.* Let's first look at how this evolved.

2.3.1 Motivation

As noted in Section 1.13, $\mu(X)$ minimizes the mean squared prediction error,

$$E[(Y - f(X))^2] \tag{2.9}$$

over all functions f. And since our assumption is that $\mu(X) = \widetilde{X}\ '\beta$, we can also say that setting $b = \beta$ minimizes

$$E[(Y - \widetilde{X}'b))^2] \tag{2.10}$$

over all vectors b.

If $W_1, ..., W_n$ is a sample from a population having mean EW, the sample analog of EW is $\overline{W} = (\sum_{i=1}^n W_i)/n$; one is the average value of W in the population, and the other is the average value of W in the sample. Let's write this correspondence as

$$EW \longleftrightarrow \overline{W} \tag{2.11}$$

It will be crucial to always keep in mind the distinction between population values and their sample estimates, especially when we discuss overfitting in detail.

Similarly, for any fixed b, (2.10) is a population quantity, the average squared error using b for prediction in the population (recall Section 1.4). The population/sample correspondence here is

$$E[(Y - \widetilde{X}'b))^2] \longleftrightarrow \frac{1}{n}\sum_{i=1}^n (Y_i - \widetilde{X_i}'b)^2 \tag{2.12}$$

where the right-hand side is the the average squared error using b for prediction in the sample.

So, since β is the value of b minimizing (2.10), it is intuitive to take our estimate, $\widehat{\beta}$, to be the value of b that minimizes (2.12). Hence the term *least squares*.

To find the minimizing b, we could apply calculus, taking the partial derivatives of (2.12) with respect to $b_i,\ i = 0, 1, ..., p$, set them to 0 and solve. Fortunately, R's **lm()** does all that for us, but it's good to know what is happening inside. Also, this will give the reader more practice with matrix expressions, which will be important in some parts of the book.

2.3.2 Matrix Formulations

Use of matrix notation in linear regression analysis greatly compactifies and clarifies the presentation. You may find that this requires a period of adjustment at first, but it will be well worth the effort.

Let A denote the $n \times p$ matrix of X values in our sample,

$$A = \begin{pmatrix} \widetilde{X_1}' \\ \widetilde{X_2}' \\ ... \\ \widetilde{X_p}' \end{pmatrix} \tag{2.13}$$

and let D be the $n \times 1$ vector of Y values,

$$D = \begin{pmatrix} Y_1 \\ Y_2 \\ ... \\ Y_n \end{pmatrix} \tag{2.14}$$

In the baseball example, row 3 of A is

$$(1, 30.78, 72) \tag{2.15}$$

and the third element of D is 210.

2.3.3 (2.12) in Matrix Terms

Our first order of business will be to recast (2.12) as a matrix expression. To start, look at the quantities $\widetilde{X_i}'b, \ i = 1, ..., n$ there in (2.12). Stringing them together in matrix form as we did in (1.19), we get

$$\begin{pmatrix} \widetilde{X_1}' \\ \widetilde{X_2}' \\ ... \\ \widetilde{X_n}' \end{pmatrix} b = Ab \tag{2.16}$$

Now consider the n summands in (2.12), before squaring. Stringing them into a vector, we get

$$D - Ab \tag{2.17}$$

We need just one more step: Recall that for a vector $a = (a_1, ..., a_k)'$,

$$\sum_{i=1}^{k} a_k^2 = a'a \tag{2.18}$$

In other words, (2.12) (except for the $1/n$ factor) is actually

$$(D - Ab)'(D - Ab) \tag{2.19}$$

Now that we have this in matrix form, we can go about finding the optimal b.

2.3.4 Using Matrix Operations to Minimize (2.12)

Remember, we will set $\widehat{\beta}$ to whatever value of b minimizes (2.19). Thus we need to take the derivative of that expression with respect to b, and set the result to 0. There is a theory of matrix derivatives, not covered here, but the point is that the derivative of (2.19) with respect to b can be shown to be

$$-2A'(D - Ab) \tag{2.20}$$

(Intuitively, this is the multivariate analog of

$$\frac{\partial}{\partial b}(d-ab)^2 = -2(d-ab)a \tag{2.21}$$

for scalar d, a, b.)

Setting this to 0, we have

$$A'D = A'Ab \tag{2.22}$$

Solving for b we have our answer:

$$\widehat{\beta} = (A'A)^{-1}A'D \tag{2.23}$$

This is what **lm()** calculates![1]

2.4 A Closer Look at lm() Output

Since the last section was rather abstract, let's get our bearings by taking a closer look at the output in the baseball example:[2]

```
> lmout <- lm(mlb$Weight ~ mlb$Height + mlb$Age)
> summary(lmout)
...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -187.6382    17.9447  -10.46  < 2e-16
mlb$Height     4.9236     0.2344   21.00  < 2e-16
mlb$Age        0.9115     0.1257    7.25 8.25e-13

(Intercept) ***
mlb$Height  ***
mlb$Age     ***
---
Signif. codes:
   0    ***    0.001    **    0.01
```

[1] For the purpose of reducing roundoff error, it uses the *QR decomposition* in place of the actual matrix inversion. But algoritthmically they are equivalent.

[2] Note the use of the ellipsis ..., indicating that portions of the output have been omitted, for clarity.

```
     *     0.05     .     0.1          1
...
Multiple R-squared:  0.318,
Adjusted R-squared:  0.3166
...
```

There is a lot here! Let's get an overview, so that the material in the coming sections will be better motivated.

2.4.1 Statistical Inference

The **lm()** output is heavily focused on *ststistical inference* — forming confidence intervals and performing significance tests — and the first thing you may notice is all those asterisks: The estimates of the intercept, the height coefficient and the age coefficient all are marked with three stars, indicating a p-value of less than 0.001. The test of the hypothesis

$$H_0 : \beta_1 = 0 \tag{2.24}$$

would be resoundingly rejected, and one could say, "Height has a significant effect on weight." Not surprising at all, though the finding for age might be more interesting, in that we expect athletes to keep fit, even as they age.

We could form a confidence interval for β_2, for instance, by adding and subtracting 1.96 times the associated standard error,[3] which is 0.1257 in this case. Our resulting CI would be about (0.66,1.16), indicating that the average player gains between 0.66 and 1.16 pounts per year; even baseball players gain weight over time.

2.4.2 Assumptions

But where did this come from? Surely there must be some assumptions underlying these statistical inference procedures. What are those assumptions? The classical ones, to which the reader may have some prior exposure, are:[4]

[3] The *standard error of an estimate* $\widehat{\theta}$ is the estimated standard deviation of that estimator. More on this coming soon.

[4] Of course, an assumption not listed here is that the linear model (2.6) is correct, at least to reasonable accuracy.

- **Normality:** The assumption is that, conditional on the vector of predictor variables X, the response variable Y has a normal distribution.

  In the weight/height/age example, this would mean, for instance, that within the subpopulation of all baseball players of height 72 and age 25, weight is normally distributed.

- **Homoscedasticity:** Just as we define the regression function in terms of the conditional mean,

  $$\mu(t) = E(Y \mid X = t) \tag{2.25}$$

  we can define the conditional variance function

  $$\sigma^2(t) = Var(Y \mid X = t) \tag{2.26}$$

  The homoscedasticity assumption is that $\sigma^2(t)$ does not depend on t.

  In the weight/height/age example, this would say that the variance in weight among, say, 70-inches-tall 22-year-olds is the same as that among the subpopulation of those of height 75 inches and age 32.

That first assumption is often fairly good, though we'll see that it doesn't matter much anyway. But the second assumption is just the opposite — it rarely holds even in an approximate sense, and in fact it turns out that it does matter. More on this in Chapter **??**.

The "R-squared" values will be discussed in Section 2.7.

2.5 Unbiasedness and Consistency

We will begin by discussing two properties of the least-squares estimator $\widehat{\beta}$.

2.5.1 $\widehat{\beta}$ Is Unbiased

One of the central concepts in the early development of statistics was *unbiasedness.* As you'll see, to some degree it is only historical baggage, but on the other hand it does become quite relevant in some contexts here.

To explain the concept, say we are estimating some population value θ, using an estimator $\widehat{\theta}$ based on our sample. Remember, $\widehat{\theta}$ is a random variable — if we take a new sample, we get a new value of $\widehat{\theta}$. So, some samples will yield a $\widehat{\theta}$ that overestimates θ, while in other samples $\widehat{\theta}$ will come out too low.

The pioneers of statistics believed that a nice property for $\widehat{\theta}$ to have would be that *on average*, i.e., averaged over all possible samples, $\widehat{\theta}$ comes out "just right":

$$E\widehat{\theta} = \theta \tag{2.27}$$

This seems like a reasonable criterion for an estimator to have, and sure enough, our least-squares estimator has that property:

$$E\widehat{\beta} = \beta \tag{2.28}$$

Note that since this is a vector equation, the unbiasedness is meant for the individual components. In other words, (2.28) is a compact way of saying

$$E\widehat{\beta}_j = \beta_j, \; j = 0, 1, ..., p \tag{2.29}$$

This is derived in the Mathematical Complements portion of this chapter, Section 2.10.2.

2.5.2 Bias As an Issue/Nonissue

Arguably the pioneers of statistics shouldn't have placed so much emphasis on unbiasednedness. Most statistiical estitmators have some degree of bias, though it is usually small and goes to 0 as the sample size n grows. Other than least-squares, none of the regression function estimators in common use is unbiased.

Indeed, there is a common estimator, learned in elementary statistics courses, that is arguably wrongly heralded as unbiased. This is the sample variance based on $W_1, ..., W_n$,

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (W_i - \overline{W})^2 \tag{2.30}$$

estimating a population variance η^2.

The "natural," sample-analog divisor in (2.30) would be n, not $n-1$. Using our "correspondence" notation,

$$\eta^2 \longleftrightarrow \frac{1}{n} \sum_{i=1}^{n} (W_i - \overline{W})^2 \tag{2.31}$$

But as the reader may be aware, use of n as the divisor would result in a biased estimate of η^2. The $n-1$ divisor is then a "fudge factor" that can be shown to produce an unbiased estimator.

Yet even that is illusory. While it is true that S^2 (with the $n-1$ divisor) is an unbiased estimate of η^2, we actually don't have much use for S^2. Instead, we use S, as in the familiar t-statistic, (2.38) for inference on means. And lo and behold, S is a *biased* estimator of η! It is shown in Section (2.10.4) that

$$ES < \eta \tag{2.32}$$

Thus one should indeed not be obsessive in pursuing unbiasedness.

However, the issue of bias does play an important role in various aspects of regression analysis. It will arise often in this book, including in the present chapter.

2.5.3 $\widehat{\beta}$ Is Consistent

In contrast to unbiasedness, which as argued above may not be a general goodness criterion for an estimator, there is a more basic property that we would insist that almost any estimator to have, *consistency*: As the sample size n goes to infinity, then the sample estimate $\widehat{\theta}$ goes to θ. This is not a very strong property, but it is a minimal one. It is shown in Section 2.10.3 that the least-squares estimator $\widehat{\beta}$ is indeed a consistent estimator of β.

2.6 Inference under Homoscedasticity

Let's see what the homoscedasticity assumption gives us.

2.6.1 Review: Classical Inference on a Single Mean

You may have noticed the familiar Student-t distribution mentioned in the output of **lm()** above. Before proceeding, it will be helpful to review this situation from elementary statistics.

> We have a random sample $W_1, ..., W_n$ from a population having mean $\nu = EW$ and variance η^2. Suppose W is normally distributed in the population. Form
>
> $$\overline{W} = \frac{1}{n} \sum_{i=1}^{n} W_i \tag{2.33}$$
>
> and
>
> $$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (W_i - \overline{W})^2 \tag{2.34}$$
>
> (It is customary to use the lower-case s instead of S in 2.34, but the capital letter is used here so as to distinguish from the s^2 quantity in the linear model, (2.47).)
>
> Then
>
> $$T = \frac{\overline{W} - \nu}{S/\sqrt{n}} \tag{2.35}$$
>
> has a Student-t distribution with $n-1$ degrees of freedom (df).
>
> This is then used for statistical inference on ν. We can form a 95% confidence interval by adding and subtracting $c \times S/\sqrt{n}$ to $\overline{W}$, where c is the point of the upper-0.025 area for the Student-t distribution with $n-1$ df.
>
> Under the normality assumption, such inference is exact; a 95% confidence interval, say, has exactly 0.95 probability of containing ν.

The normal distribution model is just that, a model, not expected to be exact. It rarely happens, if ever at all, that a population distribution is exactly normal. Human weight, for instance, cannot be negative and cannot be a million pounds; it is bounded, unlike normal distributions, whose support is $(-\infty, \infty)$. So "exact" inference using the Student-t distribution as above is not exact after all. (Though the following discussion might have been postponed to Chapter 3, it's important to get the issue out of the way earlier, right here.)

If n is large, the assumption of a normal population becomes irrelevant: The Central Limit Theorem (CLT, Appendix **??**) tells us that

$$\frac{\overline{W} - \nu}{\eta/\sqrt{n}} \tag{2.36}$$

has an approximate N(0,1) distribution *even though the distribution of W is not normal.* We then must show that if we replace η by S in (2.36), the result will still be approximately normal. This follows from *Slutsky's Theorem* and the fact that S goes to η as $n \to \infty$.[5] Thus we can perform approximate) statistical inference on ν using (2.35) and N(0,1), again *without assuming that W has a normal distribution.*

For instance, since the upper 2.5% tail of the N(0,1) distribution starts at 1.96, an approximate 95% confidence interval for ν would be

$$\overline{W} \pm 1.96 \frac{S}{\sqrt{n}} \tag{2.37}$$

What if n is small? We could use the Student-t distribution anyway, but we would have no idea how accurate it would be. We could not even use the data to assess the normality assumption on which the t-distribution is based, as we would have too little data to do so.

The normality assumption for the W_i, then, is of rather little value, and as explained in the next section, is of even less value in the regression context.

One possible virtue, though, of using Student-t would be that it gives a wider interval than does N(0,1). For example, for $n = 28$, our confidence interval would be

$$\overline{W} \pm 2.04 \frac{s}{\sqrt{n}} \tag{2.38}$$

instead of (2.37). The importance of this is that using S instead of η adds further variability to (2.35), which goes away as $n \to \infty$ but makes (2.36)

[5] In its simpler form, the theorem says that if U_n converges to a normal distribution and $V_n \to v$ as $n \to \infty$, then U_n/V_n also is asymptotically normal.

overly narrow. Using a Student-t value might compensate for that, though it may also overcompensate.

In general:

> If $\widehat{\theta}$ is an approximately normally-distributed estimator of a population value θ, then an approximate 95% confidence interval for θ is
>
> $$\widehat{\theta} \pm 1.96 \; s.e.(\widehat{\theta}) \tag{2.39}$$
>
> where the notation s.e.() denotes "standard error of." (The standard error usually comes from a theoretical derivation, as we will see.)

2.6.2 Extension to the Regression Case

The discussion in the last section concerning inference for a mean. What about inference for regression functions (which are conditional means)?

The first point to note is this:

> The distribution of the least-squares estimator $\widehat{\beta}$ is approximately $(p+1)$-variate normal, *without* assuming normality.[6]

This again follows from the CLT. Consider for instance a typical component of $A'D$ in (2.23),

$$\sum_{i=1}^{n} X_i^{(j)} Y_i \tag{2.40}$$

This is a sum of i.i.d. terms, thus approximately normal. The *delta method* (Appendix **??**) says that smooth (i.e. differentiable) functions of asymptotically normal random variables are again asymptotically normal. So, $\widehat{\beta}$ has an asymptotic $(p+1)-$variate normal distribution. (A more formal derivation is presented in Section 2.10.6.)

However, to perform statistical inference, we need the approximate covariance matrix of $\widehat{\beta}$, from which we can obtain standard errors of the $\widehat{\beta}_j$. *The standard way to do this is by assuming homoscedasticity.*

[6] The statement is true even without assuming homoscedasticity, but we won't drop that assumption until the next chapter.

So, **lm()** assumes that in (2.26), the function $\sigma^2(t)$ is constant in t. For brevity, then, we will simply refer to it as σ^2. Note that this plus our independence assumption implies

$$Cov(D|A) = \sigma^2 I \tag{2.41}$$

where I is the identity matrix.

To avoid (much) clutter, let $C = (A'A)^{-1}A'$. Then by the properties of covariance matrices (Appendix **??**),

$$\begin{aligned} Cov(\widehat{\beta}\ |A) &= Cov(CD) && (2.42)\\ &= C\ Cov(D|A)\ C' && (2.43)\\ &= \sigma^2 CC' && (2.44) \end{aligned}$$

Fortunately, the various properties of matrix transpose (Appendix **??**) can be used to show that

$$CC' = (A'A)^{-1} \tag{2.45}$$

Thus

$$Cov(\widehat{\beta}) = \sigma^2 (A'A)^{-1} \tag{2.46}$$

That's a nice (surprisingly) compact expression, but the quantity σ^2 is an unknown population value. It thus must be estimated, as we estimated η^2 by S^2 in Section 2.6.1. And again, an unbiased estimator is available. So, we take as our estimator of σ^2

$$s^2 = \frac{1}{n-p-1} \sum_{i=1}^{n} (Y_i - \widetilde{X}_i' \widehat{\beta})^2 \tag{2.47}$$

which can be shown to be unbiased.

If the normality assumption were to hold, then quantities like

$$\frac{\widehat{\beta}_i - \beta_i}{s\sqrt{a_{ii}}} \tag{2.48}$$

would have an exact Student-t distribution with $n-p-1$ degrees of freedom, where a_{ii} is the (i,i) element of $(A'A)^{-1}$.[7]

But as noted, this is usually an unrealistic assumption, and we instead rely on the CLT. Putting the above together, we have:

> The conditional distribution of the least-squares estimator $\widehat{\beta}$, given A, is approximately multivariate normal distribution with mean β and approximate covariance matrix
>
> $$s^2(A'A)^{-1} \tag{2.49}$$
>
> Thus the standard error of $\widehat{\beta}_j$ is the square root of element j of this matrix (counting the top-left element as being in row 0, column 0).
>
> Similarly, suppose we are interested in some linear combination $\lambda'\beta$ of the elements of β, estimating it by $\lambda'\widehat{\beta}$. The standard error is the square root of
>
> $$s^2\lambda'(A'A)^{-1}\lambda \tag{2.50}$$

And as before, we might as well calculate s^2 with a denominator of n, as opposed to the $n-p-1$ expression above.

Recall from Chapter 1, by the way, that R's **vcov()** function gives us the matrix (3.3), both for **lm()** and also for some other regression modeling functions that we will encounter later.

Before going to some examples, note that the conditional nature of the statements above is not an issue. Say for instance we form a 95% confidence interval for some quantity, conditional on A. Let V be an indicator variable for the event that the interval contains the quantity of interest. Then

$$P(V=1) = E[P(V=1 \mid A)] = E(0.95) = 0.95 \tag{2.51}$$

Thus the unconditional coverage probability is still 0.95.

[7] A common interpretation of the number of degrees of freedom here is, "We have n data points, but must subtract one degree of freedom for each of the $p+1$ estimated parameters."

2.6.3 Example: Bike-Sharing Data

Let's form some confidence intervals from the bike-sharing data.

```
> lmout <- lm(reg ~ temp+temp2+workingday+clearday ,
     data=shar)
> summary(lmout)
....
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1362.56     232.82  -5.852 1.09e-08
temp        11059.20     988.08  11.193  < 2e-16
temp2       -7636.40    1013.90  -7.532 4.08e-13
workingday    685.99      71.00   9.661  < 2e-16
clearday      518.95      69.52   7.465 6.34e-13
...
Multiple R-squared:  0.6548,  Adjusted R-squared:  0.651
```

We estimate that a working day adds about 686 riders to the day's ridership. An approximate 95% confidence interval for the population value for this effect is

$$685.99 \pm 1.96 \cdot 71.00 = (546.83, 825.15) \tag{2.52}$$

This is a disappointingly wide interval, but it shouldn't surprise us. After all, it is based on only 365 data points.

Given the nonlinear effect of temperature in our model, finding a relevant confidence interval here is a little more involved. Let's compare the mean ridership for our example in the last chapter — 62 degree weather, a Sunday and sunny — with the same setting but with 75 degrees.

The difference in (population!) mean ridership levels between these two settings is

$$(\beta_0 + \beta_1 0.679 + \beta_2 0.679^2 + \beta_3 0 + \beta_1 1) - (\beta_0 + \beta_1 0.525 + \beta_2 0.525^2 + \beta_3 0 + \beta_1 1)$$

$$= \beta_1 0.154 + \beta_2 0.186$$

Our sample estimate for that difference in mean ridership between the two types of days is then obtained as follows:

```
> lamb <- c(0,0.154,0.186,0,0)
> t(lamb) %*% coef(lmout)
          [,1]
[1,] 282.7453
```

or about 283 more riders on the warmer day. For a confidence interval, we need a standard error. So, in (3.4), take $\lambda = (0, 0.154, 0.186, 0, 0)'$. Our standard error is then obtained via

```
> sqrt(t(lamb) %*% vcov(lmout) %*% lamb)
          [,1]
[1,] 47.16063
```

Our confidence interval for the difference between 75-degree and 62-degree days is

$$282.75 \pm 1.96 \cdot 47.16 = (190.32, 375.18) \tag{2.53}$$

Again, a very wide interval, but it does appear that a lot more riders show up on the warmer days.

The value of s is itself probably not of major interest, as its use is usually indirect, in (3.3). However, we can determine it if need be, as **lmout$residuals** contains the *residuals*, i.e. the sample prediction errors

$$Y_i - \widetilde{X_i}'\widehat{\beta},\ i = 1, 2, ..., n \tag{2.54}$$

Using (2.47), we can find s:

```
> s <- sqrt(sum(lmout$residuals^2) / (365-4-1))
> s
> s
[1] 626.303
```

2.7 Collective Predictive Strength of the $X^{(j)}$

The R^2 quantity in the output of **lm()** is a measure of how well our model predicts Y. Yet, just as $\widehat{\beta}$, a sample quantity, estimates the population quantity β, one would reason that the R^2 value printed out by **lm()** must estimate a population quantity too. In this section, we'll make that concept precise, and deal with a troubling bias problem.

We will also introduce an alternative form of the cross-validation notion discussed in Section 1.9.3.

2.7.1 Basic Properties

Note carefully that we are working with population quantities here, generally unknown, but existent nonetheless. Note too that, for now, we are NOT assuming normality or homoscedasticity. In fact, even the assumption of having a linear regression function will be dropped for the moment.

Suppose we somehow knew the exact population regression function $\mu(t)$. Whenever we would encounter a person/item/day/etc. with a known X but unknown Y, we would predict the latter by $\mu(X)$. Define ϵ to be the prediction error

$$\epsilon = Y - \mu(X) \tag{2.55}$$

It can be shown (Section 2.10.5) that $\mu(X)$ and ϵ are uncorrelated, i.e., have zero covariance. We can thus write

$$Var(Y) = Var[\mu(X)] + Var(\epsilon) \tag{2.56}$$

With this partitioning, it makes sense to say:

> The quantity
>
> $$\omega = \frac{Var[\mu(X)]}{Var(Y)} \tag{2.57}$$
>
> is the proportion of variation of Y explainable by X.

Section 2.10.5 goes further:

> Define
>
> $$\rho = \sqrt{\omega} \tag{2.58}$$
>
> Then ρ is the correlation between our prodicted value $\mu(X)$ and the actual Y.

Again, the normality and homoscedasticity assumptions are NOT needed for these results. In fact, *they hold for any regression function, not just one satisfying the linear model.*

2.7.2 Definition of R^2

The quantity R^2 output by **lm()** is the sample analog of ρ^2:

> R^2 is the squared sample correlation between the actual response values Y_i and the predicted values $\widetilde{X}_i' \widehat{\beta}$. Also, R^2 is a consistent estimator of ρ^2.

Exactly how is R^2 defined? From (2.56) and (2.57), we see that

$$\rho^2 = 1 - \frac{Var[\epsilon]}{Var(Y)} \tag{2.59}$$

Since $E\epsilon = 0$, we have

$$Var(\epsilon) = E(\epsilon^2) \tag{2.60}$$

The latter is the average squared prediction error in the population, whose sample analog is the average squared error in our sample. In other words, using our "correspondence" notation from before,

$$E(\epsilon^2) \longleftrightarrow \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widetilde{X}_i'\widehat{\beta})^2 \tag{2.61}$$

Now considering the denominator in (2.59), the sample analog is

$$Var(Y) \longleftrightarrow \frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y})^2 \tag{2.62}$$

where of course $\overline{Y} = (\sum_{i=1}^{n} Y_i)/n$.

And that is R^2:

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widetilde{X}_i'\widehat{\beta})^2}{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y})^2} \tag{2.63}$$

(Yes, the $1/n$ factors do cancel, but it will be useful to leave them there.)

As a sample estimate of the population ρ^2, the quantity R^2 would appear to be a very useful measure of the collective predictive ability of the $X^{(j)}$. However, the story is not so simple, and curiously, the problem is actually bias.

2.7.3 Bias Issues

R^2 can be shown to be biased upward, not surprising in light of the fact that we are predicting on the same data that we had used to calculate $\widehat{\beta}$. In the extreme, we could fit an $n-1$ degree polynomial in a single predictor, with the curve passing through each data point, producing $R^2 = 1$, even though our ability to predict future data would likely be very weak.

The bias can be severe if p is a substantial portion of n. (In the above polynomial example, we would have $p = n-1$, even though we started with $p = 1$.) This is the *overfitting* problem mentioned in the last chapter, and to be treated in depth in a later chapter. But for now, let's see how bad the bias can be, using the following simulation code:

```
simr2 <- function(n,p,nreps) {
    r2s <- vector(length=nreps)
    for (i in 1:nreps) {
        x <- matrix(rnorm(n*p),ncol=p)
        y <- x %*% rep(1,p) + rnorm(n,sd=sqrt(p))
        r2s[i] <- getr2(x,y)
    }
    hist(r2s)
}

getr2 <- function(x,y) {
    smm <- summary(lm(y ~ x))
    smm$r.squared
}
```

Here we are simulating a population in which

$$Y = X^{(1)} + ... + X^{(p)} + \epsilon \tag{2.64}$$

so that β consists of a 0 followed by p 1s. We set the $X^{(j)}$ to have variance 1, and ϵ has variance $\sqrt{p}$. This gives $\rho^2 = 0.50$. Hopefully R^2 will usually

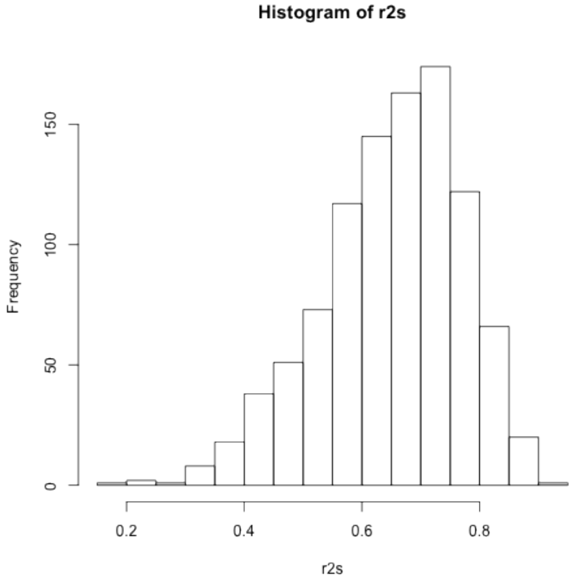

Figure 2.1: Plotted R^2 Values, n = 25

be near this value. To assess this, I ran **simr2(25,8,1000)**, with the result shown in Figure 2.1.

These results are not encouraging at all! The R^2 values are typically around 0.7, rather than 0.5 as they should be. In other words, R^2 is typically giving us much too rosy a picture as to the predictive strength of our $X^{(j)}$.

Of course, it should be kept in mind that I deliberately chose a setting which produced substantial overfitting — 8 predictors for only 25 data points, which you will see in Chapter 9 is probably too many predictors.

Running the simulation with $n = 250$ should show much better behavior. The results are shown in Figure 2.2. This is indeed much better. Note, though, that the upward bias is still evident, with values more typically above 0.5 than below it.

Note too that R^2 seems to have large variance, even in the case of $n = 250$. Thus in samples in which p/n is large, we should not take our sample's value of R^2 overly seriously.

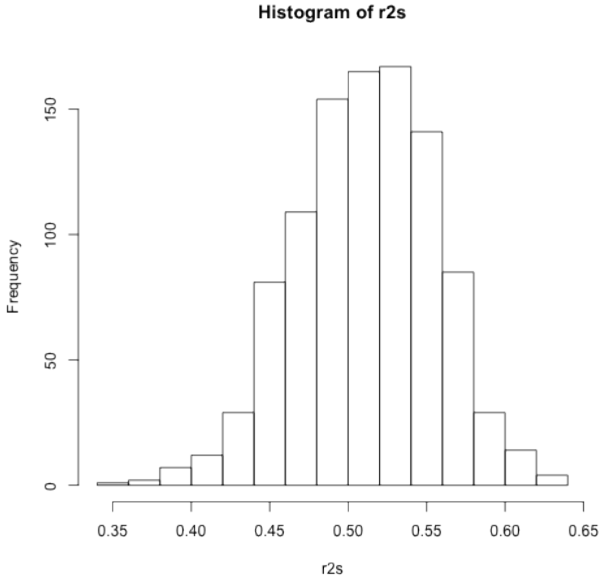

Figure 2.2: Plotted R^2 Values, n = 250

2.7.4 Adjusted-R^2

The adjusted-R^2 statistic is aimed at serving as a less biased version of the ordinary R^2. Its derivation is actually quite simple, though note that we do need to assume homoscedasticity.

Under the latter assumption, $Var(\epsilon) = \sigma^2$ in (2.59). Then the numerator in (2.63) is biased, which we know from (2.47) can be fixed by using the factor $1/(n-p-1)$ instead of $1/n$. Similarly, we know that the denominator will be unbiased if we divide by $1/(n-1)$ instead of $1/n$. Those changes do NOT make (2.63) unbiased; the ratio of two unbiased estimators is generally biased. However, the hope is that this new version of R^2, called *adjusted* R^2, will have less bias than the original.

We can explore this using the same simulation code as above. We simply change the line

```
smm$r.squared
```

to

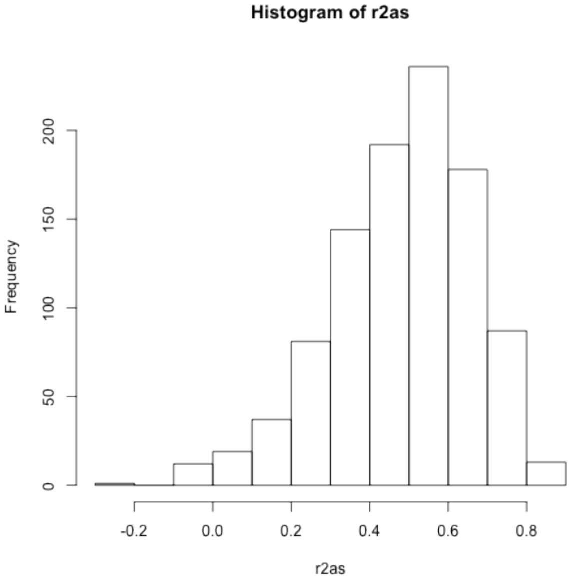

Figure 2.3: Plotted Adjusted R^2 Values, n = 25

```
smm$adj.r.squared
```

Rerunning **simr2(25,8,1000)**, we obtain the result shown in Figure 2.3. This is a good sign! The values are more or less centered around 0.5, as they should be (though there is still a considerable amount of variation).

2.7.5 The "Leaving-One-Out Method"

Our theme here in Section 2.7 has been assessing the predictive ability of our model, with the approach described so far being the R^2 measure. But recall that we have another measure: Section 1.9.3 introduced the concept of *cross-validation* for assessing predictive ability. We will now look at a variant of that method.

First, a quick review of cross-validation: Say we have n observations in our data set. With cross-validation, we randomly partition the data into a training set and a validation set, of k and $n - k$ observations, respectively. We fit our model to the training set, and use the result to predict in the validation set, and then see how well those predictions turned out.

Clearly there is an issue of the choice of k. If k is large, our validation set will be too small to obtain an accurate estimate of predictive ability. That is not a problem if k is small, but then we have a subtler problem: We are getting an estimate of strength of our model when constructed on k observations, but in the end we wish to use all n observations.

One solution is the Leaving One Out Method (LOOM). Here we set $k = n-1$, but apply the training/validation process to *all* possible $(n-1,1)$ partitions. The name alludes to the fact that LOOM repeatedly omits one observation, predicting it from fitting the model to the remaining observation. This gives us "the best of both worlds": We have n validation points, the best possible, and the training sets are of size $n-1$, i.e., nearly full-sized.

There is an added benefit that the same code to implement this method can be used to implement the *jackknife*. The latter is a *resampling* technique. To see what it does, let's look at a more general technique called the *bootstrap*, which is a method to empirically compute standard errors.

Say we wish to determine the standard error of an estimator $\widehat{\theta}$. We repeatedly take random samples of size k, with replacement, from our data set, and calculate $\widehat{\theta}$ on each of them. The resulting values form a sample from the distribution of $\widehat{\theta}$ (for sample size k). One can compute standard errors from this sample in various ways, e.g. simply by findiing their standard deviation.

The jackknife does this for $k = n-1$ (and sampling without replacement), and thus we can again approximate the sampling distribution of our estimator based on n data points.

2.7.5.1 The Code

Here is general code for Leaving-One-Out:

```
# Leaving-One-Out Method;

# use both for cross-validation and jackknife resampling

# arguments:

#    xydata: data, one row per observation, ”Y’ value last
#    regftn: regression function to apply to xydata and
#            resamples
#    postproc: function to apply to the resampling output
#    nsamp: number of leave-one-out resamples to process

# fits the specified regression model to each leave-one-out
# subsample, feeding the results into postproc()

# due to possibly large amount of computation

loom <- function(xydata,regftn,postproc,
     nsamp=nrow(xydata),...)
{
   xydata <- as.matrix(xydata)
   n <- nrow(xydata)
   if (nsamp == n) {
      toleaveout <- 1:n
   } else toleaveout <- sample(1:n,nsamp,replace=FALSE)
   jkout <- doleave(toleaveout,xydata,regftn)
   for (i in toleaveout) {
      jkout[[i]] <- regftn(xydata[-i,])
   }
   postproc(jkout,xydata,toleaveout)
}

doleave <- function(toleaveout,xyd,regftn) {
   if (is.null(xyd)) xyd <- xydata
   ntlo <- length(toleaveout)
   tmp <- list(length=ntlo)
   for (i in 1:ntlo) {
      lo <- toleaveout[i]
      tmp[[i]] <- regftn(xyd[-lo,])
   }
   tmp
}

lmregftn <- function(xydata) {
   ycol <- ncol(xydata)
   lm(xydata[,ycol] ~ xydata[,-ycol])
}

l1postproc <- function(lmouts,xydata,toleaveout) {
   l1s <- NULL
   ycol <- ncol(xydata)
   for (i in toleaveout) {
      bhat <- coef(lmouts[[i]])
      predval <- bhat %*% c(1,xydata[i,-ycol])
      realval <- xydata[i,ycol]
      l1s <- c(l1s,abs(realval - predval))
   }
   mean(l1s)
}
```

Let's look at the main arguments first:

- **xydata:** Our data set, in the same format as for instance our **xvallm()** function in Section 1.9.4: One observation per row, Y in the last column.
- **regftn:** The code is versatile, not just limited to the linear model, so the user specifies the regression function. The linear model case is handled by specifying **lmregftn**.
- **postproc:** After the leaving-one-out processing is done, the code has an R list, each element of which is the output of **regftn()**. The **postproc()** function, specified by the user, is applied to each of these elements. For instance, setting this to **l1postproc()** will result in computing the mean absolute prediction error.

The remaining arguments serve to ameliorate the major drawback of the Leaving-One-Out Method, which is large computation time:

- **nsamp:** Instead of leaving out each of observations $1, ..., n$, we do this only for a random sample of **nsamp** indices from that set.

Another possible source of speedup in the linear model case would be to use *matrix inverse update* methods, which we defer to Chapter 9.

2.7.5.2 Example: Bike-Sharing Data

To illustrate, let's look at the bike-sharing data again. To make it more interesting, let's load up the model with some more variables:

```
> shar$winter <- as.integer(shar$season == 1)
> shar$spring <- as.integer(shar$season == 2)
> shar$summer <- as.integer(shar$season == 3)
> shar$mon <- as.integer(shar$weekday == 1)
> shar$tue <- as.integer(shar$weekday == 2)
> shar$wed <- as.integer(shar$weekday == 3)
> shar$thu <- as.integer(shar$weekday == 4)
> shar$fri <- as.integer(shar$weekday == 5)
> shar$sat <- as.integer(shar$weekday == 6)
> shr <- as.matrix(shar[,c(10,17,6,18,12,13,14,20:28,15)])
```

Now try LOOM cross-validation:

```
> loom(shr,lmregftn,l1postproc)
[1] 383.3055
```

On average, we can predict ridership to about 383, when predicting *new* data. It's worthwhile comparing this to the same number obtained by repredicting the original data from itself, meaning

$$\frac{1}{n}\sum_{i=1}^{n} |Y_i - \widetilde{X}_i'\widehat{\beta}| \tag{2.65}$$

We can easily compute this number from the **lm()** output, as the latter includes the values of $Y_i - \widetilde{X}_i'\widehat{\beta}$, known as the *residuals*:

```
> lmout <- lm(shr[,17] ~ shr[,-17])
> mean(abs(lmout$residuals))
[1] 363.3149
```

This is our first concrete example of *overfitting*. The second number, about 363, is more optimistic than the cross-validdated one. The results would have likely been even worse with more variables. Again, this topic will be covered in depth in Chapter 9.

2.7.5.3 Another Use of loom(): the Jackknife

As explained in Section 2.7.5, **loom()** can also be used for *jackknife* purposes, which we will do here on the adjusted R^2 statistic. As we saw in Figure 2.3, this statistic can have considerable variation from one sample to another. But in that figure, we had the luxury of performing a simulation of many artificial samples. What can we do with our single sample of real data, to gauge how variable adjusted R^2 is in our setting?

The jackknife comes to the rescue! By repeatedly leaving one observation out, we can generate many adjusted R^2 values, amounting to a simulation from the sampling distribution of adjusted R^2. For this purpose, we set the **loom()** argument **postproc()** to the following:

```
> ar2postproc
function(lmouts,xydata,toleaveout) {
   r2s <- NULL
   for (lmout in lmouts) {
      s <- summary(lmout)
      r2s <- c(r2s,s$adj.r.squared)
   }
   r2s
}
```

Recall that, within **loom()** the function **regftn()** is called on each subsample of size $n - 1$. The function **postproc()** is then called on the results of these calls, which in this case are calls to **lm()**. Here **ar2postproc()** will then collect all the associated adjusted R^2 values.

Here is what we get for the above **lm()** analysis of the bike-sharing data:

```
> ar2out <- loom(shr,lmreg\footnote{,ar2postproc)
> hist(ar2out)
```

The results in Figure 2.4 are not too bad. There is actually rather little variation in the simulated adjusted R^2 values. This is not surprising, in that there is not much discrepancy between the ordinary and adjusted versions of R^2 in the full sample:

```
> summary(lmout)
...
Multiple R-squared:  0.7971,  Adj. R-squared:  0.7877
...
```

2.7.6 Other Measures

A number of other measures of predictive ability are in common use, notably *Mallows' C_p* and the *Akaike Information Criterion.* These will be treated in Chapter 9.

2.7.7 The Verdict

So, what is the verdict on the use of R^2 to assess the collective predictive ability of our predictor variables? There is a general consensus among data analysts that adjusted-R^2 is a better measure than R^2. And if one

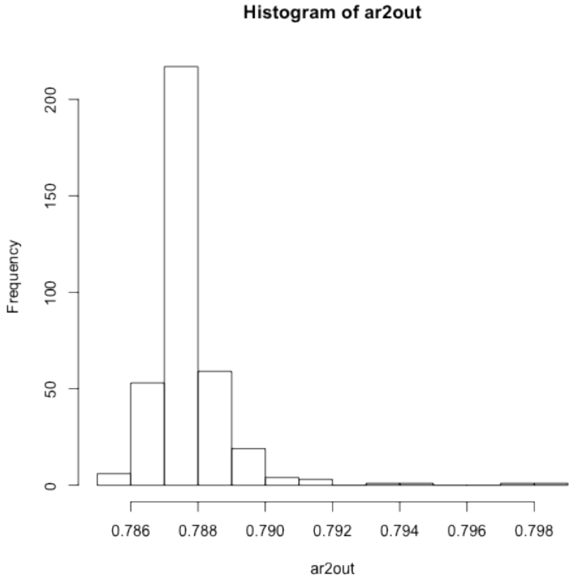

Figure 2.4: Plotted loom() 'R^2 Values, Bike-Sharing Data

observes a wide discrepancy between the two on a particular data set, this is a suggestion that we are overfitting.

On the other hand, one must keep in mind that R^2, like any other statistic, is a sample quantity subject to sampling variation. If close attention is to be paid to it, a standard error would be helpful, and would be obtainable via use of **loom()** as a jackknife (or by the bootstrap).[8]

R^2 is an appealing measure of predictive ability, as it is dimensionless, and comparable across diverse settings. But finer measures of predictive ability is obtainable via **loom()**, such as the mean absolute prediction error as shown here. R^2 involves *squared* prediction error, which accentuates the larger errors while giving smaller weight to the moderate ones, which we might consider a distortion.

[8] If we have m jackknifed versions of an estimator $\widehat{\theta}$ on a sample of size m, a standard error for full-sample version of $\widehat{\theta}$ is obtained as follows. Find the standard deviation of the m jackknifed values, and them multiply by $(m-1)/\sqrt{m}$.

2.8 Significance Testing vs. Confidence Intervals

"Sir Ronald [Fisher] has befuddled us, mesmerized us, and led us down the primrose path" — Paul Meehl, professor of psychology and the philosophy of science

When the concept of significance testing, especially the 5% value for α, was developed in the 1920s by Sir Ronald Fisher, many prominent statisticians opposed the idea — for good reason, as we'll see below. But Fisher was so influential that he prevailed, and thus significance testing became the core operation of statistics.

So, today significance testing is entrenched in the field, in spite of being widely recognized as faulty. Most modern statisticians understand this,[9] even if many continue to engage in the practice.[10]

The basic problem is that a significance test is answering the wrong question. Say in a regression analysis we are interested in the relation between $X^{(1)}$ and Y. Our test might have as null hypothesis

$$H_0 : \beta_1 = 0 \tag{2.66}$$

But we probably know *a priori* that there is at least *some* relation between the two variables; β_1 cannot be 0.000000000... to infinitely many decimal places. So we already know that H_0 is false.[11] The better approach is to form a confidence interval for β_1, so that we can gauge the *size* of β_1, i.e., the strength of the relation.

[9] This was eloquently stated in a guide to statistics prepared for the U.S. Supreme Court by two very prominent scholars (*Reference Guide on Statistics*, David Kaye and David Freedman, `http://www.fjc.gov/public/pdf.nsf/lookup/sciman02.pdf/$file/sciman02.pdf`: "Statistical significance depends on the p-value, and p-values depend on sample size. Therefore, a 'significant' effect could be small. Conversely, an effect that is 'not significant' could be large. By inquiring into the magnitude of an effect, courts can avoid being misled by p-values. To focus attention where it belongs — on the actual size of an effect and the reliability of the statistical analysis — interval estimates may be valuable. Seeing a plausible range of values for the quantity of interest helps describe the statistical uncertainty in the estimate."

[10] Many are forced to do so, e.g. to comply with government standards in pharmaceutical testing. My own approach in such situations is to quote the test results but then point out the problems, and present confidence intervals as well.

[11] A similar point holds for the F-test in **lm()** output, which tests that *all* the β_i are 0, i.e., $H_0 : \beta_1 = \beta_2 = \ldots \beta_p = 0$.

For instance, consider another UCI data set, Forest Cover, which involves a remote sensing project. The goal was to predict which one of seven types of ground cover exists in a certain inaccessible location, using variables that can be measured by satellite. One of the variables is Hillside Shade at Noon (HS12).

For this example, I restricted the data to Cover Types 1 and 2, and took a random subset of 1000 observations to keep the example manageable. The logistic model here is

$$P(\text{Cover Type 2}) = \frac{1}{1+e^{-(\beta_0+\beta_1 \text{ HS12})}} \tag{2.67}$$

Here is the **glm()** output, with column 8 being HS12 and column 56 being a dummy variable indicating Cover Type 2:

```
> glmout <-
     glm(f2512[,56] ~ f2512[,8],family=binomial)
> summary(glmout)
...
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.147856   0.634077  -3.387 0.000706 ***
f2512[, 8]   0.014102   0.002817   5.007 5.53e-07 ***
...
```

The triple-star result for β_1 would indicate that HS12 is a “very highly significant” predictor of cover type. Yet we see that $\widehat{\beta_1}$, 0.014102, is tiny. HS12 is in the 200+ range, with sample means 227.1 and 223.4 for the two cover types, differing only by 3.7. Multiplying the latter by 0.014102 gives a value of about 0.052, which is swamped in (2.67) by the $\widehat{\beta_0}$ term, -2.147856. In plain English: HS12 has almost no predictive power for Cover Type, yet the test declares it “very highly significant.”

The confidence interval for β_1 here is

$$0.014102 \pm 1.96 \cdot 0.002817 = (0.00858, 0.01962) \tag{2.68}$$

The fact that the interval excludes 0 is irrelevant. The real value of the interval here is that it shows that β_1 is quite small; even the right-hand end point is tiny.

This book’s preferred statistical inference method is confidence intervals, not significance tests.

2.9 Bibliographic Notes

For more on the Eickert-White approach to correct inference under heteroscedasticity, see (Zeileis, 2006).

2.10 Mathematical Complements

2.10.1 The Geometry of Linear Models

We'll use the notation of Section 2.3.2 here.

Since Since $\widehat{\beta}$ is the least-squares estimate, i.e. it minimizes $||D - Ab||$ over all b, then $A\widehat{\beta}$ is the closest vector in the column space of A to D. Thus the mapping

$$D \rightarrow A\widehat{\beta} \tag{2.69}$$

is a *projection.*

Since a projection forms a "right triangle," we have that

$$(A\widehat{\beta}, D - A\widehat{\beta}) = 0 \tag{2.70}$$

2.10.2 Unbiasedness of the Least-Squares Estimator

We will show that $\widehat{\beta}$ is conditionally unbiased,

$$E(\widehat{\beta} \mid X_1, ..., X_n) = \beta \tag{2.71}$$

This approach has the advantage of including the fixed-X case, and it also implies the unconditional case for random-X, since

$$E\widehat{\beta} = E[E(\widehat{\beta} \mid X_1, ..., X_n)] = E\beta = \beta \tag{2.72}$$

So let's derive (2.71). First note that, by definition of regression and the linear model,

$$E(Y \mid X) = \mu(X) = \widetilde{X}\ '\beta \tag{2.73}$$

Once again using the matrix partitioning technique as in (1.19), Equation (2.73) tells us that

$$E(D \mid A) = A\beta \tag{2.74}$$

where A and D are as in Section 2.3.2.

Now using (2.23) we have

$$\begin{aligned} E(\widehat{\beta} \mid X_1, ..., X_n) &= E[\widehat{\beta} \mid A] && (2.75) \\ &= (A'A)^{-1}A'E(D \mid A) && (2.76) \\ &= (A'A)^{-1}A'A\beta) && (2.77) \\ &= \beta && (2.78) \end{aligned}$$

thus showing that $\widehat{\beta}$ is unbiased.

2.10.3 Consistency of the Least-Squares Estimator

For technical reasons, it's easier to treat the random-X case. We'll make use of a famous theorem:

> *Strong Law of Large Numbers (SLLN): Say $W_1, W_2, ...$ are i.i.d. with common mean EW.* Then
>
> $$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} W_i = EW, \quad \text{with probability 1} \tag{2.79}$$

Armed with that fundamental theorem in probability theory, rewrite (2.23) as

$$\widehat{\beta} = \left(\frac{1}{n} A'A \right)^{-1} (\frac{1}{n} A'D) \tag{2.80}$$

To avoid clutter, we will not use the $\widetilde{X}$ notation here for augmenting with a 1 element at the top of a vector. Assume instead that the 1 is $X^{(1)}$.

By the SLLN, the (i,j) element of $\frac{1}{n}A'A$ converges as $n \rightarrow \infty$:

$$\frac{1}{n}(A'A)_{ij} = \frac{1}{n}\sum_{k=1}^{n} X_k^{(i)} X_k^{(j)} \rightarrow E[X^{(i)}X^{(j)}] = [E(XX')]_{ij} \tag{2.81}$$

i.e.,

$$\frac{1}{n}A'A \rightarrow E(XX') \tag{2.82}$$

The vector $A'D$ is a linear combination of the columns of A, with the coefficients of that linear combination being the elements of the vector D. Since the columns of A' are $X_k, \ k = 1, ..., n$, we then have

$$A'D = \sum_{k=1}^{n} Y_k X_k \tag{2.83}$$

and thus

$$\frac{1}{n}A'D \rightarrow E(YX) \tag{2.84}$$

The latter quantity is

$$\begin{aligned}
E\left[E(YX \mid X)\right] &= E\left[XE(Y \mid X)\right] && (2.85)\\
&= E\left[X(X'\beta)\right] && (2.86)\\
&= E\left[X(X'I\beta)\right] && (2.87)\\
&= E\left[(XX')I\beta)\right] && (2.88)\\
&= E(XX')\ \beta && (2.89)\\
& && (2.90)
\end{aligned}$$

So, we see that $\widehat{\beta}$ converges to

$$[E(XX')]^{-1}E(XY) = [E(XX')]^{-1}E(XX'\beta) = [E(XX')]^{-1}E(XX')\beta = \beta \tag{2.91}$$

2.10.4 Biased Nature of S

It was stated in Section 2.6.1 that S, even with the $n-1$ divisor, is a *biased* estimator of η, the population standrd deviation. We'll derive that here.

$$\begin{aligned} 0 &< Var(S) \quad &(2.92)\\ &= E(S^2) - (ES)^2 \quad &(2.93)\\ &= \eta^2 - (ES)^2 \quad &(2.94) \end{aligned}$$

since S^2 is an unbiased estimator of η^2. So,

$$ES < \eta \tag{2.95}$$

2.10.5 $\mu(X)$ and ϵ Are Uncorrelated

In Section (2.7.1), it was stated that $\mu(X)$ and ϵ are uncorrelated. This is easily shown. Since $E\epsilon = 0$ and $E(\epsilon|X) = 0$, we have

$$\begin{aligned} Cov[\mu(X), \epsilon) &= E\left[(\mu(X) - E\mu(X)) \cdot (\epsilon - E\epsilon)\right] \quad &(2.96)\\ &= E\left[(\mu(X) - E\mu(X)) \cdot (Y - \mu(X))\right] \quad &(2.97)\\ &= E\left[(\mu(X) - E\mu(X) \cdot E(Y - \mu(X)|X)\right] \quad &(2.98)\\ &= 0 \quad &(2.99) \end{aligned}$$

since

$$E(Y - \mu(X)|X) = \mu(X) - \mu(X) = 0 \tag{2.100}$$

2.10.6 Asymptotic $(p+1)$-Variate Normality of $\widehat{\beta}$

Here we show tht asymptotically $\widehat{\beta}$ has a $(p+1)$-vartiate normal distribution. We again assume the random-X setting, and as in Section 2.10.3, avoid clutter by incorporating the 1 element of $\widetilde{X}$ into X.

First, define the actual prediction errors we would have if we knew the true population value of β and were to predict the Y_i from the X_i,

$$\epsilon_i = Y_i - X_i'\beta \tag{2.101}$$

Let G denote the vector of the ϵ_i:

$$G = (\epsilon_1, ..., \epsilon_n)' \tag{2.102}$$

Then

$$D = A\beta + G \tag{2.103}$$

We will show that the distribution of $\sqrt{n}(\widehat{\beta} - \beta)$ converges to $(p+1)$-variate normal with mean 0.

Multiplying both sides of (2.103) by $(A'A)^{-1}A'$, we have

$$\widehat{\beta} = \beta + (A'A)^{-1}A'G \tag{2.104}$$

Thus

$$\sqrt{n}(\widehat{\beta} - \beta) = (A'A)^{-1}\sqrt{n}\ A'G \tag{2.105}$$

Using Slutsky's Theorem and (2.82), the right-hand side has the same asymptotic distribution as

$$[E(XX')]^{-1}\sqrt{n}\ (\frac{1}{n}A'G) \tag{2.106}$$

In the same reasoning that led to (2.84), we have that

$$A'G = \sum_{i=1}^{n} \epsilon_i X_i \tag{2.107}$$

This is a sum of i.i.d. terms with mean 0, so the CLT says that $\sqrt{n} \cdot (A'G/n)$ is asymptotically normal with mean 0 and covariance matrix equal to that of ϵX, where ϵ is a generic random variable having the distribution of the ϵ_i.

Putting this information together with (2.105), we have:

> $\widehat{\beta}$ is asymptotically $(p+1)$-variate normal with mean β and covariance matrix
>
> $$\frac{1}{n}\,[E(XX')]^{-1} Cov(\epsilon X)[E(XX')]^{-1} \tag{2.108}$$

2.10.7 Derivation of (3.14)

It is again convenient to treat the random-X case, building on the material in Section 2.10.6. Since (2.108) is so complex, let's simplify things by focusing on the $Cov(\epsilon X)$ factor in that equation. Since $E(\epsilon|X) = 0$, we have

$$\begin{aligned} Cov(\epsilon X) &= E[(\epsilon X)\ (\epsilon X)'] && (2.109)\\ &= E(\epsilon^2 XX') && (2.110) \end{aligned}$$

By the SLLN, this could be estimated by

$$\frac{1}{n}\sum_{i=1}^{n} \epsilon_i^2 X_i X_i' = \frac{1}{n}\sum_{i=1}^{n} (Y_i - X_i'\beta)^2 X_i X_i' \tag{2.111}$$

This is getting pretty close to what we need for (3.14), but the latter involves the $\widehat{\epsilon_i}$ instead of the ϵ_i:

$$\frac{1}{n}\sum_{i=1}^{n} \widehat{\epsilon}_i^2 X_i X_i' = \frac{1}{n}\sum_{i=1}^{n} (Y_i - X_i'\widehat{\beta})^2 X_i X_i' \tag{2.112}$$

But this can be resolved by various methods of advanced probability theory. For example, because $\widehat{\beta} \to \beta$, the Uniform Strong Law of Large Numbers says that under reasonable conditions, (2.112) has the same limit as (2.111).[12]

[12] See for instance *Asymptotic Theory of Statistics and Probability*, Anirban DasGupta, Springer, 2008. Roughly speaking, the conditions involve boundedness of the variables, which is very reasonable in practice. Adult human heights are bounded above by 8 feet, for instance.

Now let $B = \text{diag}(\widehat{\epsilon}_i, \ldots, \widehat{\epsilon}_n)$. Recalling that the columns of A' are the X_i, we see that $A'B$ is a $p \times n$ matrix whose i^{th} column is $\widehat{\epsilon}_i^2 X_i$. In partitioned matrix form, then

$$A'B = (\widehat{\epsilon}_1^2 X_1, \ldots \widehat{\epsilon}_n^2 X_n) \tag{2.113}$$

But also in partitioned matrix form,

$$A = \left(\begin{array}{c} X_1' \\ \ldots \\ X_n' \end{array} \right) \tag{2.114}$$

Taking the product in the last two equations, we obtain

$$A'BA = \sum_{i=1}^{n} \widehat{\epsilon}_i^2 X_i X_i' \tag{2.115}$$

So, in (2.108), we can replace $Cov(\epsilon S)$ by $A'BA/n$. From previous calculations, we know that $E(XX')$ can be replaced by $A'A/n$. So, we can approximate (2.108) by

$$\left[(\frac{1}{n}A'A)^{-1} \ (\frac{1}{n}A'BA) \ (\frac{1}{n}A'A)^{-1} \right] /n = (A'A)^{-1} \ A'BA \ (A'A)^{-1} \tag{2.116}$$

The right-hand side is (3.14)!

2.10.8 Distortion Due to Transformation

Consider this famous inequality:

> *Jensen's Inequality:* Suppose h is a convex function,[13] and V is a random variable for which the expected values in (2.117) exist. Then
>
> $$E[h(V)] \geq h(EV) \tag{2.117}$$

[13] This is "concave up," in the calculus sense.

In our context, h is our transformation in Section 3.3.7, and the $E()$ are conditional means, i.e., regression functions. In the case of the log transform (and the square-root transform), h is concave-down, so the sense of the inequality is reversed:

$$E[\ln Y|X = t] \leq \ln(E(Y|X = t) \tag{2.118}$$

Since equality will hold only in trivial cases, we see that the regression function of $\ln Y$ will be smaller than the log of the regression function of Y.

Say we assume that

$$E(Y|X = t) = e^{\beta_0 + \beta_1 t} \tag{2.119}$$

and reason that this implies that a linear model would be reasonable for $\ln Y$:

$$E(\ln Y|X = t) = \beta_0 + \beta_1 t \tag{2.120}$$

Jensen's Inequality tells us that such reasoning may be risky. In fact, if we are in a substantially heteroscedastic setting (for Y, not $\ln Y$), the discrepancy between the two sides of (2.118) could vary a lot with t, potentially producing quite a bit of distortion to the shape of the regression curve. This follows from a result of Robert Becker,[14] who expresses the difference between the left- and right-hand sides of (2.117) in terms of $Var(V)$.

[14] *The Variance Drain and Jensen's Inequality*, CAEPR Working Paper 2012-004, Indiana University, 2012.

Chapter 3

The Assumptions in Practice

This chapter will take a practical look at the classical assumptions of linear regression models:

(a) Linearity:

$$E(Y \mid \widetilde{X} = \widetilde{t}) = \widetilde{t}'\beta \tag{3.1}$$

(b) Normality: The conditional distribution of Y given X is normal.

(c) Independence: The data (X_i, Y_i) are independent across i.

(d) Homoscedasticity:

$$Var(Y \mid X = t) \tag{3.2}$$

is constant in t.

Verifying assumption (a), and dealing with substantial departures from it, is the subject of an entire chapter, Chapter 6. So, this chapter will focus on assumptions (b)-(d).

3.1 Normality Assumption

We already discussed (b) in Section 2.6.2, but the topic deserves further comment. First, let's review what was found before.

Neither normality nor homoscedasticity is needed; $\widehat{\beta}$ is unbiased, consistent and asymptotically normal without those assumptions. Standard statistical inference procedures, however, assume homoscedasticity. We'll return to the latter issue in Section 3.3. But for now, let's concentrate on the normality assumption. Retaining the homoscedasticity assumption for the moment, we found in the last chapter that:

> The conditional distribution of the least-squares estimator $\widehat{\beta}$, given A, is approximately multivariate normal distributed with mean β and approximate covariance matrix
>
> $$s^2(A'A)^{-1} \tag{3.3}$$
>
> Thus the standard error of $\widehat{\beta}_j$ is the square root of element j of this matrix (counting the top-left element as being in row 0, column 0).
>
> Similarly, suppose we are interested in some linear combination $\lambda'\beta$ of the elements of β, estimating it by $\lambda'\widehat{\beta}$. The standard error is the square root of
>
> $$s^2\lambda'(A'A)^{-1}\lambda \tag{3.4}$$

The reader should not overlook the word *asymptotic* in the above. Without assumption (a) above, our inference procedures (confidence intervals, significance tests) are valid, but only approximately. On the other hand, the reader should be cautioned (as in Section 2.6.1) that so-called "exact" inference methods, based on the Student-t distribution and so on, are themselves only approximate, since true normal distributions rarely if ever exist in real life.

In other words:

> We must live with approximations one way or the other, and the end result is that the normality assumption is not very important.

3.2 Independence Assumption — Don't Overlook It

Statistics books tend to blithely say things like "Assume the data are independent and identically distributed (i.i.d.)," without giving any comment to (i) how they might be nonindependent and (ii) what the consequences are of using standard statstical methods on nonindependent data. Let's take a closer look at this.

3.2.1 Estimation of a Single Mean

Note the denominator $S/\sqrt{n}$ in (2.35). This is the standard error of $\overline{W}$, i.e. the estimated standard deviation of that quantity. That in turn comes from a derivation you may recall from statistics courses,

$$\begin{aligned} Var(\overline{W}) &= \frac{1}{n^2} Var(\sum_{i=1}^{n} W_i) & (3.5) \\ &= \frac{1}{n^2} \sum_{i=1}^{n} Var(W_i)) & (3.6) \end{aligned}$$

and so on.

In going from the second equation to the third, we are making use of the usual assumption that the W_i are independent. But suppose the W_i are correlated. Then the correct equation is

$$Var(\sum_{i=1}^{n} W_i) = \sum_{i=1}^{n} Var(W_i) + 2 \sum_{1 \leq i < j \leq n} Cov(W_i, W_j) \tag{3.7}$$

It is often the case that our data are positively correlated. Many data sets, for instance, consist of multiple measurements on the same person, say 10 blood pressure readings for each of 100 people. In such cases, the covariance terms in (3.7) will be positive, and (3.5) will yield too low a value. Thus the denominator in (2.35) will be smaller than it should be. That means that our confidence interval (2.37) will be too small (as will be p-values), a serious problem in terms of our ability to do valid inference.

Here is the intuition behind this: Although we have 1000 blood pressure readings, the positive intra-person correlation means that there is some degree of repetition in our data. Thus we don't have "1000 observations worth" of data, i.e. our effective n is less than 1000. Hence our confidence interval, computed using $n = 1000$, is overly optimistic.

Note that $\overline{W}$ will still be an unbiased and consistent estimate of ν. In other words, $\overline{W}$ is still useful, even if inference procedures computed from it may be suspect.

3.2.2 Estimation of Linear Regression Coefficients

All of this applies to inference on regression coefficients as well. If our data is correlated, i.e. rows within (A, D) are not independent, then (**??**) will be incorrect, because the off-diagnonal elements won't be 0s. And if they are positive, (**??**) will be "too small," and the same will be true for (3.3). Again, the result will be that our confidence intervals and p-values will be too small, i.e. overly optimistic. In such a situation, then our $\widehat{\beta}$ will still be useful, but our inference procedures will be suspect.

3.2.3 What Can Be Done?

This is a difficult problem. Some possibilities are:

- Simply note the dependency problem, e.g. in our report to a client, and state that though our estimates are valid (in the sense of statistical consistency), we don't have reliable standard errors.

- Somehow model the dependency, i.e. the off-diagonal elements of $(A'A)^{-1}$.

- Collapse the data in some way to achieve independence.

An example of this last point is presented in the next section.

3.2.4 Example: MovieLens Data

The MovieLens data (`http://grouplens.org/`) consists of ratings of various movies by various users. The 100K version, which we'll analyze here, consists of columns User ID, Movie ID, Rating and Timestamp. There is one row per rating. If a user has rated, say eight movies, then he/she will have eight rows in the data matrix. Of course, most users have not rated most movies.

Let Y_{ij} denote the ratings of user i, $j = 1, 2, ..., N_i$, where N_i is the number of movies rated by this user. We are not taking into account which movies the user rates here, just analyzing general user behavior. We are treating the users as a random sample from a conceptual population of all potential users.

As with the blood pressure example above, for fixed i, the Y_{ij} are not independent, since they come from the same user. Some users tend to give harsher ratings, others tend to give favorable ones. But we can form

$$T_i = \frac{1}{N_i} \sum_{j=1}^{N_i} Y_{ij} \tag{3.8}$$

the average rating given by user i, and we have *independent* random variables. And, if we treat the N_i as random too, and i.i.d., then the T_i are i.i.d., enabling standard statistical analyses.

For instance, the MovieLens data include a few demographic variables for the users, and we can run the model, say,

$$\text{mean rating} = \beta_0 + \beta_1 \text{ age} + \beta_2 \text{ gender} \tag{3.9}$$

and then pose questions such as “Do older people tend to give lower ratings?”

3.3 Dropping the Homoscedasticity Assumption

For an example of problems with the homoscedasticity assumption, again consider weight and height. It is intuitive that tall people have more variation in weight than do short people, for instance. We can confirm that in our baseball player data. Let’s find the sample standard deviations for each height group (restricting to the groups with over 50 observations), seen in Section 1.5.2):

```
> m70 <- mlb[mlb$Height >= 70 & mlb$Height <= 77,]
> sds <- tapply(m70$Weight,m70$Height,sd)
> plot(70:77,sds)
```

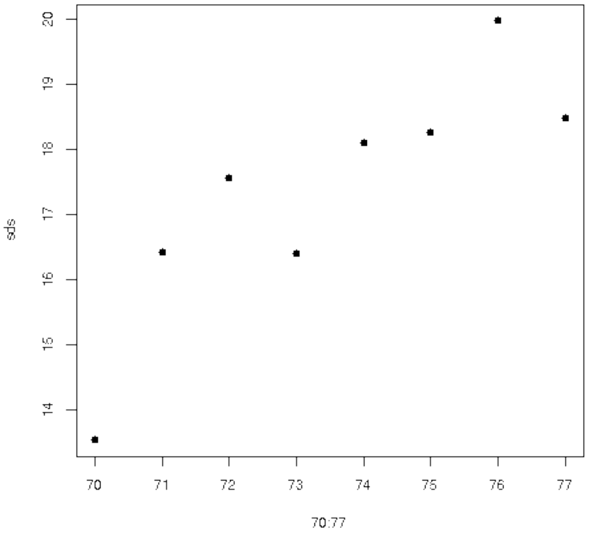

Figure 3.1: Standard Deviations of Weight, By Height Group

The result is shown in Figure 3.1. The upward trend is clearly visible, and thus the homoscedasticity assumption is not reasonable.

(2.23) is called the *ordinary least-squares* (OLS) estimator of β, in contrast to *weighted least-sqaures* (WLS), a weighted version to be discussed shortly. Statistical inference on β using OLS is usually based on (2.46), which is in turn based on the homoscedasticity assumption — that (2.26) is constant in t. Yet that assumption is rarely if ever valid.

Given the inevitable nonconstancy of (2.26), there are questions that must be raised:

- Do departures from constancy of (2.26) in t substantially impact the validity of statistical inference procedures that are based on (2.46)?
- Can we somehow estimate the function $\sigma^2(t)$, and then use that information to perform a WLS analysis?
- Can we somehow modify (2.46) for the heteroscedastic case?

These points will be addressed in this section.

3.3.1 Robustness of the Homoscedasticity Assumption

In statistics parlance, we ask, "Is classical inference on β *robust* to the homoscedasticity assumption, meaning that there is not much effect on the validity of our inference procedures (confidence intervals, significance tests) unless the setting is quite profoundly heteroscedastic?" We can explore this idea via simulation.

Let's investigate settings in which

$$\sigma(t) = |\mu(t)|^q \tag{3.10}$$

where q is a parameter to vary in the investigation. This includes several important cases:

- $q = 0$: Homoscedasticity.
- $q = 0.5$: Conditional distribution of Y given X is Poisson.
- $q = 1$: Conditional distribution of Y given X is exponential.

Here is the code:

```
simhet <- function(n,p,nreps,sdpow) {
   bh1s <- vector(length=nreps)
   ses <- vector(length=nreps)
   for (i in 1:nreps) {
      x <- matrix(rnorm(n*p),ncol=p)
      meany <- x %*% rep(1,p)
      sds <- abs(meany)^sdpow
      y <- meany + rnorm(n,sd=sds)
      lmout <- lm(y ~ x)
      bh1s[i] <- coef(lmout)[2]
      ses[i] <- sqrt(vcov(lmout)[2,2])
   }
   mean(abs(bh1s - 1.0) < 1.96*ses)
}
```

The simulation finds the true confidence level (providing **nreps** is set to a large value) corresponding to a nominal 95% confidence interval. Table 3.1 shows the results of a few runs, all with **nreps** set to 100000. We see that there is indeed an effect on the true confidence level.

n	p	q	conf. lvl.
100	5	0.0	0.94683
100	5	0.5	0.92359
100	5	1.0	0.90203
100	5	1.5	0.87889
100	5	2.0	0.86129

Table 3.1: Heteroscedasticity Effect Simulation

3.3.2 Weighted Least Squares

If one knows the function $\sigma^2(t)$ (at least up to a constant multiple), one can perform a weighted least-squares (WLS) analysis. Here, instead of minimizing (2.12), one minimizes

$$\frac{1}{n} \sum_{i=1}^{n} \frac{1}{w_i} (Y_i - \widetilde{X_i}' b)^2 \tag{3.11}$$

(without the $1/n$ factor, of course), where

$$w_i = \sigma^2(X_i) \tag{3.12}$$

Just as one can show that in the homoscedastic case, OLS gives the optimal (minimum-variance unbiased) estimator, the same is true for WLS in heteroscedastic settings, provided we know the function $\sigma^2(t)$.[1]

R's **lm()** function has an optional **weights** argument for specifying the w_i. But needless to say, this situation is not common. To illustrate this point, consider the classical inference procedure for a single mean, reviewed in Section 2.6.1. If we don't know the population mean ν, we are even less likely to know the population variance η^2. The same holds in the regression context, concerning conditional means and conditional variances.

One option would be to estimate the function $\sigma(t)$ using nonparametric regression techniques.[2] For instance, we can use our k-NN function **knnest()**

[1] Mathematically, one is essentially transforming the original heteroscedastic problem to a homoscedastic one by replacing Y and X by $Y/\sigma(X)$ and $X/\sigma(X)$, respectively.

[2] First proposed in R. Rose, *Nonparametric Estimation of Weights in Least-Squares*

of Section 1.7.4, with the default for **regestpts** and **applyf = var**. Let's run the analysis with and then without weights:

```
> w <- knnest(mlb[,c(4,6,5)],mlb[,c(4,6)],20,
   scalefirst=TRUE, applyf=var)
> summary(lm(mlb$Weight ~ mlb$Height+mlb$Age,
   weights=w))
...
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -188.8282    19.1967  -9.836  < 2e-16 ***
mlb$Height       4.9473     0.2490  19.872  < 2e-16 ***
mlb$Age          0.8996     0.1402   6.415 2.16e-10 ***
...
> summary(lm(mlb$Weight ~ mlb$Height+mlb$Age))
...
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -187.6382    17.9447  -10.46  < 2e-16 ***
mlb$Height       4.9236     0.2344   21.00  < 2e-16 ***
mlb$Age          0.9115     0.1257    7.25 8.25e-13 ***
...
```

The weighted analysis, the "true" one (albeit with the weights being only approximate), did give slightly different results than those of OLS. The standard error for the Age coefficient, for instance, was about 12% larger with WLS. This may seem small (and is small), but a 12% difference will have a large effect on the true confidence level. Consider this computation:

```
> 1 - 2*pnorm(-0.88*1.96)
[1] 0.9154365
```

In other words, a nominal 95% confidence interval only has confidence level at about 91.5%. Or, a nominal p-value of 5% is actually 8.5%. Use of the estimated weights makes a difference, with impacts on our statistical inference.

On the other hand, we used a rather small value of **k** here, and there is no clear way to choose it.

Regression Analysis, PhD dissertation, University of California, Davis, 1978, later studied extensively in various papers by Raymond Carroll.

3.3.3 A Procedure for Valid Inference

In principle, the delta method (Appendix **??**) could be used to not only show that $\widehat{\beta}$ is asymptotically normal, but also find its asymptotic covariance matrix without assuming homoscedasticity. We would then have available standard errors for the $\widehat{\beta}_i$ and so on.

However, the matrix differentiation needed to use the delta method would be far too complicated. Fortunately, though, there exists a rather simple procedure, originally developed by Eickert[3] and later refined by White and others,[4] for finding valid asymptotic standard errors for $\widehat{\beta}$ in the heteroscedastic case. This section will present the methodology, and test it on data.

3.3.4 The Methodology

Let's first derive the correct expression for $Cov(\widehat{\beta}|A)$ without the homoscedasticity assumption. Using our properties of covariance, this is

$$Cov(\widehat{\beta} \mid A) = (A'A)^{-1}A' \text{ diag}(\sigma^2(X_1), ..., \sigma^2(X_n) \ A(A'A)^{-1} \tag{3.13}$$

where diag$(a_1, ..., a_k)$ denotes the matrix with the a_i on the diagonal and 0s elsewhere.

Rather unwieldy, but the real problem is that we don't know the $\sigma(X_n)$. However, the situation is more hopeful than it looks. It can be proven (as outlined in Section 2.10.7) that:

> In the heteroscedastic case, the approximate covariance matrix of $\widehat{\beta}$ given A is
>
> $$(A'A)^{-1}A' \text{ diag}(\widehat{\epsilon}_1^2, ..., \widehat{\epsilon}_n^2) \ A(A'A)^{-1} \tag{3.14}$$

[3] Friedhelm Eickert, 1967, "Limit Theorems for Regression with Unequal and Dependent Errors," *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, 5982.

[4] For example, Halbert White, (1980), "A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity," *Econometrica*, 1980, 817838; James MacKinnon and Halbert White, "Some Heteroskedastic-Consistent Covariance Matrix Estimators with Improved Finite Sample Properties," *Journal of Econometrics*, 1985, 305325.

n	p	q	conf. lvl.
100	5	0.0	0.95176
100	5	0.5	0.94928
100	5	1.0	0.94910
100	5	1.5	0.95001
100	5	2.0	0.95283

Table 3.2: Heteroscedasticity Correction Simulation

where

$$\widehat{\epsilon}_i = Y_i - \widetilde{X}_i' \widehat{\beta} \tag{3.15}$$

Again, the expression in (3.14) is rather unwieldy, but it is easy to program, and most important, we're in business! We can now conduct statistical inference even in the heteroscedastic case.

Code implementing (3.14) is available in R's **car** and **sandwich** packages, as the functions **hccm()** and **vcovHC()**, respectively. (These functions also offer various refinements of the method.) These functions are drop-in replacements to the standard **vcov()**.

3.3.5 Simulation Test

Let's see if it works, at least in the small simulation experiment in Section 3.3.1. We use the same code as before, simply replacing the call to **vcov()** by one to **vcovHC()**. The results, shown in Table 3.2, are excellent.

3.3.6 Example: Bike-Sharing Data

In Section 2.6.3, we found that the standard error for $\mu(75,0,1)-\mu(62,0,1)$ was about 47.16. Let's get a more accurate standard error, that does not assume homoscedasticity:

```
> sqrt(t(lamb) %*% vcovHC(lmout) %*% lamb)
            [,1]
[1,]  44.0471
```

Not a very large difference in this case, but still of interest.

3.3.7 Variance-Stabilizing Transformations

Most classical treatments of regression analysis devote a substantial amount of space to *transformations* of the data. For instance, one might replace Y by $\ln Y$, and possibly apply the log to the predictors as well. There are several reasons why this might be done:

(a) The distribution of Y given X may be skewed, and applying the log may make it more symmetric, thus more normal-like.

(b) Log models may have some meaning relevant to the area of application, such as *elasticity* models in economics.

(c) Applying the log may convert a heteroscedastic setting to one that is close to homoscedastic.

One of the themes of this chapter has been that the normality assumption is not of much practical importance, which would indicate that Reason (a) above may not so useful. Reason (b) is domain-specific, and thus outside the scope of this book. But Reason (c) relates directly our current discussion on heteroscedasticity. Here is how transformations come into play.

Recall that the delta method (Appendix **??**) says, roughly, that if the random variable W is approximately normal with a small *coefficient of variation* (ratio of standard deviation to mean), and g is a smooth function, then the new random variable $g(W)$ is also approximately normal, with mean $g(EW)$ and variance

$$[g'(EW)]^2 Var(W) \tag{3.16}$$

Let's consider that in the context of (3.10). Assuming that the regression function is always positive, (3.10) reduces to

$$\sigma(t) = \mu^q(t) \tag{3.17}$$

Now, suppose (3.17) holds with $q = 1$. Take $g(t) = \ln(t)$. Then since

$$\frac{d}{dt} \ln t = \frac{1}{t} \tag{3.18}$$

we see that (3.16) becomes

$$\frac{1}{\mu^2(t)} \cdot \mu^2(t) = 1 \tag{3.19}$$

In other words $Var(\ln Y \mid X = t)$ is approximately 1, and we are back to the homoscedastic case. Similarly, if $q = 0.5$, then setting $g(t) = \sqrt{t}$ would give us approximate homoscedasticity.

However, this method has real drawbacks: Distortion of the model, difficulty interpreting the coefficients and so on.

Let's look at a very simple model that illustrates the distortion issue. (It is further explored in Section 2.10.8.) Suppose X takes on the values 1 and 2. Given $X = 1$, Y is either 2 or 1/2, with probability 1/2 each. If $X = 2$, then Y is either 4 or 1/4, with probability 1/2 each. Let $U = \log_2 Y$.

Let μ_Y and μ_U denote the regression functions of Y and U on X. An advantage of this very simple model is that, since X takes on only two values, both of these regression functions are linear.

Then

$$\mu_U(1) = 0.5 \cdot 1 + 0.5 \cdot (-1) = 0 \tag{3.20}$$

and similarly $\mu_U(2) = 0$ as well.

So, look at what we have. There is no relation between U and X at all! Yet the relation between Y and X is quite substantial. The transformation has destroyed the latter relation.

Of course, this example is contrived, and one can construct examples with the opposite effect. Nevertheless, it shows that a log transformation can indeed bring about considerable distortion. This is to be expected in a sense, since the log function flattens out as we move to the right. Indeed, the U.S. Food and Drug Administration once recommended against using transformations.[5]

[5] Quoted in The Log Transformation Is Special, *Statistics in Medicine*, Oliver Keene, 1995, 811-819. That author takes the opposite point of view.

3.3.8 The Verdict

While the examples here do not constitute a research study (the reader is encouraged to try the code in other settings, simulated and real), an overall theme is suggested.

In principle, WLS provides more efficient estimates and correct statistical inference. What are the implications?

If our goal is Prediction, then forming correct standard errors is typically of secondary interest, if at all. And unless there is really strong variation in the proper weights, having efficient estimates is not so important. In other words, for Prediction, OLS may be fine.

The picture changes if the goal is Description, in which case correct standard errors may be important. For this, given that the method of Section 3.3.3, is now commonly available in statistical software packages (albeit not featured), this is likely to be the best way to cope with heteroscedasticity.

3.4 Bibliographic Notes

For more on the Eickert-White approach to correct inference under heteroscedasticity, see (Zeileis, 2006).

Chapter 4

Nonlinear Models

Consider our bike-sharing data (e.g./ Section 1.12.2). Suppose we have several years of data. On the assumption that ridership trends are seasonal, and that there is no other time trend (e.g. no long-term growth in the program), then there would be a periodic relation between ridership R and G, the day in our data; here G would take the values 1, 2, 3, ..., with the top value being, say, $3 \times 365 = 1095$ for three consecutive years of data.[1] Assuming that we have no other predictors, we might try fitting the model with a sine term:

$$\text{mean } R = \beta_0 + \beta_1 \sin(2\pi \cdot G/365) \tag{4.1}$$

Just as adding a quadratic term didn't change the linearity of our model in Section 1.11.1, the model (4.1) is linear too. In the notation of Section 2.3.2), as long as we can write our model as

$$\text{mean } D = A \ \beta \tag{4.2}$$

then by definition the model is linear. In the bike data model above, A would be

$$A = \left(\begin{array}{cc} 1 & \sin(2\pi \cdot 1/365) \\ 1 & \sin(2\pi \cdot 2/365) \\ ... & \\ 1 & \sin(2\pi \cdot 1095/365) \end{array} \right) \tag{4.3}$$

[1] We'll ignore the issue of leap years here, to keep things simple.

But in this example, we have a known period, 365. In some other periodic setting, the period might be unknown, and would need to be estimated from our data. Our model might be, say,

$$\text{mean } Y = \beta_0 + \beta_1 \sin(2\pi \cdot X/\beta_2) \tag{4.4}$$

where β_2 is the unknown period. This does not correspond to (4.2). The model is still parametric, but is nonlinear.

Nonlinear parametric modeling, then, is the topic of this chapter. We'll develop procedures for computing least squares estimates, and forming confidence intervals and p-values, again without assuming homoscedasticity. The bulk of the chapter will be devoted to the *Generalized Linear Model* (GLM), which is a widely-used broad class of nonlinear regression models. Two important special cases of the GLM will be the *logistic* model introduced briefly in Section 1.12.2, and *Poisson regression.*

4.1 Example: Enzyme Kinetics Model

Data for the famous Michaelis-Menten enzyme kinetics model is available in the **nlstools** package on CRAN. For the data set **vmkm**, we predict the reaction rate V from substrate concentration S. The model used was suggested by theoretical considerations to be

$$E(V \mid S = t) = \frac{\beta_1 t}{\beta_2 + t} \tag{4.5}$$

In the second data set, **vmkmki**,[2] an addiitonal predictor I, inhibitor concentration, was added, with the model being

$$E(V \mid S = t, I = u) = \frac{\beta_1 t}{t + \beta_2 \ (1 + u/\beta_3)} \tag{4.6}$$

We'll fit the model using R's **nls()** function:

```
> library(nlstools)
> data(vmkmki)
> regftn <- function(t,u,b1,b2,b3)
     b1 * t / (t + b2 * (1 + u/b3))
```

[2] There were 72 observation in this data, but the last 12 appear to be anomalous (gradient 0 in all elements), and thus were excluded.

All nonlinear least-squares algorithms are iterative: We make an initial guess at the least-squares estimate, and from that, use the data to update the guess. Then we update the update, and so on, iterating until the guesses converge. In **nls()**, we specify the initial guess for the parameters, using the **start** argument, an R list.[3] Let's set that up, and then run the analysis:

```
> bstart <- list(b1=1,b2=1, b3=1)
```

The values 1 here were arbitrary, not informed guesses at all. Domain expertise can be helpful.

```
> z <- nls(v ~ regftn(S,I,b1,b2,b3),data=vmkmki,
   start=list(b1=1,b2=1, b3=1))
> z
Nonlinear regression model
  model: v ~ regftn(S, I, b1, b2, b3)
   data: vmkmki
   b1    b2    b3
18.06 15.21 22.28
 residual sum-of-squares: 177.3

Number of iterations to convergence: 11
Achieved convergence tolerance: 4.951e-06
```

So, $\widehat{\beta_1} = 18.06$ etc.

We can apply **summary()**, **coef()** and **vcov()** to the output of **nls()**, just as we did earlier with **lm()**. For example, here is the approximate covariance matrix of the coefficient vector:

```
> vcov(z)
          b1        b2         b3
b1 0.4786776  1.374961  0.8930431
b2 1.3749612  7.568837 11.1332821
b3 0.8930431 11.133282 29.1363366
```

This assumes homoscedasticity. Under that assumption, an approximate 95% confidence interval for β_1 would be

$$18.06 \pm 1.96\ \sqrt{0.4786776} \tag{4.7}$$

[3] This also gives the code a chance to learn the names of the parameters, needed for computation of derivatives.

One can use the approach in Section 3.3.3 to adapt **nls()** to the heteroscedastic case, and we will do so in Section 4.2.2.

4.2 Least-Squares Computation

A point made in Section 1.3 was that the regression function, i.e. the conditional mean, is the optimal predictor function, minimizing mean squared prediction error. This still holds in the nonlinear (and even nonparametric) case. The problem is that in the nonlinear setting, the least-squares estimator does not have a nice, closed-form solution like (2.23) for the linear case. Let's see how we can compute the solution through iterative approximation.

4.2.1 The Gauss-Newton Method

Denote the nonlinear model by

$$E(Y \mid X = t) = g(t, \beta) \tag{4.8}$$

where both t and β are possibly vector-valued. In (4.5), for instance, t is a scalar but β is a vector. The least-squares estimate $\widehat{\beta}$ is the value of b that minimizes

$$\sum_{i=1}^{n} [Y_i - g(X_i, b)]^2 \tag{4.9}$$

Many methods exist to minimize (4.9), most of which involve derivatives with respect to b. (The reason for the plural *derivatives* is that there is a partial derivative for each of the elements of b.)

The best intuitive explanation of derivative-based methods, which will also prove useful in a somewhat different context later in this chapter, is to set up a Taylor series approximation for $g(X_i, b)$ (Appendix **??**):

$$g(X_i, b) \approx g(X_i, \widehat{\widehat{\beta}}) + h(X_i, \widehat{\beta})'(b - \widehat{\beta}) \tag{4.10}$$

where $h(X_i, b)$ is the derivative vector of $g(X_i, b)$ with respect to b, and the prime symbol, as usual, means matrix transpose (not a derivative). The

(4.13)	(2.12)
$Y_i - g(X_i, b_{k-1}) + h(X_i, b_{k-1})' b_{k-1}$	Y_i
$h(X_i, b_{k-1})$	$\tilde{X}_i$

Table 4.1:

value of $\widehat{\beta}$, is of course yet unknown, but let's put that matter aside for now. Then (4.9) is approximately

$$\sum_{i=1}^{n} [Y_i - g(X_i, \widehat{\beta}) + h(X_i, \widehat{\beta})' \widehat{\beta} - h(X_i, \widehat{\beta})' \ b]^2 \tag{4.11}$$

At iteration k we take our previous iteration b_{k-1} to be an approximation to $\widehat{\beta}$, and make that substitution in (4.11), yielding

$$\sum_{i=1}^{n} [Y_i - g(X_i, b_{k-1}) + h(X_i, b_{k-1})' b_{k-1} - h(X_i, b_{k-1})' \ b]^2 \tag{4.12}$$

Our b_k is then the value that minimizes (4.12) over all possible values of b. But why is that minimization any easier than minimizing (4.9)? To see why, write (4.12) as

$$\sum_{i=1}^{n} [\underbrace{Y_i - g(X_i, b_{k-1}) + h(X_i, b_{k-1})' b_{k-1}} - h(X_i, b_{k-1})' \ b]^2 \tag{4.13}$$

This should look familiar. It has exactly the same form as (2.12), with the correspondences shown in Table 4.1. In other words, what we have in (4.13) is a *linear* regression problem!

In other words, we can find the minimizing b in (4.13) using **lm()**. There is one small adjustment to be made, though. Recall that in (2.12), the quantity $\widetilde{X}_i$ includes a 1 term (Section 2.1), i.e. the first column of A in (2.13) consists of all 1s. That is not the case in Table 4.1 (second row, first column), which we need to indicate in our **lm()** call. We can do this via specifying "-1" in the formula part of the call.

Another issue is the computation of $h()$. Instead of burdening the user with with this, it is typical to compute $h()$ using numerical approximation, e.g. using R's **numericDeriv()** function or the **numDeriv** package.

4.2.2 Eickert-White Asymptotic Standard Errors

As noted, **nls()** assumes homoscedasticity, which generally is a poor assumption (Section 2.4.2). It would be nice, then, to apply the Eickert-White method (Section 3.3.3). Actually, it is remarkably easy to adapt that method to the nonlinear computation above.

The key is to note the linear approximation (4.2.1). One way to look at this is that it has already set things up for us to use the delta method, which uses a linear approximation. Thus we can apply Eickert-White to the **lm()** output, say using **vcovHC()**, as in Section 3.3.4.

Below is code along these lines. It requires the user to run **nlsLM()**, an alternate version of **nls()** in the CRAN package **minpack.lm**.[4]

```
library(minpack.lm)
library(sandwich)

# uses output of nlsLM() of the minpack.lm package
# to get an asymptotic covariance matrix without
# assuming homoscedasticity

# arguments:
#
#     nlslmout: return value from nlsLM()
#
# value: approximate covariance matrix for the
#        estimated parameter vector

nlsvcovhc <- function(nlslmout) {
   # notation: g(t,b) is the regression model,
   # where x is the vector of variables for a
   # given observation; b is the estimated parameter
   # vector; x is the matrix of predictor values
   b <- coef(nlslmout)
   m <- nlslmout$m
```

[4] This version is needed here because it provides the intermediate quantities we need from the computation. However, we will see in Section 4.2.4 that this version has other important advantages as well.

```
   # y - g:
   resid <- m$resid()
   # row i of hmat will be deriv of g(x[i,],b)
   # with respect to b
   hmat <- m$gradient()
   # calculate the artificial "x" and "y" of
   # the algorithm
   fakex <- hmat
   fakey <- resid + hmat %*% b
   # -1 means no constant term in the model
   lmout <- lm(fakey ~ fakex - 1)
   vcovHC(lmout)
}
```

In addition to nice convergence behavior, the advantage for us here of **nlsLM()** over **nls()** is that the former gives us access to the quantities we need in (4.13), especially the matrix of $h()$ values. We then apply **lm()** one more time, to get an object of type **"lm"**, needed by **vcovHC()**.

Applying this to the enzyme data, we have

```
> nlsvcovhc(z)
           fakex1     fakex2    fakex3
fakex1 0.4708209   1.706591  2.410712
fakex2 1.7065910  10.394496 20.314688
fakex3 2.4107117  20.314688 53.086958
```

This is rather startling. Except for the estimated variance of $\widehat{\beta}_1$, the estimated variances and covariances from Eickert-White are much larger than what **nls()** found under the assumption of homoscedasticity.

Of course, with only 60 observations, both of the estimated covariance matrices must be "taken with a grain of salt." So, let's compare the two approaches by performing a simulation. Here

$$E(Y \mid X = t) = \frac{1}{t'\beta} \tag{4.14}$$

where $t = (t_1, t_2)'$ and $\beta = (\beta_1, \beta_2)'$. We'll take the components of X to be independent and exponentially distributed with mean 1.0, with the heteroscedasticity modeled as being such that the standard deviation of Y given X is proportional to the regression function value at X. We'll use as a check the fact that a N(0,1) variable is less than 1.28 90% of the time, Here is the simulation code:

```
sim <- function(n,nreps) {
   b <- 1:2
   res <- replicate(nreps,{
      x <- matrix(rexp(2*n),ncol=2)
      meany <- 1 / (x %*% b)
      y <- meany + (runif(n) - 0.5) * meany
      xy <- cbind(x,y)
      xy <- data.frame(xy)
      nlout <- nls(X3 ~ 1 / (b1*X1+b2*X2),
         data=xy,start=list(b1 = 1,b2=1))
      b <- coef(nlout)
      vc <- vcov(nlout)
      vchc <- nlsvcovhc(nlout)
      z1 <- (b[1] - 1) / sqrt(vc[1,1])
      z2 <- (b[1] - 1) / sqrt(vchc[1,1])
      c(z1,z2)
   })
   print(mean(res[1,] < 1.28))
   print(mean(res[2,] < 1.28))
}
```

And here is a run of the code:

```
> sim(250,2500)
[1] 0.6188
[1] 0.9096
```

That's quite a difference! Eickert-White worked well, whereas assuming homoscedasticity fared quite poorly. (Similar results were obtained even for $n = 100$.)

4.2.3 Example: Bike Sharing Data

In our bike-sharing data (Section 1.12.2), there are two kinds of riders, *registered* and *casual*. We may be interested in factors determining the mix, i.e.

$$\frac{\text{registered}}{\text{registered + casual}} \tag{4.15}$$

Since the mix proportion is between 0 and 1, we might try the logistic model, introduced in (4.18), in the context of classification, though the example here does not involve a classification problem, and use of **glm()** would be inappropriate. Here are the results:

```
> shar <- read.csv("day.csv",header=T)
> shar$temp2 <- shar$temp^2
> shar$summer <- as.integer(shar$season == 3)
> shar$propreg <- shar$reg / (shar$reg+shar$cnt)
> names(shar)[15] <- "reg"
> library(minpack.lm)
> logit <- function(t1,t2,t3,t4,b0,b1,b2,b3,b4)
   1 / (1 + exp(-b0 - b1*t1 -b2*t2 -b3*t3 -b4*t4))
> z <- nlsLM(propreg ~
logit(temp,temp2,workingday,summer,b0,b1,b2,b3,b4),
   data=shar,start=list(b0=1,b1=1,b2=1,b3=1,b4=1))
> summary(z)
...
Parameters:
    Estimate Std. Error t value Pr(>|t|)
b0 -0.083417   0.020814  -4.008 6.76e-05 ***
b1 -0.876605   0.093773  -9.348  < 2e-16 ***
b2  0.563759   0.100890   5.588 3.25e-08 ***
b3  0.227011   0.006106  37.180  < 2e-16 ***
b4  0.012641   0.009892   1.278    0.202
...
```

As expected, on working days, the proportion of registered riders is higher, as we are dealing with the commute crowd on those days. On the other hand, the proportion doesn't seem to be much different during the summer, even though the vacationers would presumably add to the casual-rider ount.

But are those standard errors trustworthy? Let's look at the Eickert-White versions:

```
> sqrt(diag(nlsvcovhc(z)))
     fakex1      fakex2      fakex3      fakex4
0.021936045 0.090544374 0.092647403 0.007766202
     fakex5
0.007798938
```

Again, we see some substantial differences.

4.2.4 The "Elephant in the Room": Convergence Issues

So far we have sidestepped the fact that any iterative method runs the risk of nonconvergence. Or it might converge to some point at which there is only a local minimum, not the global one — worse than nonconvergence, in the sense that the user might be unaware of the situation.

For this reason, it is best to try multiple, diverse sets of starting values. In addition, there are refinements of the Gauss-Newton method that have better convergence behavior, such as the Levenberg-Marquardt method.

Gauss-Newton sometimes has a tendency to "overshoot," producing too large an increment in b from one iteration to the next. Levenberg-Marquardt generates smaller increments. Interestingly it is a forerunner of *ridge regression* that we'll discuss in Chapter 8. It is implemented in the CRAN package **minpack.lm**, which we used earlier in this chapter.

4.2.5 Example: Eckerle4 NIST Data

The U.S. National Institute of Standards and Technology has available several data sets that serve as test beds for nonlinear regression modeling. The one we'll use here is called Eckerle4.[5] The data are from a NIST study involving circular interference transmittance. Here we predict transmittance from wavelength, with a model like a normal density:

$$\text{mean transmittance} = \frac{\beta_1}{\beta_2}\ \exp\left[-0.5\ \left(\frac{\text{wavelength} - \beta_3}{\beta_2}\right)^2\right] \tag{4.16}$$

NIST also provides sets of starting values. For this data set, the two suggested starting vectors are (1,10,500) and (1.5,5,450), values that apparently came from informal inspection of a plot of the data, seen in Figure 4.1. It is clear, for instance, that $\widehat{\beta}_3$ is around 450. The standard deviation of a normal curve is the distance from the center of the curve to either inflection point, say about 10 here. Since since the maximum value of the curve is about 0.4, we can then solve for an initial guess for $\widehat{\beta}_1$.

It turns out that ordinary **nls()** works for the second set but not the first:

```
> eck <- read.table("Eckerle4.dat",header=T)
> frm <- y ~ (b1/b2) * exp(-0.5*((x-b3)/b2)^2)
```

[5] See http://www.itl.nist.gov/div898/strd/nls/data/LINKS/DATA/Eckerle4.dat.

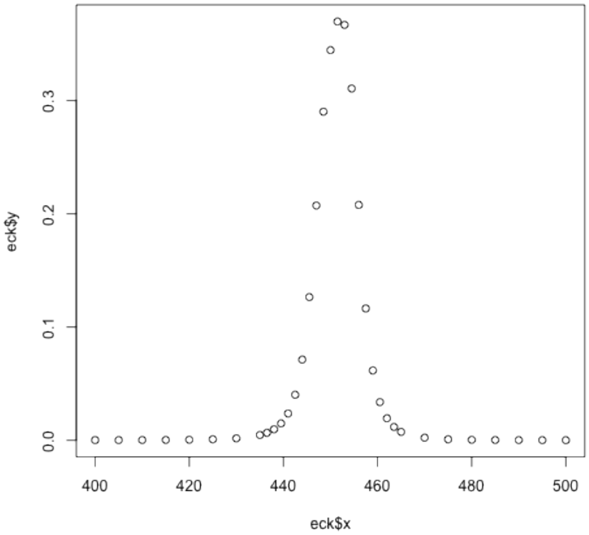

Figure 4.1: Eckerle4 Data

```
...
> bstart
$b1
[1] 1
$b2
[1] 10
$b3
[1] 500
> nls(frm,data=eck,start=bstart)
Error in nls(frm, data = eck, start = bstart) : singular gradient
> bstart <- list(b1=1.5, b2=5, b3=450)
> nls(frm,data=eck,start=bstart)
Nonlinear regression model
  model: y ~ (b1/b2) * exp(-0.5 * ((x - b3)/b2)^2)
   data: eck
     b1       b2       b3
  1.554    4.089  451.541
 residual sum-of-squares: 0.001464
```

```
Number of iterations to convergence: 6
Achieved convergence tolerance: 1.395e-06
```

But **nlsLM()** worked with both sets

4.2.6 The Verdict

Nonlinear regression models can be powerful, but may be tricky to get to converge properly. Moreover, convergence might be achieved at a local, rather than global minimum, in which case the statistical outcome may be problematic. A thorough investigation of convergence (and fit) issues in any application is a must.

Fortunately, for the special case of Generalized Linear Models, the main focus of this chapter, convergence is rarely a problem. So, let's start discussing GLM.

4.3 The Generalized Linear Model

Recall once again the logistic model, introduced in (4.18). We are dealing with a classification problem, so the Y takes on the values 0 and 1. Let $X = (X^{(1)}, X^{(2)}, ..., X^{(p)})'$ denote the vector of our predictor variables.

4.3.1 Definition

Our model is

$$P(Y = 1 \mid X^{(1)} = t_1, ..., X^{(p))} = t_p) = \ell(\beta_0 + \beta_1 t_1 + ... + \beta_p t_p) \qquad (4.17)$$

where

$$\ell(s) = \frac{1}{1 + e^{-s}} \qquad (4.18)$$

and $t = (t_1, ..., t_p)'$.[6]

[6] Recall from Section 1.12.1 that te classification problem is a special case of regression.

The key point is that even though the right-hand side of (4.17) is not linear in t, it is *a function of* a linear expression in t, hence the term *generalized* linear model (GLM).

So, GLM is actually a broad class of models. We can use many different functions $q()$ in place of $\ell()$ in (4.18); for each such function, we have a different GLM.

4.3.2 Example: Poisson Regression

For example, setting $q() = \exp()$ gives us a model known as *Poisson regression*, which assumes

$$E(Y = 1 \mid X^{(1)} = t_1, ..., X^{(p))} = t_p) = e^{\beta_0+\beta_1 t_1+...+\beta_p t_p} \tag{4.19}$$

In addition, GLM assumes a specified parametric class for the conditional distribution of Y given X, which we will denote $F_{Y|X}$. In Poisson regression, this assumption is, not surprisingly, that the conditional distribution of Y given X is Poisson. In the logistic case, the assumption is trivially that the distribution is *Bernoulli*, i.e. binomial with number of trials equal to 1. Having such assumptions enables maximum likelihood estimation.

In particular, the core of GLM assumes that $F_{Y|X}$ belongs to an *exponential family*. This is formally defined as a parametric family whose probability density/mass function has the form

$$\exp[\eta(\theta)T(x) - A(\theta) + B(x)] \tag{4.20}$$

where θ is the parameter vector and x is a value taken on by the random variable. Though this may seem imposing, it sufficies to say that the above formulation includes many familiar distribution families such as Bernoulli, binomial, Poisson, exponential and normal. In the Poisson case, for instance, setting $\eta(\theta) = \log \lambda$, $T(x) = k$, $A(\theta) = -\lambda$ and $B(x) = -\log(k!)$ yields the expression

$$\frac{e^{-\lambda}\lambda^k}{k!} \tag{4.21}$$

the famous form of the Poisson probability mass function.

GLM terminology centers around the *link function*, which is the functional inverse of our function $q()$ agove. For Poisson regression, the link function is the inverse of exp() i.e. log(). For logit, set $u = \ell(s) = (1 + \exp(-s))^{-1}$, and solve for s, giving us the link function,

$$\text{link}(u) = \frac{u}{1-u} \tag{4.22}$$

4.3.3 GLM Computation

Though estimation in GLM uses maximum likelihood, it can be shown that the actual computation can be done extending the ideas in Section 4.2, i.e. via least-squares models. The only new aspect is the addition of a weight function, which works as follows.

Let's review Section 3.11, which discussed weighted least squares in the case of a linear model. Using our usual notation $\mu(t) = E(Y \mid X = t)$ and $\sigma^2(t) = Var(Y \mid X = t)$, the optimal estatimor of β is the value of b that minimizes

$$\sum_{i=1}^{n} \frac{1}{\sigma^2(\widetilde{X_i})} (Y_i - \widetilde{X_i}' b)^2 \tag{4.23}$$

Now consider the case of Poisson regression. One of the famous properties of the Poisson distribution family is that the variance equals the mean. Thus (4.9) becomes

$$\sum_{i=1}^{n} \frac{1}{g(X_i, b)} [Y_i - g(X_i, b)]^2 \tag{4.24}$$

Then (4.13) bceoms

$$\sum_{i=1}^{n} \frac{1}{g(X_i, b_{k-1})} [\underbrace{Y_i - g(X_i, b_{k-1}) + h(X_i, b_{k-1})' b_{k-1}} - h(X_i, b_{k-1})' \ b]^2 \tag{4.25}$$

and we again solve for the new iterate b_k by calling **lm()**, this time making use of the latter's **weights** argument.

We iterate as before, but now the weights are updated at each iteration too. For that reason, the process is known as *iteratively reweighted least squares.*

4.3.4 R’s glm() Function

Of course, the **glm()** function does all this for us. For ordinary usage, the call is the same as for **lm()**, except for one extra argument, **family**. In the Poisson regression case, for example, the call looks like

```
glm(y ~ x, family = poisson)
```

The **family** argument actually has its own online help entry:

```
> ?family
family                      package:stats
R Documentation

Family Objects for Models

Description:
...

Usage:

     family(object, ...)

     binomial(link = ”logit”)
     gaussian(link = ”identity”)
     Gamma(link = ”inverse”)
     inverse.gaussian(link = ”1/mu^2”)
     poisson(link = ”log”)
     quasi(link = ”identity”, variance = ”constant”)
     quasibinomial(link = ”logit”)
     quasipoisson(link = ”log”)

...
```

Ah, so the **family** argument is a function! There are built-in ones we can use, such as the **poisson** one we used above, or a user could define her own custom function.

Well, then, what are the arguments to such a function? A key argument is **link**, which is obviously the link function **q()** discussed above, which we found to be **log()** in the Poisson case.

For a logistic model, as noted earlier, $F_{Y|X}$ is binomial with number of trials m equal to 1. Recall that the variance of a binomial random variable with m trials is $mr(1-r)$, where r is the “success” probability on each trial, Recall too that the mean of a 0-1-valued random variable is the probability of a 1. Putting all this together, we have

$$\sigma^2(t) = \mu(t)[1 - \mu(t)] \tag{4.26}$$

Sure enough, this appears in the code of the built-in function **binomial()**:

```
> binomial
function (link = ”logit”)
{
...
    variance <- function(mu) mu * (1 - mu)
```

Let's now turn to details of two of the most widely-used models, the logistic and Poisson.

4.4 GLM: the Logistic Model

The logistic regression model, introduced in Section 1.12.2, is by far the most popular nonlinear regression method. Here we are predicting a response variable Y that takes on the values 1 and 0, indicating which of two classes our unit belongs to. As we saw in Section 1.12.1, this indeed is a regression situation, as $E(Y \mid X = t)$ reduces to $P(Y = 1 \mid X = t)$.

The model, again, is

$$P(Y = 1 \mid X = (t_1, ..., t_p)) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 t_1 + ... + \beta_p t_p)}} \tag{4.27}$$

4.4.1 Motivation

We noted in Section 1.12.2 that the logistic model is appealing for two reasons: (a) It takes values in [0,1], as a model for probabilities should, and (b) it is monotone in the predictor variables, as in the case of a linear model.

But there's even more. It turns out that the logistic model is related to many familiar distribution families. We'll show that in this section. In addition, the derivations here will deepen the reader's insight into the various conditional probabilities involved in the overall classification problem.

To illustrate that, consider a very simple example of text classification, involving Twitter tweets. Suppose we wish to automatically classify tweets into those involving financial issues and all others. We'll do that by having our code check whether a tweet contains words from a list of financial terms we've set up for this purpose, say *bank*, *rate* and so on.

Here Y is 1 or 0, for the financial and nonfinancial classes, and X is the number of occurrences of terms from the list. Suppose that from past data we know that *among financial tweets*, the number of occurrences of words from this list has a Poisson distribution with mean 1.8, while for nonfinancial tweets the mean is 0.2. Mathematically, that says that $F_{X|Y=1}$ is Poisson with mean 1.8, and $F_{X|Y=0}$ is Poisson with mean 0.2. (Be sure to distinguish the situation here, in which $F_{X|Y}$ is a Poisson distribution, from Poisson regression (Section 4.3.2), in which it is assumed that $F_{Y|X}$ is Poisson.) Finally, suppose 5% of all tweets are financial.

Recall once again (Section 1.12.1) that in the classification case, our regression function takes the form

$$\mu(t) = P(Y = 1 \mid X = t) \tag{4.28}$$

Let's calculate this function:

$$\begin{aligned} P(Y = 1 \mid X = t) &= \frac{P(Y = 1 \text{ and } X = t)}{PX = t)} \qquad (4.29) \\ &= \frac{P(Y = 1 \text{ and } X = t)}{P(Y = 1 \text{ and } X = t \text{ or } Y = 1 \text{ and } X = t)} \\ &= \frac{\pi \; P(X = t \mid Y = 1)}{\pi \; P(X = t \mid Y = 1) + (1-\pi) \; P(X = t \mid Y = 0)} \end{aligned}$$

where $\pi = P(Y = 1)$ is the population proportion of individuals in class 1.

The numerator in (4.29) is

$$0.05 \cdot \frac{e^{-1.8} \; 1.8^t}{t!} \tag{4.30}$$

and similarly the denominator is

$$0.05 \cdot \frac{e^{-1.8} \; 1.8^t}{t!} + 0.95 \cdot \frac{e^{-0.2} \; 0.2^t}{t!} \tag{4.31}$$

Putting this into (4.29) and simplifying, we get

$$P(Y = 1 \mid X = t) = \frac{1}{1 + 19e^{1.6(\frac{1}{9})^t}} \tag{4.32}$$

$$= \frac{1}{1 + \exp(\log 19 - 0.2 - t \log 9)} \tag{4.33}$$

That last expression is of the form

$$\frac{1}{1 + \exp[-(\beta_0 + \beta_1 t)]} \tag{4.34}$$

with

$$\beta_0 = -\log 19 + 0.2 \tag{4.35}$$

and

$$\beta_1 = \log 9 \tag{4.36}$$

In other words the setting in which $F_{X|Y}$ is Poisson implies the logistic model!

This is true too if $F_{X|Y}$ is an exponential distribution. Since this is a continuous distribution family rather than a discrete one, the quantities $P(X = t|Y = i)$ in (4.32) must be replaced by density values:

$$P(Y = 1 \mid X = t) =$$

$$\frac{\pi \ f_1(X = t \mid Y = 1)}{\pi \ f_1(X = t \mid Y = 1) + (1 - \pi) \ f_0(X = t \mid Y = 0)} \tag{4.37}$$

where the within-class densities of X are

$$f_i(w) = \lambda_i e^{-\lambda_i w}, \quad i = 0, 1 \tag{4.38}$$

After simplifying, we again obtain a logistic form.

Most important, consider the multivariate normal case: Say for groups $i = 0, 1$, $F_{X|Y=i}$ is a multivariate normal distribution with mean vector μ_i and covariance matrix Σ, where the latter does *not* have a subscript i. This is a generalization of the classical two-sample t-test setting, in which two (scalar) populations are assumed to have possibly different means but the same variance.[7] Again using (4.37), and going through a lot of algebra, we find that again $P(Y = 1 \mid X = t)$ turns out to have a logistic form,

$$P(Y = 1 \mid X = t) = \frac{1}{1 + e^{-(\beta_0 + \overline{\beta}' t)}} \tag{4.39}$$

with

$$\beta_0 = \log(1 - \pi) - \log \pi + \frac{1}{2}(\mu_1' \mu_1 - \mu_0' \mu_0) \tag{4.40}$$

and

$$\overline{\beta} = (\mu_0 - \mu_1)' \Sigma^{-1} \tag{4.41}$$

where t is the vector of predictor variables, the β vector is broken down into $(\beta_0, \overline{\beta})$, and π is $P(Y = 1)$. The messy form of the coefficients here is not important; instead, the point is that we find that the multivariate normal model implies the logistic model, giving the latter even more justification.

In summary:

> Not only is the logistic model intuitively appealing because it is a monotonic function with values in (0,1), but also because it is implied by various familiar parametric models for the within-class distribution of X.

No wonder the logit model is so popular!

4.4.2 Example: Pima Diabetes Data

Another famous UCI data set is from a study of the Pima tribe of Native Americans, involving factors associated with diabetes. There is data on 768

[7] It is also the setting for *Fisher's Linear Discriminant Analysis*, to be discussed in Section **??**.

women.[8] Let's preduct diabetes from the other variables:

```
> logitout <- glm(Diab ~ .,data=pima,family=binomial)
> summary(logitout)
...
Coefficients:
              Estimate Std. Error z value
(Intercept) -8.4046964  0.7166359 -11.728
NPreg        0.1231823  0.0320776   3.840
Gluc         0.0351637  0.0037087   9.481
BP          -0.0132955  0.0052336  -2.540
Thick        0.0006190  0.0068994   0.090
Insul       -0.0011917  0.0009012  -1.322
BMI          0.0897010  0.0150876   5.945
Genet        0.9451797  0.2991475   3.160
Age          0.0148690  0.0093348   1.593
            Pr(>|z|)
(Intercept)  < 2e-16 ***
NPreg       0.000123 ***
Gluc         < 2e-16 ***
BP          0.011072 *
Thick       0.928515
Insul       0.186065
BMI         2.76e-09 ***
Genet       0.001580 **
Age         0.111192
...
```

4.4.3 Interpretation of Coefficients

In nonlinear regression models, the parameters β_i do not have the simple marginal interpretation they enjoy in the linear case. Statements like we made in Section 1.7.1.2, "We estimate that, on average, each extra year of age corresponds to almost a pound in extra weight," are not possible here.

However, in the nonlinear case, the regression function is still defined as the conditional mean, or in the logit case, the conditional probability of a 1. Practical interpretation is definitely still possible, if slightly less convenient.

Consider for example the estimated Glucose coefficient in our diabetes data

[8] The data set is at `https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`. I have added a header record to the file.

above, 0.035. Let's apply that to the people similar to the first person in the data set:

```
> pima[1,]
  NPreg Gluc BP Thick Insul  BMI Genet Age Diab
1     6  148 72    35     0 33.6 0.627  50    1
```

Ignore the fact that this woman has diabetes. Let's consider the population group of all women with the same characteristics as this one, i.e. all who have had 6 pregnancies, a glucose level of 148 and so on, through an age of 50. The estimated proportion of women in this population group is

$$\frac{1}{1+e^{-(8.4047+0.1232\cdot 6+...+0.0149\cdot 50)}} \tag{4.42}$$

We don't have to plug these numbers in by hand, of course:

```
> l <- function(t) 1/(1+exp(-t))
> pima1 <- pima[1,-9]  # exclude diab. var.
> pima1 <- unlist(pima[1,-9])  # had been a data frame
> l(coef(logitout) %*% c(1,pima1))
          [,1]
[1,] 0.7217266
```

So, about 72% of women in this population group have diabetes. But what about the population group of the same characteristics, but of age 40 instead of 50?

```
> w <- pima1
> w['Age'] <- 40
> l(coef(logitout) %*% c(1,w))
          [,1]
[1,] 0.6909047
```

Only about 69% of the younger women have diabetes.

So, there is an effect of age on developing diabetes, but only a mild one; a 10-year increase in age only increased the chance of diabetes by about 3.1%. However, note carefully that this was for women having a given set of the other factors, e.g. 6 pregnancies. Let's look at a different population group, those with 2 pregnancies and a glucose level of 120, comparing 40- and 50-year-olds:

```
> u <- pima1
> u[1] <- 2
> u[2] <- 100
> v <- u
> v[8] <- 40
> l(coef(logitout) %*% c(1,u))
          [,1]
[1,] 0.2266113
> l(coef(logitout) %*% c(1,v))
          [,1]
[1,] 0.2016143
```

So here, the 10-year age effect was somewhat less, about 2.5%. A more careful analysis would involve calculating standard errors for these numbers, but the chief point here is that the effect of a factor in nonlinear situations depends on the values of the other factors.

$$P(Y = 0 \mid X^{(1)} = t_1, ..., X^{(p))} = t_p) = 1 - \ell(\beta_0 + \beta_1 t_1 + ... + \beta_p t_p) \tag{4.43}$$

Some analysts like to look at the *log-odds ratio*,

$$\frac{P(Y = 1 \mid X^{(1)} = t_1, ..., X^{(p))} = t_p)}{P(Y = 0 \mid X^{(1)} = t_1, ..., X^{(p))} = t_p)} \tag{4.44}$$

in this case the ratio of the probability of having and not having the disease. By Equation (4.17), this simplifies to

$$\beta_0 + \beta_1 t_1 + ... + \beta_p t_p \tag{4.45}$$

— a linear function! Thus, in interpreting the coefficients output from a logisitic analysis, it is convenient to look at the log-odds ratio, as it gives us a single number for each factor. This may be sufficient for the application at hand, but a more thorough analysis should consider the effects of the factors on the probabilities themselves.

4.4.4 The predict() Function

IN the previous section, we evaluated the estimated regression function (and thus predicted values as well) the straightforward but messy way, e.g.

```
> l(coef(logitout) %*% c(1,v))
```

The easy way is to use R's **predict()** function:

```
> predict(object=logitout,newdata=pima[1,-9],
   type='response')
        1
0.7217266
```

Let's see what just happened. First, **predict()** is what is called a *generic* function in R. What this means is that it is not a single function, but rather an umbrella leading into a broad family of functions, depending on the class of the object on which it is invoked.

In our case here, we invoked it on **logitout**. What is the class of that object?

```
> class(logitout)
[1] "glm" "lm"
```

So, it is an object of class **"glm"** (and, we see, the latter is a subclass of the class **"lm"**).

In processing our call to **predict()**, the R interpreter found that our object had class **"glm", and thus looked for a function predict.glm()**, which the authors of **glm()** had written for us. So, in R terminology, the interpreter **dispatched**, i.e. relayed, our **priedct()**call to **predict.glm()**. What did the latter then do?

The argument **newdata** is a data frame consisting of what its name implies — new cases to predict. As you will recall, the way we predict a new case in regression analysis is to take as our prediction the value of the regression function for that casew. The net result is then that **predict()** is giving us the value of the regression function at our requested points, in this case for the population group of all women with the same traits as the first person in our data.

So, **predict()** while doesn't give us any new capabilities, it certainly makes things more convenient. The reader should ponder, for instance, how to use this function to simplify the code in Section 1.9.4.1.

4.4.5 Linear Boundary

In (4.27), which values of $t = (t_1, ..., t_p)'$ will cause us to gues $Y = 1$ and which will result in a guess of $Y = 0$? The boundary occurs when (4.27) has the value 0.5. In other words, the boundary consists of all t such that

$$\beta_0 + \beta_1 t_1 + ... + \beta_p t_p = 0 \tag{4.46}$$

So, the boundary has linear form, a *hyperplane* in p-dimensional space. This may seem somewhat abstract now, but it will have value later on.

4.5 GLM: the Poisson Model

Since in the above data the number of pregnancies is a count, we might consider predicting it using Poission regression. (The reader may wonder why we may be interested in such a "reverse" prediction. It could occur, for instance, with *multiple imputation* methods to deal with missing data.) Here's how we can do this with **glm()**:

```
> poisout <- glm(NPreg ~ .,data=pima,family=poisson)
> summary(poisout)
...
Coefficients:
             Estimate Std. Error z value
(Intercept)  0.2963661  0.1207149   2.455
Gluc        -0.0015080  0.0006704  -2.249
BP           0.0011986  0.0010512   1.140
Thick        0.0000732  0.0013281   0.055
Insul       -0.0003745  0.0001894  -1.977
BMI         -0.0002781  0.0027335  -0.102
Genet       -0.1664164  0.0606364  -2.744
Age          0.0319994  0.0014650  21.843
Diab         0.2931233  0.0429765   6.821
            Pr(>|z|)
(Intercept)  0.01408 *
Gluc         0.02450 *
BP           0.25419
Thick        0.95604
Insul        0.04801 *
BMI          0.91896
Genet        0.00606 **
Age          < 2e-16 ***
Diab        9.07e-12 ***
...
```

On the other hand, even if we believe that our count data follow a Poisson distribution, there is no law dictating that we use Poisson regression, i.e. the model (4.3.2). The main motivation for using exp() in that model is to ensure that our regression function is nonnegative, conforming to the nonnegative nature of Poisson random variables. This is not unreasonable, but as noted in a somewhat different context in Section 3.3.7, transformations like this can produce distortions. Let's try an alternative:

```
> quasiout <- glm(NPreg ~ .,data=pima,
     family=quasi(variance="mu^2"),start=rep(1,9))
```

This "quasi" family is a catch-all option, specifying a linear model but here allowing us to specify a Poisson variance function.

Well, then, which model performed better? As a rough, quick look, ignoring issues of overfitting and the like, let's consider R^2. This quantity is not calculated by **glm()**, but recall from Section 2.7.2 that R^2 is the squared correlation between the predicted and actual Y values. This quantity makes sense for any regression situation, so let's calculate it here:

```
> cor(poisout$fitted.values,poisout$y)^2
[1] 0.2314203
> cor(quasiout$fitted.values,quasiout$y)^2
[1] 0.3008466
```

The "unorthodox" model performed better. We cannot generalize from this, but it does show again that one must use transformations carefully.

Chapter 5

Multiclass Classification Problems

In classification problems we've discussed so far, we have assumed just two classes. The patient either has the disease in question, or not; the customer chooses to buy a certain item, or not; and so on.

But in many applications, we have multiple classes. We may, for instance, be considering several different diseases that a patient might have.[1] In computer vision applications, the number of classes can be quite large, say face recognition with data on a large number of people. Let m denote the number of classes, and label them 0, 1, ..., m - 1.

Say for instance we wish to do machine recognition of handwritten digits, so we have 10 classes, with our variables being various patterns in the pixels, e.g. the number of (approximately) straight line segments. Instead of having a single response variable Y as before, we would now have 10 of them, setting $Y^{(i)}$ to be 1 or 0, according to whether the given digit is i, for $i = 0, 1, ..., 9$. We could run 10 logistic regression models, and then use each one to estimate the probability that our new image represents a certain digit.

In general, as above, let $Y^{(i)}, i = 0, ..., m - 1$ be the indicator variables for

[1] For a classification problem, the classes must be mutually exclusive. In this case, there would be the assumption that the patient does not have more than one of the diseases.

the classes, and define the *class probabilities*

$$\pi_i = P(Y^{(i)} = 1), \quad i = 0, 1..., m-1 \tag{5.1}$$

Of course, we must have

$$\sum_{i=0}^{m-1} \pi_i = 1 \tag{5.2}$$

We will still refer to Y, now meaning the value of i for which $Y^{(i)} = 1$.

Note that in this chapter, we will be concerned primarily with the Prediction goal, rather than Description.

5.1 The Key Equations

Equations (4.29) and (4.37), and their generalizations, will play a key role here. Let's relate our new multiclass notation to what we had in the two-class case before. If $m = 2$, then:

- What we called $Y^{(1)}$ above was just called Y in our previous discussion of the two-class case.
- The class probability π_1 here was called simply π previously.

Now, let's review from the earlier material. (Keep in mind that typically X will be vector-valued, as we typically have more than one predictor variable.) For $m = 2$:

- The quantity of interest is $P(Y = 1 \mid X = t)$.
- If X has a discrete distribution, then

$$\mu(t) = P(Y = 1 \mid X = t) = \frac{\pi \; P(X = t \mid Y = 1)}{\pi \; P(X = t \mid Y = 1) + (1 - \pi) \; P(X = t \mid Y = 0)} \tag{5.3}$$

- If X has a continuous distribution,

$$\mu(t) = P(Y = 1 \mid X = t) = \frac{\pi \ f_1(t)}{\pi \ f_1(t) + (1 - \pi) \ f_0(t)} \tag{5.4}$$

  where the within-class densities of X are f_1 and f_0.[2]

- Sometimes it is more useful to use the following equivalence to (5.4):

$$P(Y = 1 \mid X = t) = \frac{1}{1 + \frac{1-\pi}{\pi} \frac{f_0(t)}{f_1(t)}} \tag{5.5}$$

Note that, in keeping with the notion that classification amounts to a regression problem (Section 1.12.1), we have used our regression function notation $\mu(t)$ above.

Things generalize easily to the multiclass case. We are now interested in the quantities

$$P(Y = i) = \mu_i(t) = P(Y^{(i)} = 1 \mid X = t), \ i = 0, 1, ..., m - 1 \tag{5.6}$$

For continuous X, (5.4) becomes

$$P(Y = i) = \mu_i(t) = P(Y^{(i)} = 1 \mid X = t) = \frac{\pi_i \ f_i(t)}{\sum_{j=0}^{m-1} \pi_j \ f_j(t)} \tag{5.7}$$

5.2 How Do We Use Models for Prediction?

In Section 1.8, we discussed the specifics of predicting new cases, in which we know "X" but not "Y," after fitting a model to data in which both "X" and "Y" are known (our training data). The parametric and nonparametric cases were slightly different.

[2] Another term for the classification probabilities π_i is *prior probabilities*. Readers familar with the controversy over *Bayesian* versus *frequentist* approaches to statistics may wonder if we are dealing with Bayesian analyses here. Actually, that is not the case; we are not working with subjective, "gut feeling" probabilities as in the controversial Bayesian methods. There is some connection, in the sense that (5.3) and (5.4) make use of Bayes' Rule, but the latter is standard for all statisticians, frequentist and Bayesian alike. Note by the way that quantities nrobabilities like (5.4) are often termed *posterior* probabilities, again sounding Bayesian but again actually Bayesian/frequentist-neutral.

The situation is the same here in the multiclass setting. The only difference is that now multiple functions $\mu_i(t), i = 0, 1, ..., m-1$ need to be estimated from our training data, as opposed to just $\mu(t)$ before.

It should be noted, though, that some nonparametric methods do not explicitly estimate $\mu_i(t)$, and instead only estimate "boundaries" involving those functions. These methods will be discussed in Chapter 11.

5.3 Misclassification Costs

One context to consider is the informal. Say we are trying to determine whether a patient has a particular disease, based on a vector X of various test results, demographic variables and so on for this patient. Denote the value of X by t_c, and suppose our estimate of $P(Y^{(1)} = 1 \mid X = t_c)$ is 0.02. We estimate that this patient has only a 2% chance of having the disease. This isn't very high, so we might simply stop there.

On the other hand, a physician may have a hunch, based on information not in X and thus not in our sample data, that leads her to suspect that the patient does have the disease. The physician may thus order further tests and so on, in spite of the low estimated probability.

Moreover, in the case of a catastrophic disease, the *misclassification costs* may not be equal; failing to detect the disease when it's present may be a much more serious error than ordering further medical tests that turn out to be negative.

So, in the context in which we're doing classification based on "gut feeling," our estimated $P(Y^{(i)} = 1 \mid X = t_c)$ can be used as just one of several components that enter our final decision.

In many applications today, though, our classification process will be automated, done entirely by machine. Consider the example in Section 4.4.1 of classifying subject matter of Twitter tweets, say into financial tweets and all others, a two-class setting. Here again there may be unequal misclassification costs, depending on our goals. If so, the prescription

$$\text{guess for } Y = \begin{cases} 1, & \text{if } \mu(t_c) > 0.5 \\ 0, & \text{if } \mu(t_c) \leq 0.5 \end{cases} \tag{5.8}$$

from Section 1.13.2 is not what we want, as it implicitly assumed equal costs.

If we wish to automate, we'll probably need to set up a formal cost structure. Let ℓ_0 denote our cost for guessing Y to be 1 when it's actually 0, and define ℓ_1 for the opposite kind of error. Now reason as follows as to what we should guess for Y, knowing that $X = t_c$. For convenience, write

$$p = P(Y = 1 \mid X = t_c) \tag{5.9}$$

Suppose we guess Y to be 1. Then our expected cost is

$$(1-p)\ell_0 \tag{5.10}$$

If on the other hand we guess Y to be 0, our expected cost is

$$p\ell_1 \tag{5.11}$$

So, our strategy could be to choose our guess to be the one that gives us the smaller of (5.10) and (5.11):

$$\text{guess for } Y = \begin{cases} 1, & \text{if } (1-p)\ell_0 \leq p\ell_1 \\ 0, & \text{if } (1-p)\ell_0 > p\ell_1 \end{cases} \tag{5.12}$$

In other words, given X, we guess Y to be 1 if

$$\mu(X) \geq \frac{\ell_0}{\ell_0 + \ell_1} \tag{5.13}$$

This all seems pretty abstract, but it is actually simple. If we consider wrongly guessing Y to be 1 as 10 times worse than wrongly guessing Y to be 0, then the right-hand side of (5.13) is 1/11, or about 0.09. So, if we estimate the conditional probability of Y being 1 is more than 9%, we go ahead and guess 1.

From this point onward, we will assume equal costs.

5.4 One vs. All or All vs. All?

Let's consider the Vertebral Column data from the UC Irvine Machine Learning Repository. Here there are $m = 3$ classes: Normal, Disk Hernia and Spondylolisthesis. The predictors are, as described on the UCI site, "six biomechanical attributes derived from the shape and orientation of the pelvis." Consider two approaches we might take to predicting the status of the vertebral column, based on logistic regression:

- **One vs. All (OVA):** Here we would fit 3 logit models to our training data, predicting each of the 3 classes, one at a time. The i^{th} model would regress $Y^{(i)}$ against the 6 predictor variables, yielding $\widehat{\mu}_i(t), i = 0, 1, 2$. To predict Y for $X = t_c$, we would guess Y to be whatever i has the largest value of $\widehat{\mu}_i(t_c)$, i.e. the most likely class, given the predictor values.
- **All vs. All (AVA):** Here we would fit 3 logit models again, but with one model for each possible pair of clases. Our first model would pit class 0 against class 1, meaning that we would restrict our data to only those cases in which the class is 0 or 1, then predict class 0 in that restricted data set. Our second logit model would restrict to the classes 0 and 2, and predict 0, while the last model would be for classes 1 and 2, predicting 1. (We would still use our 6 predictor variables in each model.)

Note that it was just coincidence that we have the same number of models in the OVA and AVA approaches here (3 each). In general, with m classes, we will run m logistic models (or kNN or whatever type of regression modeling we like) under OVA, but $C(m, 2) = m(m-1)/2$ models under AVA.[3]

5.4.1 R Code

To make this concrete, here is code for the two approaches:

```
# One-vs.-All (OVA) and All-vs.All (AVA),
# logit models

# arguments:

#    m:     number of classes
#    trnxy:  X, Y training set; Y in last column;
#            Y coded 0,1,...,m-1 for the m classes
#    predx:  X values from which to predict Y values
#    tstxy:  X, Y test set, same format
```

[3] Here the notation $C(r, s)$ means the number of combinations one can form from r objects, taking them s at a time.

```

##########################################################
# ovalogtrn: generate estimated regression functions
##########################################################

# arguments:

#     m:  as above
#     trnxy:  as above

# value:

#     matrix of the betahat vectors, one per column

ovalogtrn <- function(m,trnxy) {
   p <- ncol(trnxy)
   x <- as.matrix(trnxy[,1:(p-1)])
   y <- trnxy[,p]
   outmat <- NULL
   for (i in 0:(m-1)) {
      ym <- as.integer(y == i)
      betahat <- coef(glm(ym ~ x,family=binomial))
      outmat <- cbind(outmat,betahat)
   }
   outmat
}

##########################################################
# ovalogpred: predict Ys from new Xs
##########################################################

# arguments:
#
#     coefmat:  coef. matrix, output from ovalogtrn()
#     predx:  as above
#
# value:
#
#     vector of predicted Y values, in {0,1,...,m-1},
#     one element for each row of predx

ovalogpred <- function(coefmat,predx) {
   # get est reg ftn values for each row of predx
```

```
    # and each col of coefmat; vals from
    # coefmat[,] in tmp[,i]
    tmp <- as.matrix(cbind(1,predx)) %*% coefmat
    tmp <- logit(tmp)
    apply(tmp,1,which.max) - 1
}

##########################################################
# avalogtrn: generate estimated regression functions
##########################################################

# arguments:

#    m:  as above
#    trnxy:  as above

# value:

#    matrix of the betahat vectors, one per column,
#    in the order of combin()

avalogtrn <- function(m,trnxy) {
    p <- ncol(trnxy)
    n <- nrow(trnxy)
    x <- as.matrix(trnxy[,1:(p-1)])
    y <- trnxy[,p]
    outmat <- NULL
    ijs <- combn(m,2)
    doreg <- function(ij) {
        i <- ij[1] - 1
        j <- ij[2] - 1
        tmp <- rep(-1,n)
        tmp[y == i] <- 1
        tmp[y == j] <- 0
        yij <- tmp[tmp != -1]
        xij <- x[tmp != -1,]
        coef(glm(yij ~ xij,family=binomial))
    }
    coefmat <- NULL
    for (k in 1:ncol(ijs)) {
        coefmat <- cbind(coefmat,doreg(ijs[,k]))
    }
    coefmat
```

```
}

##########################################################
# avalogpred: predict Ys from new Xs
##########################################################

# arguments:
#
#    m: as above
#    coefmat: coef. matrix, output from avalogtrn()
#    predx: as above
#
# value:
#
#    vector of predicted Y values, in {0,1,...,m-1},
#    one element for each row of predx

avalogpred <- function(m,coefmat,predx) {
   ijs <- combn(m,2)  # as in avalogtrn()
   n <- nrow(predx)
   ypred <- vector(length = n)
   for (r in 1:n) {
      # predict the rth new observation
      xrow <- c(1,unlist(predx[r,]))
      # wins[i] tells how many times class i-1 has won
      wins <- rep(0,m)
      for (k in 1:ncol(ijs)) {
         i <- ijs[1,k]  # class i-1
         j <- ijs[2,k]  # class j-1
         bhat <- coefmat[,k]
         mhat <- logit(bhat %*% xrow)
         if (mhat >= 0.5) wins[i] <- wins[i] + 1 else
         wins[j] <- wins[j] + 1
      }
      ypred[r] <- which.max(wins) - 1
   }
   ypred
}

logit <- function(t) 1 / (1+exp(-t))
```

For instance, under OVA, we call **ovalogtrn()** on our training data, yielding a logit coefficient matrix having m columns; the i^{th} column will consist of the estimated coefficients from fitting a logit model predicting $Y^{(i)}$. We then use this matrix as input for predicting Y in all future cases that come our way, by calling **ovalogpred()** whenever we need to do a prediction.

Under AVA, we do the same thing, calling **avalogtrn()** and **avalogpred()**.

5.4.2 Which Is Better?

Clearly, AVA involves a lot of computation. For fixed number of predictor variables p, here is a rough time estimate. For a logit model, the computation will be proportional to the number of cases n (due to computing various sums over all cases). Say our training data is approximately balanced in terms of sizes of the classes, so that the data corresponding to class i has about n/m cases in it, Then the computation for one pair will be $O(n/m)$, but there will be $O(m^2)$ pairs, so the total amount of computation will be $O(mn)$ — potentially much larger than the $O(n)$ used by OVA.

Well, then, do we benefit from that extra computation? At least at first glance, AVA would not seem to have much to offer. For instance, since each of its models uses much less than our full data, the resulting estimated coefficients will likely be less accurate than what we calculate under OVA. And if m is large, we will have so many pairs that at least some will likely be especially inaccurate. And yet some researchers claim they find AVA to work better.

To better understand the situation, let's consider an example and draw upon some intuition.

5.4.3 Example: Vertebrae Data

Here we continue with the vertebrae data, applying the OVA and AVA methods to a training set of 225 randomly chosen records, then predicting the remaining records:[4]

```
> vert <- read.table('Vertebrae/column_3C.dat',
     header=FALSE)
> vert$V7 <- as.numeric(vert$V7) - 1
> trnidxs <- sample(1:310,225)
> predidxs <- setdiff(1:310,trnidxs)
```

[4] To avoid clutter, some messages, "glm.fit: fitted probabilities numerically 0 or 1 occurred," have been removed, here and below. The warnings should not present a problem.

```
> ovout <- ovalogtrn(3,vert[trnidxs,])
> predy <- ovalogpred(ovout,vert[predidxs,1:6])
> mean(predy == vert[predidxs,7])
[1] 0.8823529
> avout <- avalogtrn(3,vert[trnidxs,])
> predy <- avalogpred(3,avout,vert[predidxs,1:6])
> mean(predy == vert[predidxs,7])
[1] 0.8588235
```

Note that **ovalogpred()** requires that Y be coded $0, 1, ..., m-1$, hence the call to **as.numeric()**.

The two correct-classification rates here are, of course, subject to sampling error, but in any case AVA did not seem superior.

5.4.4 Intuition

To put this in context, consider the artificial example in Figure 5.1, adapted from Friedman (1996). Here we have $m = 3$ classes, with $p = 2$ predictors. For each class, the bulk of the probability mass is assumed to lie within one of the circles.

Now suppose a logistic model were used here. It implies that the prediction boundary between our two classes is linear. The figure shows that a logit model would fare well under AVA, because for any pair of classes, there is a straight line (pictured) that separates that pair of classes well. But under OVA, we'd have a problem; though a straight line separates the top circle from the bottom two, there is no straight line that separates the bottom-left circle well from the other two; the boundary between that bottom-left circle and the other two would be a curve.

The real problem here, of course is that the logit is not a good model in such a situation.

5.4.5 Example: Letter Recognition Data

Following up on the notion that AVA may work to reduce model bias, i.e. that AVA's value occurs in settings in which our model is not very good, let's look at an example in which we know the model is imperfect.

The UCI Letters Recognition data set [5] uses various summaries of pixel

[5] `https://archive.ics.uci.edu/ml/datasets/Letter+Recognition`; a lso available

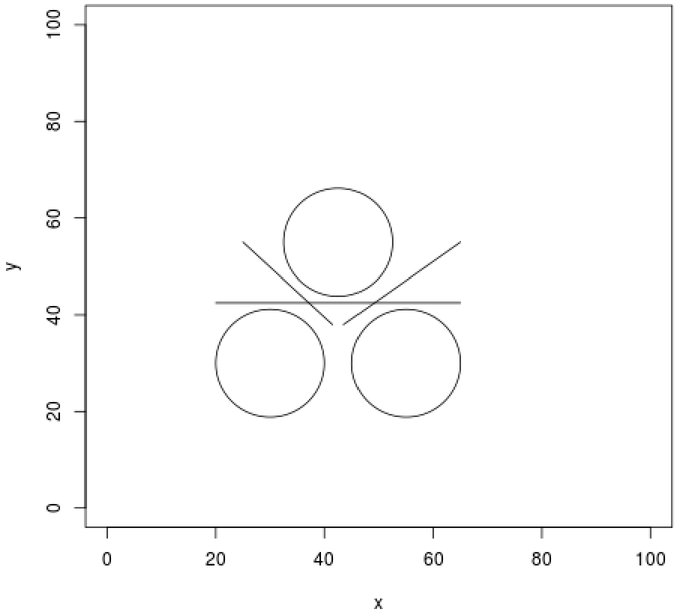

Figure 5.1: Three Artificial Regression Lines

patterns to classify images of capital English letters. A naively applied logistic model may sacrifice some accuracy here, due to the fact that the predictors do not necessarily have monotonic relations with the response variable, the class identity.

The naive approach actually doesn't do too poorly:

```
> library(mlbench)
> data(LetterRecognition)
> lr <- LetterRecognition
> lr[,1] <- as.numeric(lr[,1]) - 1
> # training and test sets
> lrtrn <- lr[1:14000,]
> lrtest <- lr[14001:20000,]
> ologout <- ovalogtrn(26,lrtrn[,c(2:17,1)])
> ypred <- ovalogpred(ologout,lrtest[,-1])
> mean(ypred == lrtest[,1])
[1] 0.7193333
```

We will see shortly that one can do considerably better. But for now, we have a live candidate for a "poor model example," on which we can try AVA:

```
> alogout <- avalogtrn(26,lrtrn[,c(2:17,1)])
> ypred <- avalogpred(26,alogout,lrtest[,-1])
> mean(ypred == lrtest[,1])
[1] 0.8355
```

That is quite a difference! So, apparently AVA fixed a poor model. Of course, its better to make a good model in the first place. Based on our previous observation that the boundaries may be better approximated by curves than lines, let's try a quadratic model.

A full quad model with have all squares and interactions among the 16 predictors. But there are $16 \cdot 17/2 = 136$ of them! That risks overfitting, so let's settle for just adding the squares of the predictors:

```
> for (i in 2:17) lr <- cbind(lr,lr[,i]^2)
> lrtrn <- lr[1:14000,]
> lrtest <- lr[14001:20000,]
> ologout <- ovalogtrn(26,lrtrn[,c(2:33,1)])
> ypred <- ovalogpred(ologout,lrtest[,-1])
> mean(ypred == lrtest[,1])
[1] 0.8086667
```

in the R package **mlbench**.

Ah, much better. Not quite as good as AVA, but rather commensurate with sampling error, and we didn't even try interaction terms.

5.4.6 The Verdict

With proper choice of model, it is my experience that OVA does as well as AVA, if not better. And in paper supporting OVA, Rifkin (2004) contends that some of the pro-AVA experiments were not done properly.

Clearly, though, our letters recognition example shows that AVA is worth considering. We will return to this issue later.

5.5 The Classical Approach: Fisher Linear Discriminant Analysis

Sir Ronald Fisher (1890-1962) was one of the pioneers of statistics. He called his solution to the multiclass problem *linear discriminant analysis* (LDA), now considered a classic.

It is assumed that within class i, the vector of predictor variables X has a multivariate normal distribution with mean vector μ_i and covariance matrix Σ. Note that the latter does *not* have a subscript i, i.e. the covariance matrix for X is the same in each class.

5.5.1 Background

To explain this method, let's review some material from Section 4.4.1.

Let's first temporarily go back to the two-class case, and use our past notation:

$$Y = Y^{(1)}, \quad \pi = \pi_1 \tag{5.14}$$

For convenience, let's reproduce (5.4) here:

$$P(Y = 1 \mid X = t) = \frac{\pi \; f_1(t)}{\pi \; f_1(t) + (1-\pi) \; f_0(t)} \tag{5.15}$$

5.5.2 Derivation

As noted in Section 4.4.1, after substituting the multivariate normal density for the f_i in (5.15), we find that

$$P(Y = 1 \mid X = t) = \frac{1}{1 + e^{-(\beta_0 + \overline{\beta}' t)}} \tag{5.16}$$

with

$$\beta_0 = \log(1 - \pi) - \log \pi + \frac{1}{2}(\mu_1' \mu_1 - \mu_0' \mu_0) \tag{5.17}$$

and

$$\overline{\beta} = (\mu_0 - \mu_1)' \Sigma^{-1} \tag{5.18}$$

Intuitively, if we observe $X = t$, we should predict Y to be 1 if

$$P(Y = 1 \mid X = t) > 0.5 \tag{5.19}$$

and this was shown in Section 1.12.1 to be the optimal strategy.[6] Combining this with (5.16), we predict Y to be 1 if

$$\frac{1}{1 + e^{-(\beta_0 + \overline{\beta}' t)}} > 0.5 \tag{5.20}$$

which simplifies to

$$\overline{\beta}' t < -\beta_0 \tag{5.21}$$

So it turns out that our decision rule is linear in t, hence the term *linear* in *linear discriminant analysis*.[7]

Without the assumption of equal covariance matrices, (5.21) turns out to be quadratic in t, and is called *quadratic discriminant* analysis.

[6] Again assuming equal costs of the two types of misclassification.

[7] The word *discriminant* alludes to our trying to distinguish between $Y = 1$ and $Y = 0$.

5.5.3 Example: Vertebrae Data

Let's apply this to the vertebrae data, using the **lda()** function in the **MASS** library that is built-in to R. This function assumes that the class variable is a factor, so we won't convert to numeric codes this time.

5.5.3.1 LDA Code and Results

Here is the code:

```
> ldaout <- lda(V7 ~ .,data=vert,CV=TRUE)
> mean(ldaout$class == vert$V7)
[1] 0.8096774
```

That **CV** argument tells **lda()** to predict the classes after fitting the model, using (5.4) and the multivariate normal means and covariance matrix that it estimated from the data. Here we find a correct-classification rate of about 81%. This is biased upward, since we didn't bother here to set up separate training and test sets, but even then we did not do as well as our earlier logit analysis. Note that in the latter, we didn't assume a common covariance matrix within each class, and that may have made the difference.

5.5.3.2 Comparison to kNN

By the way, that 81% figure is worth comparing to to that obtained using the k-Nearest Neighbor method. Here is the **regtools** code:

```
ovaknntrn <- function(y,xdata,m,k) {
   if (m < 3) stop('m must be at least 3; use knnest()3')
   x <- xdata$x
   outmat <- NULL
   for (i in 0:(m-2)) {
      yi <- as.integer(y == i)
      knnout <- knnest(yi,xdata,k)
      outmat <- cbind(outmat,knnout$regest)
   }
   outmat <- cbind(outmat,1-apply(outmat,1,sum))
   xdata$regest <- outmat
   xdata
}

ovaknnpred <- function(xdata,predpts) {
   x <- xdata$x
   if (is.vector(predpts))
      predpts <- matrix(predpts,nrow=1)
   # need to scale predpts with the same values that had been used in
   # the training set
   ctr <- xdata$scaling[,1]
   scl <- xdata$scaling[,2]
   predpts <- scale(predpts,center=ctr,scale=scl)
   tmp <- get.knnx(x,predpts,1)
   idx <- tmp$nn.index
   regest <- xdata$regest[idx,]
   apply(regest,1,which.max) - 1
}
```

Recall that **xdata** is the output of **preprocessx()**, which determines the nearest neighbors and so on. We call the basic kNN function **knnest()** for each class, i.e. $Y = i,\ i = 0, 1, ..., m-2$ (the probabilities for class $m-1$ are obtained by subtraction from 1). These results are tacked onto **xdata**, and then input to **ovaknnpred()** whenever we need to do a prediction.

And here is the run:

```
> xdata <- preprocessx(vert[,-7],50)
> ovout <- ovaknntrn(vert[,7],xdata,3,50)
> predy <- ovaknnpred(ovout,vert[,-7])
> # proportion correctly classified
> mean(predy == vert$V7)
[1] 0.8
```

Again, this was without the benefit of cross-validation, but about the same as for LDA.

5.5.4 Multinomial Logistic Model

Within the logit realm, one might also consider *multinomial logistic regression*. Here one makes the assumption that the m sets of coefficients $\beta_i,\ i > 0$ are the same across all classes, with only β_0 varying from class to class. Recall from Section 4.4.5 that the logistic model implies that interclass boundaries are linear, i.e. hyperplanes; the multinomial logit model assumes these are parallel, as they are in the LDA case. This is a very stringent assumption, so this model may perform poorly.

5.5.5 The Verdict

If LDA's assumptions hold, especially the one positing the same covariance matrix for "X" within each class, it can be a powerful tool. However, that assumption rarely holds in practice.

As noted, the LDA setting for two classes implies that of the logistic model. Thus the latter (and not in multinomial form) is more general and "safer" than LDA. Accordingly, LDA is less widely used than in the past, but it gives us valuable perspective on the multiclass problem.

5.6 Classification Via Density Estimation

Since classification amounts to a regression problem, we could use nonparametric regression methods such as k-Nearest Neighbor if we desire a nonparametric approach, as above. However, Equations (5.4) and (5.7) suggest that one approach to the classification problem would be to estimate the within-class densities f_i. Actually, this approach is not commonly used, as it is difficult to get good estimates, especially if the number of predictors is large. However, we will examine it in this section anyway, as it will yield some useful insights.

5.6.1 Methods for Density Estimation

Say for simplicity that X is one-dimensional. You are already familiar with one famous nonparametric method for density estimation — the histogram! In R, we could use the **hist()** function for this, though we must remember to set the argument **freq** to FALSE, so as to have total area under the histogram equal to 1.0, as with a density. In the return value of **hist()**, the **density** component gives us $\widehat{f}_i(t)$.

There are more advanced density estimation methods, such as the *kernel* method, which is implemented in R's **density()** function in one dimension, and some other packages on CRAN do so for some small numbers of dimensions. Also, the k-Nearest Neighbor approach can be used for density estimation. All of this is explained in the Mathematical Complements section at the end of this chapter.

5.6.2 Procedure

Whenever we have a case to predict, i.e. we know its X value t_c and need to predict its Y value, we do the following:

For each $i = 0, 1, ..., m-1$:

- Collect the observations in our training data for which $Y = i$.
- Set $\widehat{\pi}_i$ to the proportion of cases for which $Y = i$, among all the cases.
- Use the X values in our collected data to form our estimated density for X within class i, $\widehat{f_i}(t_c)$.
- Guess the Y value for the new case to be whichever i yields the largest value of (5.7), with $\widehat{f_j}(t_c)$ in place of $f_j(t)$ and with $\widehat{\pi}_j$ in place of π_j.

5.7 The Issue of "Unbalanced (and Balanced) Data"

Here will discuss a topic that is usually glossed over in treatments of the classification problem, and indeed is often misunderstood in the machine learning (ML) research literature, where one sees much questionable hand-wringing over "the problem of unbalanced data." On the contrary, the real problem is often that the data are *balanced.*

For concreteness and simplicity, consider the two-class problem of predicting whether a customer will purchase a certain item ($Y = 1$) or not ($Y = 0$), based on a single predictor variable, X, the customer's age. Suppose also that most of the purchasers are older.

5.7.1 Why the Concern Regarding Balance?

Though one typically is interested in the overall rate of incorrect classification, we may also wish to estimate rates of "false positives" and "false negatives." In our customer purchasing example, for instance, we wish to ask, What percentage of the time do we predict that the customer does not purchase the item, among cases in which the purchase actually is made? One problem is that, although our overall misclassification rate is low, we may do poorly on a conditional error rate of this sort. This may occur, for example, if we have unbalanced data, as follows.

Suppose only 1.5% of the customers in the population opt to buy the product.[8] The concern among some ML researchers and practitioners is that, with random sampling (note the qualifier), the vast majority of the data in our sample will be from the class $Y = 0$, thus giving us "unbalanced" data. Then our statistical decision rule may not predict the other class well, and indeed may just predict almost everything to be Class 0. Let's address this concern.

Say we are using a logit model. If the model is accurate throughout the range of X, unbalanced data may not be such a problem. If most of our data have their X values in a smaller subrange, this will likely increase standard errors of the estimated regression coefficients, but not be a fundamental issue.

On the other hand, say we do classification using nonparametric density estimation. Since even among older customers, rather few buy the product, we won't have much data from Class 1, so our estimate of $\widehat{f_1}$ probably won't be very accurate. Thus Equation (5.5) then suggests we have a problem. Nevertheless, short of using a parametric model, there really is no solution.

Ironically, a more pressing issue is that we may have data that is *too* balanced, the subject of our next section.

5.7.2 A Crucial Sampling Issue

In this chapter, we have often dealt with expressions such as $P(Y = 1)$ and $P(Y = 1 \; |X = t)$. These seem straightforward, but actually they may be undefined, due to our sampling design, as we'll see here.

In our customer behavior context, $P(Y = 1)$ is the *un*conditional probability that a customer will buy the given item. If it is equal to 0.12, for example, that means that 12% of all customers purchase this item. By contrast, $P(Y = 1 \mid X = 38)$ is a *conditional* probability, and if it is equal to 0.18, this would mean that *among all people of age 38, 18% of them buy the item.*

The quantities $\pi = P(Y = 1)$ and $1 - \pi = P(Y = 0)$ play a crucial role, as can be seen immediately in (5.3) and (5.4). Let's take a closer look at this. Continuing our customer-age example, X (age) has a continuous distribution, so (5.4) applies. Actually, it will be more useful to look at the equivalent equation, (5.5).

[8] As mentioned earlier in the book, in some cases it may be difficult to define a target population, even conceptually. There is not much that can be done about this, unfortunately.

5.7.2.1 It All Depends on How We Sample

Say our training data set consists of records on 1000 customers. Let N_1 and N_0 denote the number of people in our data set who did and did not purchase the item, with $N_1 + N_0 = 1000$. If our data set can be regarded as a statistical random sample from the population of all customers, then we can estimate π from the data. If for instance 141 of the customers in our sample purchased the item, then we would set

$$\widehat{\pi} = \frac{N_1}{1000} = 0.141 \tag{5.22}$$

The trouble is, though, that the expression $P(Y = 1)$ may not even make sense with some data. Consider two sampling plans that may have been followed by whoever assembled our data set.

(a) He sampled 1000 customers from our full customer database.[9]

(b) He opted to sample 500 customers from those who purchased the item, and 500 from those who did not buy it.

Say we are using the density estimation approach to estimate $P(Y \mid X = t)$, in (5.5). In sampling scheme (a), N_1 and N_0 are random variables, and as noted we can estimate π by the quantity $N_1/1000$. But in sampling scheme (b), we have no way to estimate π from our data.

Or, suppose we opt to use a logit model here. It turns out that we will run into similar trouble in sampling scheme (b), as follows. From (4.27) and (5.5), write the population relation

$$\beta_0 + \beta_1 t_1 + ... + \beta_p t_p = -\ln((1-\pi)/\pi) - \ln[f_0(t)/f_1(t)] \tag{5.23}$$

where $t = (t_1, ..., t_p)'$. Here one can see that if one switches sampling schemes, then $\beta_i, i > 0$ will not change, because the f_i are *within-class* densities; only β_0 changes. Indeed, our logit-estimation software will "think" that π_1 and π_0 are equal (or roughly so, since we just have a sample, not the population distribution), and thus produce the wrong constant term.

In other words:

[9] Or, this was our entire customer database, which we are treating as a random sample from the population of all customers.

> Under sampling scheme (b), we are obtaining the wrong $\widehat{\beta}_0$, though the other $\widehat{\beta}_i$ are correct.

If our goal is merely Description rather than Prediction, this may not be a concern, since we are usually interested only in the values of $\beta_i, i > 0$. But if Prediction is our goal, as we assuming in this chapter, we do have a serious problem, since we will need all of the estimated coefficients in order to estimate $P(Y|X = t)$ in (4.27).

A similar problem arises if we use the k-Nearest Neighbor method. Suppose for instance that the true value of π is low, say 0.06, i.e. only 6% of customers buy the product. Consider estimation of $P(Y \mid X = 38)$. Under the k-NN approach, we would find the k closest observations in our sample data to 38, and estimate $P(Y \mid X = 38)$ to be the proportion of those neighboring observations in which the customer bought the product. The problem is that under sampling scenario (b), there will be many more among those neighbors who bought the product than there "should" be. Our analysis won't be valid.

So, all the focus on unbalanced data in the literature is arguably misplaced. As we saw in Section 5.7.1, it is not so much of an issue in the parametric case, and in any event there really isn't much we can do about it. At least, things do work out as the sample size grows. By contrast, with sampling scheme (b), we have a permanent bias, even as the sample size grows.

Scenario (b) is not uncommon. In the UCI Letters Recognition data set mentioned earlier for instance, there are between 700 and 800 cases for each English capital letter, which does not reflect that wide variation in letter frequencies. The letter 'E', for example, is more than 100 times as frequent as the letter 'Z', according to published data (see below).

Fortunately, there are remedies, as we will now see.

5.7.2.2 Remedies

As noted, use of "unnaturally balanced" data can seriously bias our classification process. In this section, we turn to remedies.

It is assumed here that we have an external data source for the class probabilities π_i. For instance, in the English letters example above, there is much published data, such as at the Web page Practical Cryptography.[10] They find that $\pi_A = 0.0855$, $\pi_B = 0.0160$, $\pi_C = 0.0316$ and so on.

[10] *http://practicalcryptography.com/cryptanalysis/ letter-frequencies-various-languages/english-letter-frequencies/.*

So, if we do have external data on the π_i (or possibly want to make some "what if" speculations), how do we adjust our code output to correct the error?

For LDA, R's **lda()** function does the adjustment for us, using its **priors** argument. That code is based on the relation (4.40), which we now see is a special case of (5.23).

The latter equation shows how to deal with the logit case as well: We simply adjust the $\widehat{\beta}_0$ that **glm()** gives us as follows.

(a) Add $\ln(N_0/N_1)$.

(b) Subtract $\ln[(1-\pi)/\pi)]$, where π is the true class probability.

Note that for an m-class setting, we estimate m logistic regression functions, adjusting $\widehat{\beta}_0$ in each case. The function **ovalogtrn()** now will include an option for this:

```
ovalogtrn <- function (m, trnxy, truepriors = NULL)
{
    p <- ncol(trnxy)
    x <- as.matrix(trnxy[, 1:(p - 1)])
    y <- trnxy[, p]
    outmat <- NULL
    for (i in 0:(m - 1)) {
        ym <- as.integer(y == i)
        betahat <- coef(glm(ym ~ x, family = binomial))
        outmat <- cbind(outmat, betahat)
    }
    if (!is.null(truepriors)) {
        tmp <- table(y)
        wrongpriors <- tmp/sum(tmp)
        outmat[1, ] <- outmat[1, ]
           - log((1 - truepriors)/truepriors)
           + log((1 - wrongpriors)/wrongpriors)
    }
    outmat
}
```

What about nonparametric settings? Equation (5.5) shows us how to make the necessary adjustment, as follows:

(a) Our software has given us an estimate of the left-hand side of that equation.

(b) We know the value that our software has used for its estimate of $(1-\pi)/\pi$, which is N_0/N_1.

(c) Using (a) and (b), we can solve for the estimate of $f_o(t)/f_1(t)$.

(d) Now plug the correct estimate of $(1-\pi)/\pi$, and the result of (c), back into (5.5) to get the proper estimate of the desired conditional probability.

Code for this is straightforward:

```
classadjust <- function(econdprobs,wrongratio,trueratio) {
   fratios <- (1 / econdprobs - 1) * (1 / wrongratio)
   1 / (1 + trueratio * fratios)
}
```

Note that if we are taking the approach described in the paragraph labeled “A variation” in Section 1.8.2, we do this adjustment only at the stage in which we fit the training data. No further adjustment at the prediction stage is needed.

5.8 Example: Letter Recognition

Let’s try out the kNN analysis on the letter data. First, some data prep:

```
> library(mlbench)
> data(LetterRecognition)
> lr <- LetterRecognition
> # code Y values
> lr[,1] <- as.numeric(lr[,1]) - 1
> # training and test sets
> lrtrn <- lr[1:14000,]
> lrtest <- lr[14001:20000,]
```

As discussed earlier, this data set has approximately equal frequencies for all the letters, which is unrealistic. The **regtools** package contains the correct frequencies, obtained from the Practical Cryptography Web site cited before. Let’s load those in:

```
> wrongpriors <- tmp / sum(tmp)
> data(ltrfreqs)
> ltrfreqs <- ltrfreqs[order(ltrfreqs[,1]),]
> truepriors <- ltrfreqs[,2] / 100
```

(Recall from Footnote 2 that the term *priors* refers to class probabilities, and that word is used both by frequentists and Bayesians. It is not "Bayesian" in the sense of subjective probability.)

So, here is the straightforward analysis, taking the letter frequencies as they are, with 50 neighbors:

```
> trnout <- ovaknntrn(lrtrn[,1],xdata,26,50)
> ypred <- ovaknnpred(trnout,lrtest[,-1])
> mean(ypred == lrtest[,1])
[1] 0.8641667
```

In light of the fact that we have 26 classes, 86% accuracy is pretty good. But it's misleading: We did take the trouble of separating into training and test sets, but as mentioned, the letter frequencies are unrealistic. How well would our classifier do in the "real world"? To simulate that, let's create a second test set with correct letter frequencies:

```
> newidxs <- sample(0:25,6000,replace=T,prob=truepriors)
> lrtest1 <- lrtest[newidxs,]
```

Now we can try our classifier on this more realistic data:

```
> ypred <- ovaknnpred(trnout,lrtest1[,-1])
> mean(ypred == lrtest1[,1])
[1] 0.7543415
```

Only about 75%. But in order to prepare for the real world, we can make use of the **truepriors** argument in **ovaknntrn()**

```
> trnout1 <- ovaknntrn(lrtrn[,1],xdata,26,50,truepriors)
> ypred <- ovaknnpred(trnout1,lrtest1[,-1])
> mean(ypred == lrtest1[,1])
[1] 0.8787988
```

Ah, very nice!

5.9 Mathematical Complements

5.9.1 Nonparametric Density Estimation

5.10 Bibliographic Notes

5.11 Further Exploration: Data, Code and Math Problems

Exercises

1. Here we will consider the "OVA and AVA" approaches to the multiclass problem (Section 5.4), using the UCBAdmissions data set that is built in to R. This data set comes in the form of a counts table, which can be viewed in proportion terms via

```
UCBAdmissions / sum(UCBAdmissions)
```

For the sake of this experiment, let's take those cell proportions to be population values, so that for instance 7.776% of all applicants are male, apply to departmental program F and are rejected. The accuracy of our classification process then is not subject to the issue of variance of estimators of logistic regression coefficients or the like.

(a) Which would work better in this population, OVA or AVA, say in terms of overall misclassification rate?

[Computational hint: First convert the table to an artificial data frame:

```
ucbd <- as.data.frame(UCBAdmissions)
```

]

(b) Write a general function

```
ovaavatbl <- function(tbl,yname)
```

that will perform the computation in part (a) for any table **tbl**, with the class variable having the name **yname**, returning the two misclassification rates. Note that the name can be retrieved via

```
names(attr(tbl,'dimnames'))
```

2. Consider the two-class classification problem with scalar predictor.

(a) Show that if the within-class distributions are exponential, the logistic model again is valid.

(b) Do the same for the Poisson case.

(c) Find general conditions in (4.20) that imply the logit model.

Chapter 6

Model Fit: Assessment and Improvement

The Box quote from our first chapter is well worth repeating:

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

We have quite a bit of powerful machinery to fit parametric models. But are they any good on a given data set? We'll discuss this subject here in this chapter.

6.1 Aims of This Chapter

Most regression books have a chapter on *diagnostics*, methods for assessing model fit and checking assumptions needed for statistical inference (confidence intervals and significance tests).

In this chapter we concerned only with the model itself. Does, for instance, the model assume a linear relationship of the response variable with the predictors, when it actually is nonlinear? Are there extreme or erroneous observations that mar the fit of our model?

We are *not* concerned here with assumptions that only affect inference, as those were treated in Chapter 2.[1]

6.2 Methods

There is a plethora of diagnostic methods! Entire books could and have been written on this topic. I treat only a few such methods here — some classical, some of my own — with the choice of methods presented stemming from these considerations:

- This book generally avoids statistical methods that rely on assuming that our sample data is drawn from a normally distributed population.[2] Accordingly, the material here on unusual observations does not make such an assumption.
- Intuitive clarity of a method is paramount. If a method can't be explained well to say, a reasonably numerate but nonstatistician client, then I prefer to avoid it.

6.3 Notation

Say we have data $(X_i, Y_i),\ i = 1, ..., n$. Here the X_i are p-component vectors,

$$X_i = (X_i^{(1)}, ..., X_i^{(p)})' \tag{6.1}$$

and the Y_i are scalars (including the case $Y = 0, 1, ..., m-1$ in classification applications). We typically won't worry too much in this chapter whether the n observations are independent.

As usual, let

$$\mu(t) = E(Y \mid X = t) \tag{6.2}$$

be our population regression function, and let $\widehat{\mu}(t)$ its estimate from our sample data.

[1] The assumption of statistical independence was not covered there, and indeed will not be covered elsewhere in the book, in the sense of methods for assessing independence. The issue will of course be discussed in individual examples.

[2] "Rely on" here means that the method is not robust to the normality assumption.

6.4 Goals of Model Fit-Checking

What do we really mean when we ask whether a model fits a data set well? Our answer ought to be as follows:

> **Possible Fit-Checking Goal:**
>
> Our model fits well if $\widehat{\mu}(t)$ is near $\mu(t)$ for all t, or at least for $t = X_1, ..., X_n$.

That criterion is of course only conceptual; we don't know the values of $\mu(t)$, so it's an impossible criterion to verify. Nevertheless, it may serves well as a goal, and our various model-checking methods will be aimed at that goal.

Part of the answer to our goals question goes back to the twin regression goals of Prediction and Description. We'll explore this in the following sections.

6.4.1 Prediction Context

If our regression goal is Prediction and we are doing classification, our above Fit-Checking Goal may be much too stringent. Say for example $m = 2$. If $\mu(t)$ is 0.9 but $\widehat{\mu}(t = 0.62)$, we will still make the correct guess, $Y = 1$, even though our regression function estimate is well off the *mark*. Similarly, if $\mu(t)$ is near 0 (or less than 0.5, actually), we will make the proper guess for Y as long as our estimated value $\widehat{\mu}(t)$ is under 0.5.

Still, other than the classification case, the above Fit-Checking Goal is appropriate. Errors in our estimate of the population regression function will impact our ability to predict, no matter what the size is of the true regression function.

6.4.2 Description Context

Good model fit is especially important when our regression goal is Description. We really want assurance that the estimated regression coefficients represent the true regression function well. since we will be using them to describe the underlying process.

6.4.3 Center vs. Fringes of the Data Set

Consider a Prediction setting, in the classification case. In Section 6.4.1 above, we saw that actually can afford a rather poor model fit in regions of the predictor space in which the population regression function is near 1 or 0.

The same reasoning shows, though, that having a good fit in regions where $\mu(t)$ is mid-range, say in (0.25,0.75) *is* important. If $\widehat{\mu}(t)$ and $\mu(t)$ are on opposite sites of the number 0.5 (i.e. one below 0.5 and the other above), we will make the wrong decision (even though we make still be lucky and guess Y correctly).

In classification contexts, the p-variate density of X is often "mound-shaped," if not bell-shaped, within each class. (In fact, many clustering algorithms are aimed at this situation.) For such data, the regions of most sensitivity in the above sense will be the lines/curves separating the pairs of mounds. (Recall Figure 5.1.) The fringes of the data set, far from these pairwise boundaries, will be regions in which model fit is less important, again assuming a Prediction goal.

In regression contexts (continous Y, count data etc.), the full data set will tend to be mound-shaped. Here good estimation will be important for Predicition and Description throughout the entire region. However, one must keep in mind that model fit will typically be better near the center of the data than at the fringes — and that the observations at the fringes typically have the heaviest impact on the estimated coefficients. This latter consideration is of course of great import in the Description case.

We will return to these considerations at various points in this chapter.

6.5 Example: Currency Data

Fong and Ouliaris (1995) do an analysis of relations between currency rates for Canada, Germany, France, the UK and Japan (pre-European Union days). Do they move together? Let's look at predicting the Japanese it yen from the others.

This is time series data, and the authors of the above paper do a very sophisticated analysis along those lines. So, the data points, such for the *pound*, are not independent through time. But since we are just using the data as an example and won't be doing inference (confidence intervals and significance tests), we will not worry about that here.

Let's start with a straightforward linear model:

```
> curr <- read.table('EXC.ASC',header=TRUE)
> fout <- lm(Yen ~ .,data=cur1)
> head(curr)
     Can  Mark Franc   Pound   Yen
1 0.9770 2.575 4.763 0.41997 301.5
2 0.9768 2.615 4.818 0.42400 302.4
3 0.9776 2.630 4.806 0.42976 303.2
4 0.9882 2.663 4.825 0.43241 301.9
5 0.9864 2.654 4.796 0.43185 302.7
6 0.9876 2.663 4.818 0.43163 302.5
> summary(fout)
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  102.855     14.663   7.015 5.12e-12 ***
Can          -45.941     11.979  -3.835 0.000136 ***
Mark         147.328      3.325  44.313  < 2e-16 ***
Franc        -21.790      1.463 -14.893  < 2e-16 ***
Pound        -48.771     14.553  -3.351 0.000844 ***
...
Multiple R-squared: 0.8923, Adjusted R-squared: 0.8918
```

Not surprisingly, this model works well, with an adjusted R-squared value of about 0.89. The signs of the coefficients are interesting, with the it yen seeming to fluctuate opposite to all of the other currencies except for the German *mark*. Of course, professional financial analysts (*domain experts*, in the data science vernacular) should be consulted as to the reasons for such relations, but here we will proceed without such knowledge.

It may be helpful to scale our data so as to better understand the roles of the predictors, though, so as to put make all the predictors commensurate. Each predictor will be divided by its standard deviation (and have its mean subtracted off), so all the predictors have standard deviation 1.0:

```
> curr1 <- curr
> curr1[,-5] <- scale(curr1[,-5])
> fout1 <- lm(Yen ~ .,data=curr1)
> summary(fout1)
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 224.9451     0.6197 362.999  < 2e-16 ***
```

```
Can          -5.6151     1.4641   -3.835 0.000136 ***
Mark         57.8886     1.3064   44.313  < 2e-16 ***
Franc       -34.7027     2.3301  -14.893  < 2e-16 ***
Pound        -5.3316     1.5909   -3.351 0.000844 ***
...
```

So Germany and France appear to have the most influence.[3] This, and the signs (positive for the *mark*, negative for the *franc*), form a good example of the use of regression analysis for the Description goal.

In the next few sections, we'll use this example to illustrate the basic conceptss.

6.6 Overall Measures of Model Fit

We'll look two broad categories of fit assessment methods. The first will consist of overall measures, while the second will involve relating fit to individual predictor variables.

6.6.1 R-Squared, Revisited

We have already seen one overall measure of model fit, the R-squared value (Section **??**). As noted before, its cousin, Adjusted R-squared, is considered more useful, as it is aimed at compensating for overfitting.

For the currency data above, the two R-squared values (ordinary and adjusted) were 0.8923 and 0.8918, both rather high. Note that they didn't differ much from each other, as there were well over 700 observations, which should easily handle a model with only 4 predictors (a topic we'll discuss in Chapter 9).

Recall that R-squared, whether a population value or the sample estimate reported by **lm()**, is the squared correlation between Y and its predicted value $\mu(X)$ or $\widehat{\mu}(X)$, respectively. Thus it can be calculated for any method of regression function estimation, not just the linear model. In particular, we can apply the concept to k-Nearest Neighbor methods.

The point of doing this with kNN is that the latter in principle does not have model-fit issues. Whereas our linear model for the currency data assumes

[3] Or, at least, seem to have the strongest relation with the Yen. We cannot claim causation.

a linear relationship between the Yen and the other currencies, kNN makes no assumptions on the form of the regression function. If kNN were to have a substantially larger R-squared value than that of our linear model, then we may be "leaving money on the table," i.e. not fitting as well as we could with a more sophisticated parametric model.[4]

So, let's see what kNN tells us in the currency example:

```
> xdata <- preprocessx(curr1[,-5],25,xval=TRUE)
> kout <- knnest(curr1[,5],xdata,25)
> cor(kout$regest,curr1[,5])^2
[1] 0.9717136
```

This would indicate that, in spite of a seemingly good fit for our linear model, it does not adequately describe the currency fluctuation process.

However, we should raise a question as to whether the value of k here, 25, is a good one. Let's investigate this with the **regtools** function **kmin()**:

```
> xdata <- preprocessx(curr1[,-5],150,xval=TRUE)
> kminout <- kmin(curr1$Yen,xdata,predwrong,nk=30)
> kminout$kmin
[1] 5
```

The "best" k, as determined by a cross-validated estimate of mean squared prediction error, is reported here to be 5 (which is the smallest value this function tried). This may seem very small, but as will be discused in Chapter 10, it is to be expected in light of the high R^2 value we have with this data.

Let's rerun our R^2 calculation with this value of k:

```
> kout <- knnest(curr1[,5],xdata,5)
> cor(kout$regest,curr1[,5])^2
[1] 0.9920137
```

Even better! So, our linear model, which seemed so nice at first, is missing something. Maybe we can determine why via the methods in the following sections.

[4] We could of course simply use kNN in the first place. But this would not give us the Description usefulness of the parametric model, and also would give us a higher estimation variance.

6.6.2 Plotting Parametric Fit Against Nonparametric One

Let's plot the $\widehat{\mu}$ values of the linear model against those of kNN:

```
> parvsnonparplot(fout1,kout)
```

The result is shown in Figure 6.1. It suggests that the linear model is overestimating the regression function at times when the Yen is very low, moderately low or very high, and possibly underestimating in the moderately high range.

We must view this cautiously, though. First, of course, there is the issue of sampling variation; the apparent model bias effects here may just be sampling anomalies.

Second, kNN itself is subject to some bias at the edges of a data set. This will be discussed in detail in Section **??**, but the implications are that in the currency case kNN tends to overestimate for low values of the Yen, and underestimate at the high end. This can be addressed by doing locally-linear smoothing, an option offered by **knnest()**, but let's not use it for now.

The "hook shape" at the left end, and a "tail" in the middle suggest odd nonlinear effects, possibly some local nonlinearities, which kNN is picking up but which the linear model misses.

6.6.3 Residuals vs. Smoothing

In any regression analysis, the quantities

$$r_i = Y_i - \widehat{\mu}(X_i) \tag{6.3}$$

are traditionally called the *residual values*, or simply the *residuals.* They are of course the prediction errors we obtain when fitting our model and then predicting our Y_i from our X_i. The smaller these values are in absolute value, the better, but also we hope that they may inform us of inaccuracies in our model, say nonlinear relations between Y and our predictor variables.

In the case of a linear model, the residuals are

$$r_i = Y_i - \widehat{\beta}_0 - \widehat{\beta}_1 X_i^{(1)} - ... - \widehat{\beta}_p X_i^{(p)} \tag{6.4}$$

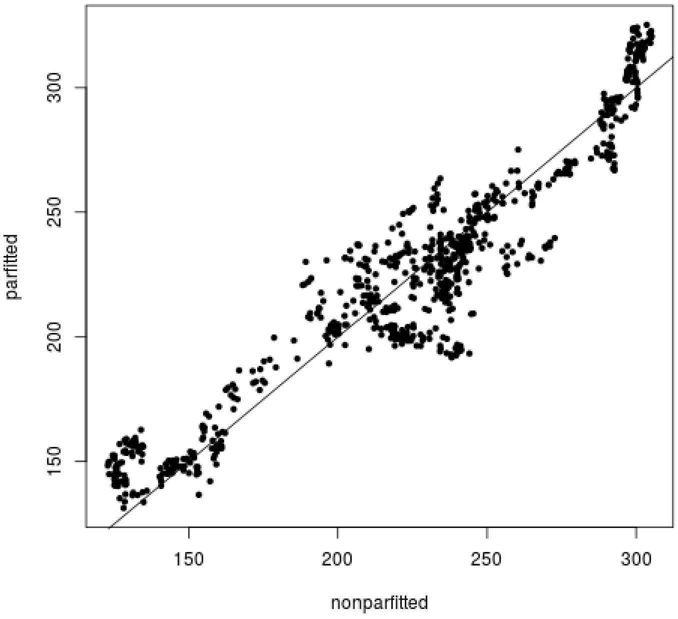

Figure 6.1: Estimation of Regression Values, Two Methods

Many diagnostic methods for checking linear regression models are based on residuals. In turn, their convenient computation typically involves first computing the *hat matrix*, about which there is some material in the Mathematical Complements section at the end of this chapter.

The generic R function **plot()** can be applied to any object of class **"lm"** (including the subclass **"glm"**). Let's do that with **fout1**:

```
> plot(fout1)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

We obtain a series of graphs, displayed sequentially. Most of them involve more intricate concepts than we'll use in this book (recall Section 6.2), but let's look at the first plot, shown in Figure 6.2. The "hook" and "tail" are visible here too.

Arguably the effects are more clearly in Figure 6.1. This is due to the fact that the latter figure is plotting smoothed values, not residuals. In other words, residuals and smoothing play complementary roles to each other: Smoothing-based plots can more easily give us "the big picture," but residuals may enable us to spot some fine details.

In any case, it's clear that the linear model does not tell the whole story.

6.7 Diagnostics Related to Individual Predictors

It may be that the relationship with the response variable Y is close to linear for some predictors $X^{(i)}$ but not for others. How might we investigate that?

6.7.1 Partial Residual Plots

We might approach this by simply plotting a scatter diagram of Y against each predictor variable. However, the relation of Y with $X^{(i)}$ may change in the presence of the other $X^{(j)}$. A more sophisticated approach may be *partial residual plots*, also known as *component + residual plots.* These would be easy to code on one's own, but the **crPlot()** function in the **car** package does the job nicely for us:

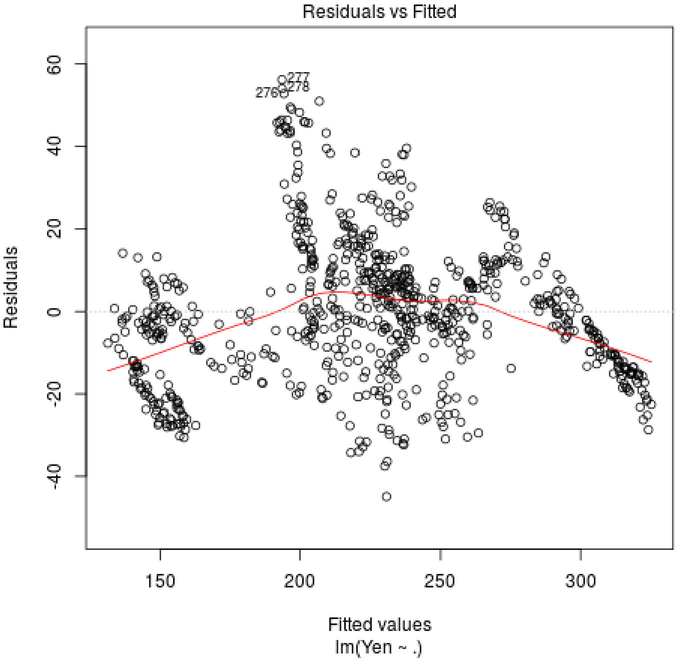

Figure 6.2: Residuas Against Linear Fitted Values

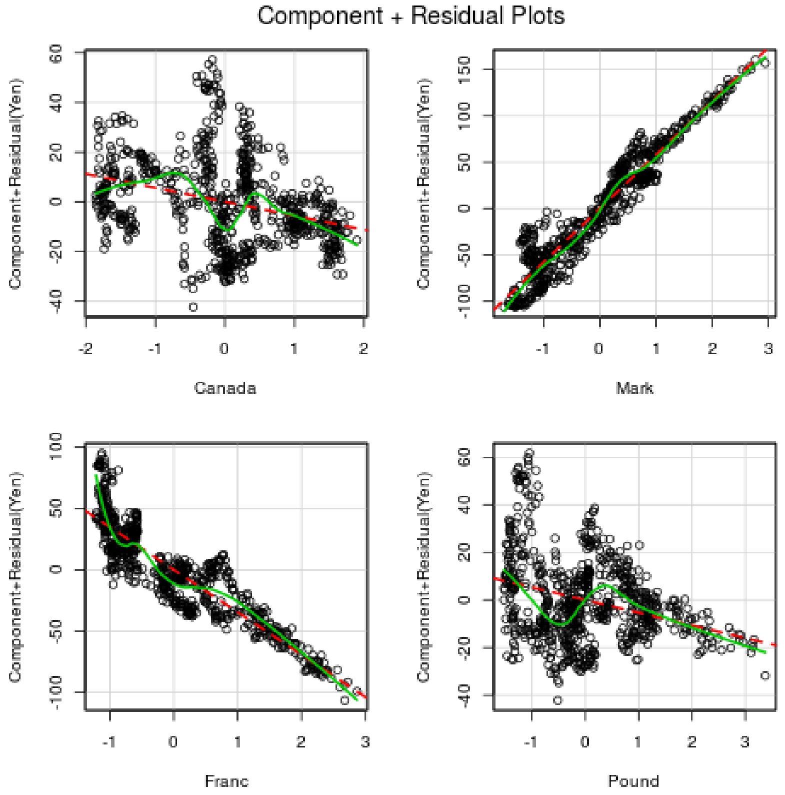

Figure 6.3: Partial Residuals Plot, Currency Data

```
> crPlots(fout1)
```

The resulting graph is shown in Figure 6.3. Before discussing these rather bizarre results, let's ask what these plots are depicting.

Here is how the partial-residual method works. The partial residuals for a predictor $X^{(j)}$ are defined to be

$$
\begin{aligned}
p_i &= r_i + \widehat{\beta}_j X_i^{(j)} && (6.5)\\
&= Y_i - \widehat{\beta}_0 - \widehat{\beta}_1 X_i^{(1)} - ... - \widehat{\beta}_1 X_i^{(j-1)} - \widehat{\beta}_1 X_i^{(j+1)} - ... - \widehat{\beta}_p X_i^{(p)} && (6.6)
\end{aligned}
$$

In other words, we've started with (6.4), but removed the linear term contributed by predictor j, i.e. removed $\widehat{\beta}_p X_i^{(j)}$. We then plot the p_i against predictor j, to try to discern a relation. In effect we are saying,

> Cancel that linear contribution of predictor j. Let's start fresh with this predictor, and see how adding it in a possibly nonlinear form might extend the collective predictive ability of the other predictors.

If the resulting graph looks nonlinear, we may profit from modifying our model to one that reflects a nonlinear relation.

In that light, what might we glean from Figure 6.3? First, we see that the only "clean" relations are the one for the *franc* and the one for the *mark*. No wonder, then, that we found earlier that these two currencies seemed to have the strongest linear relation to the Yen. There does seem to be some nonlinearity in the case of the *franc*, with a more negative slope for low *franc* values, and this may be worth pursuing, say by adding a quadratic term.

For the Canadian *dollar* and the *pound*, though, the relations don't look "clean" at all. On the contrary, the points in the graphs clump together much more than we typically encounter in scatter plots.

But even the *mark* is not off the hook (pardon the pun), as the "hook" shape noticed earlier is here for that currency, and apparently for the Canadian *dollar* as well. So, whatever odd phenomenon is at work may be related to these two currencies,

6.7.2 Plotting Nonparametric Fit Against Each Predictor

As noted, one approach would be to draw many scatter diagrams, plotting Y individually against each $X^{(i)}$. But scatter diagrams are, well, scattered. A better way is to plot the nonparametric fit against each predictor. The **regtools** function **nonparvsxplot()** does this, plotting one graph for each predictor, presented in succession with user prompts:

```
> nonparvsxplot(kout)
next plot
next plot
next plot
next plot
```

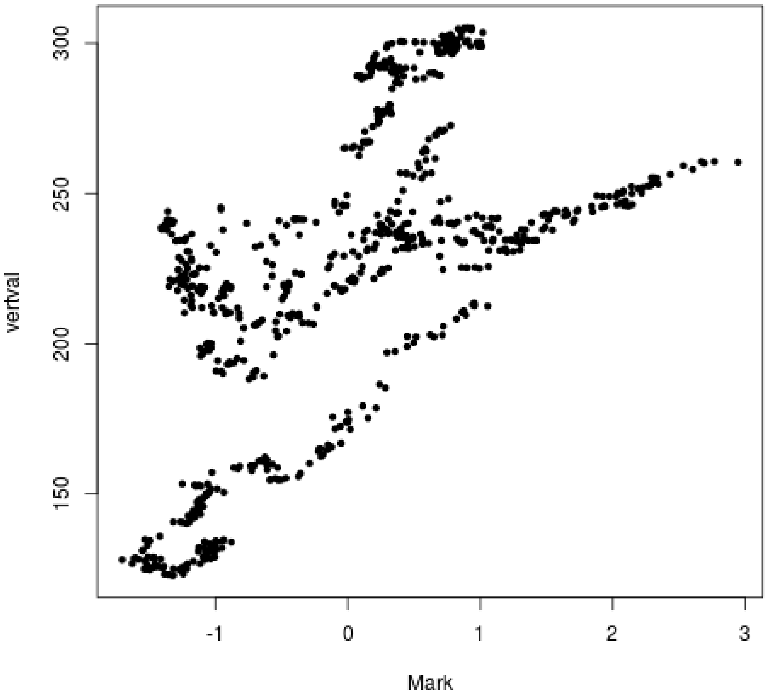

Figure 6.4: Nonparametric Fit Against the Mark

The graph for for the *mark* is shown in Figure 6.4. Oh my gosh! With the partial residual plots, the *mark* and the *franc* seemed to be the only "clean" ones. Now we see that the situation for the *mark* is much more complex. The same is true for the other predictors (not shown here). This is indeed a difficult data set.

Again, note that the use of smoothing has brought these effects into better focus, as discussed in Section 6.6.3.

6.7.3 Freqparcoord

Another graphical approach is via **freqparcoord** package, written by Yingkang Xie and me. The call

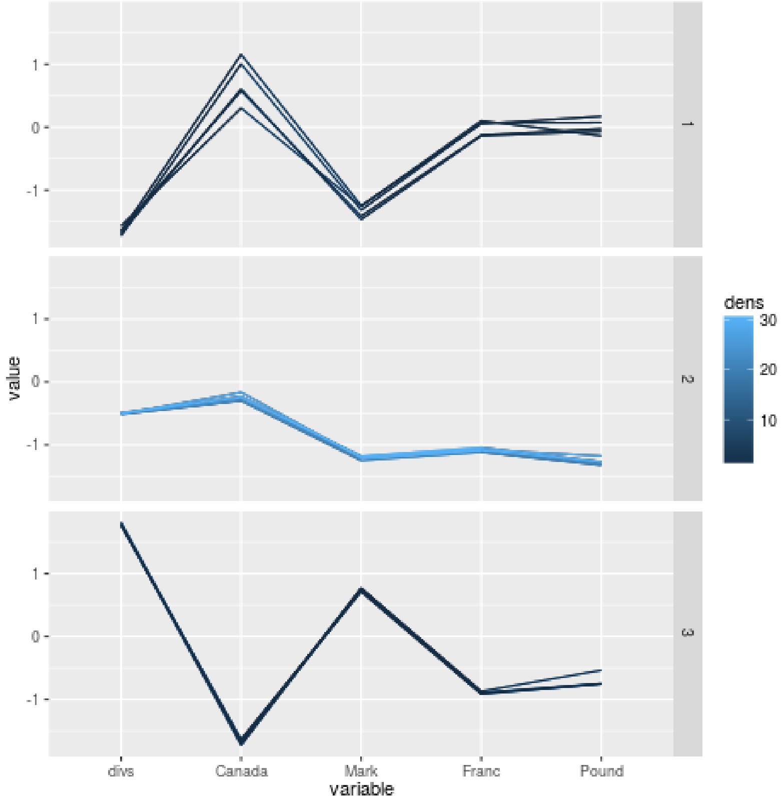

Figure 6.5: Freqparcoord Plot, Currency Data

produces the graph in Figure 6.5.

What is depicted in this graph? First, this is a *parallel coordinates plot*, which is a method for visualizing multidimensional data in a 2-dimensional graph. This approach dates back to the 1800s, but was first developed in depth in modern times; see Inselberg (2009).

The general method of parallel coordinates is quite simple. Here we draw p vertical axes, one for each variable. For each of our n data points, we draw a polygonal line from each axis to the next. The height of the line on axis j is the value of variable j for this data point. As Inselberg pointed out, in mathematical terms the plot performs a transformation mapping p-dimensional points to $p-1$-segment lines. The practical effect is that we can visualize how our p variables vary together.

However, if our number of data points n is large, our parallel coordinates will consist of a chaotic jumble of lines, maybe even with the "black screen problem," meaning that so much has been plotted that the graph is mostly black, no defining features.

The solution to that problem taken by **freqparcoord** is to plot only the most frequently-occurring lines, meaning those corresponding to the original data points having the largest estimated p-dimensional density funciton.

A function in the **freqparcoord** package, **regdiag()**, applies this to regression diagnostics. The first variable plotted, i.e. the first vertical axis, is what we call the *divergences*, meaning the differences beween the parametric and nonparametric estimates of the population regression function,

$$\widehat{\mu}_{linmod}(X_i) - \widehat{\mu}_{knn}(X_i), \;\; i = 1, ..., n \tag{6.7}$$

The other axes represent our predictor variables. Vertical measures are numbers of standard deviations from the mean of the given variable.

There are three groups, thus three subgraphs, for the upper 10%, middle 80% and lower 10% of the divergence values. So for instance the upper subgraph describes data points X_i at which the linear model greatly overestimates the true regression function.

What we see, then, is that in regions in which the linear model underestimates, the Canadian *dollar* tends to be high and the *mark* low, with an opposite relation for the region of overestimation. Note that this is not the same as saying that the correlation between those two currencies is negative; on the contrary, running **cor(curr1)** shows their correlation to be positive and tiny, about 0.01. This suggests that we might try adding a *dollar/mark* interaction term to our model, though the effect here seems mild, with peaks and valleys of only about 1 standard deviation..

6.8 Effects of Unusual Observations on Model Fit

Suppose we are doing a study of rural consumer behavior in a small country C in the developing world. One day, a few billionaires discover the idyllic beauty of rural C and decide to move there. We almost certainly would want to exclude data on these interlopers from our analysis. Second, most data contains errors. Obviously these must be excluded too, or corrected if possible. These two types of observations are sometimes collectively called *outliers*, or simply *unusual*.

Though we may hear that Bill Gates has moved to C, and we can clean the data to remove the obvious errors, such as a human height of 25 feet or a negative weight. other extreme values or errors may not jump out at us. Thus it would be useful to have methods that attempt to find such observations in some mechanical way.

6.8.1 The influence() Function

Base R includes a very handy function, **influence()**. We input an object of type **"lm"**, and it returns an R list. One of the components of that list, **coefficients**, is just what we want: It has a column for each $\widehat{\beta}_j$, and a row for each observation in our data set. Row i, column j tells us how much $\widehat{\beta}_j$ would change if observation i were deleted from the data set.[5] If the change is large, we should take a close look at observation i, to determine whether it is "unusual."

6.8.1.1 Example: Currency Data

Let's take a look:

```
> infcfs <- influence(fout1)$coef
> head(infcfs)
   (Intercept)        Can         Mark        Franc
1  -0.01538183 0.03114368 -0.009961471 -0.07634899
2  -0.02018040 0.03743135 -0.018141461 -0.09488785
3  -0.02196501 0.03583000 -0.024654614 -0.08885551
4  -0.02877846 0.02573926 -0.050914092 -0.08862127
5  -0.02693571 0.02564799 -0.046012297 -0.08261002
6  -0.02827297 0.02524051 -0.050186868 -0.08733993
         Pound
1  0.07751692
2  0.10129967
3  0.10190745
4  0.13201466
5  0.12140985
6  0.13027910
```

[5] The entire computation does not need to be done from scratch. The Sherman-Morrison-Woodbury formula provides a shortcut. See the Mathematical Complements section at the end of this chapter.

So, if we were to remove the second data point, this says that $\widehat{\beta}_0$ would decline by 0.02018040, $\widehat{\beta}_1$ would increase by 0.03743135, and so on. Let's check to be sure:

```
> coef(fout1)
(Intercept)          Can         Mark        Franc
 224.945099    -5.615144    57.888556   -34.702731
      Pound
  -5.331583
> coef(lm(Yen ~ .,data=curr1[-2,]))
(Intercept)          Can         Mark        Franc
 224.965279    -5.652575    57.906698   -34.607843
      Pound
  -5.432882
> -5.652575 + 0.037431
[1] -5.615144
```

Excellent. Now let's find which points have large influence.

A change in an estimated coefficient should be considered "large" only relative to the standard error, so let's scale accordingly, dividing each change by the standard error of the correspoding coefficient:

```
> se <- sqrt(diag(vcov(fout1)))
> infcfs <- infcfs %*% diag(1/se)
```

So, how big do the changes brought by deletions get in this data? And for which observations does this occur?

```
> ia <- abs(infcfs)
> max(ia)
[1] 0.1928661
> f15 <- function(rw) any(rw > 0.15)
> ia15 <- apply(ia,1,f15)
> names(ia15) <- NULL
> which(ia15)
 [1] 744 745 747 748 749 750 751 752 753 754 755
 [12] 756 757 758 759 760 761
```

Here we (somewhat arbitrarily) decided to identify which deletions of observations would result in an absolute change of *some* coefficient of more than 0.15.

Now this is interesting. There are 761 observations in this data set, and now we find that all of the final 18 (and more) are influential. Let's look more closely:

```
> tail(ia,5)
            [,1]      [,2]      [,3]      [,4]
757 0.05585538 0.1311110 0.1572411 0.1305925
758 0.05851087 0.1294412 0.1563013 0.1259741
759 0.05838813 0.1386614 0.1629851 0.1358875
760 0.05818730 0.1429951 0.1654354 0.1385146
761 0.05626212 0.1316884 0.1534207 0.1211305
           [,5]
757 0.021300177
758 0.015701005
759 0.020431391
760 0.019962502
761 0.006793673
```

So, the influence of these final observations was on the coefficients of the Canadian *dollar*, the *mark* and the *franc* — but not on the one for the *pound*.

Something special was happening in those last time periods. It would be imperative for us to track this down with currency experts.

Each of the observations has something like a 0.15 impact, and intuitively, removing all of these observations should cause quite a change. Let's see:

```
> curr2 <- curr1[-(744:761),]
> lm(Yen ~ .,data=curr2)
...
Coefficients:
(Intercept)       Canada         Mark
    225.780      -10.271       52.926
      Franc        Pound
    -27.126       -6.431

> fout1
...
Coefficients:
(Intercept)       Canada         Mark
    224.945       -5.615       57.889
      Franc        Pound
    -34.703       -5.332
```

These are very substantial changes! The coefficient for the Canadian currency almost doubled, and even the *pound*'s value changed almost 30%. That latter is a dramatic difference, in view of the fact that each individual observation had only about a 2% influence on the *pound*.

6.9 Automated Outlier Resistance

The term *robust* in statistics generally means the ability of methodology to withstand problematic conditions. Linear regression models, for instance, are said to be "robust to the normality" assumption, meaning that (at least large-sample) inference on the coefficients with work well even though the distribution of Y given X is not normal.

Here we are concerned with robustness of regression models to outliers. Such methods are called *robust regression*. There are many such methods, one of which is *median regression*, to be discussed next.

6.9.1 Median Regression

Suppose we wish to estimate the mean of some variable in some population, but we are concerned about unusual observations. As an alternative, we might consider estimating the median value of the variable, which will be much less sensitive to unusual observations. Recall our hypothetical example earlier, in which we were interested in income distributions. If one very rich person moves into the area, the mean may be affected substantially — but the median would likely not change at all.

We thus say that the median is robust to unusual data points. One can do the same thing to make regression analysis robust in this sense.

It turns out (see the Mathematical Complements section at the end of this chapter) that

$$\nu(t) = \text{median}(Y \mid X = t) = \text{argmin}_m E(|Y - m| \, |X = t) \tag{6.8}$$

In other words, in contrast to the regression function, i.e. the conditional mean, which minimizes mean squared prediction error, the conditional median minimizes mean *absolute* error.

Remember, as with regression, we are estimating an entire function here, as t varies. A nonparametric approach to this would be to use **knnest()** with **nearf** set to

```
function(predpt,nearxy)
```

```
{
   ycol <- ncol(nearxy)
   median(nearxy[, ycol])
}
```

However, in this chapter we are primarily concerned with parametric models. So, we might, in analogy to the linear regression model, make the assumption that (6.8) has the familiar form

$$\nu(t) = \beta_0 + \beta_1 t_1 + ... + \beta_p t_p \tag{6.9}$$

Solving this at the sample level is a *linear programming* problem, which has been implemented in the CRAN package **quantreg**. As the package name implies, we can estimate general conditional quantile functions, not just the conditional median. The argument **tau** of the **rq()** function specifies what quantile we want, with the value 0.5, i.e. the median, being the default.

It is important to understand that $\nu(t)$ is *not* the regression function, i.e. not the conditional mean. Thus **rq()** is not estimating the same quantity as is **lm()**. Thus the term *quantile regression*, in this case the term *median regression*, is somewhat misleading here. But we can use $\nu(t)$ as an alternative to $\mu(t)$ in one of two senses:

(a) We may believe that $\nu(t)$ is close to $\mu(t)$. They will be exactly the same, of course, if the conditional distribution of Y given X is symmetric, at least if the unusual observations are excluded. (This is an assumption we can assess by looking at the residuals.)

(b) We may take the point of view that the conditional median is just as meaningful as the conditional mean (no pun intended this time), so why not simply model $\nu(t)$ in the first place?

Sense (a) above will be particularly relevant here.

6.9.2 Example: Currency Data

Let's apply **rq()** to the currency data:

```
> qout <- rq(Yen ~ .,data=curr1)
> qout
...
Coefficients:
```

```
(Intercept)         Can         Mark        Franc
 224.517899  -11.038238    53.854005   -27.443584
      Pound
  -5.320035
...
> fout1
...
Coefficients:
(Intercept)         Can         Mark
    224.945      -5.615       57.889
      Franc       Pound
    -34.703      -5.332
```

The results are strikingly similar to what we obtained in Section 6.8.1.1 by calling **lm()** with the bad observations at the end of the data set removed. In other words,

> Median regression can be viewed as an automated method for removing (the effects of) the unusual data points.

6.10 Example: Vocabulary Acquisition

The Wordbank data, http://wordbank.stanford.edu/, concerns child vocabulary development, in not only English but also a number of other languages, such as Cantonese and Turkish. These are mainly toddlers, ages from about a year to 2.5 years.

Let's read in the English set:

```
> engl <- read.csv('English.csv')
> engl <- engl[,c(2,5:8,10)]
> encc <- engl[complete.cases(engl),]
> nrow(encc)
[1] 2741
```

One of the variables is Birth Order. Let's make it numeric:

```
> z <- engcc$birth_order
> numorder <- vector(length=length(z))
> for (i in 1:length(z)) {
+     numorder[i] <- if(z[i]=='First') 1 else
+            if(z[i]=='Second') 2 else
+            if(z[i]=='Third') 3 else
+            if(z[i]=='Fourth') 4 else
+            if(z[i]=='Fifth') 5 else
+            if(z[i]=='Sixth') 6 else
+            if(z[i]=='Seventh') 7 else 8
+ }
> encc$birthord <- numorder
```

Let's convert the variable on the mother's education to a rough number of years of school:

```
> z <- encc$mom_ed
> momyrs <- vector(length=length(z))
> for (i in 1:length(z)) {
+    momyrs[i] <- if(z[i]=='Primary') 4 else
+                 if(z[i]=='Some Secondary') 10 else
+                 if(z[i]=='Secondary') 12 else
+                 if(z[i]=='Some College') 14 else
+                 if(z[i]=='College') 16 else
+                 if(z[i]=='Some Gradute') 18 else 20
+ }
> encc$momyrs <- momyrs
```

Also, create the needed dummy variables, for gender and nonwhite categories:

```
> encc$male <- as.numeric(encc$sex=='Male')
> encc$asian <- as.numeric(encc$ethnicity=='Asian')
> encc$black <- as.numeric(encc$ethnicity=='Black')
> encc$latino <- as.numeric(encc$ethnicity=='Hispanic')
> encc$othernonwhite <- as.numeric(encc$ethnicity=='Other')
```

Running **knnest()** (not shown), there seemed to be an approximately linear relation between vocabulary and age (for the age range studied). Let's run a linear regression analysis:

```
> encc1 <- encc[,-c(2:5)]
> summary(lm(vocab ~ .,data=encc1))
...
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -473.0402    23.1001 -20.478  < 2e-16 ***
age             33.3905     0.6422  51.997  < 2e-16 ***
birthord       -20.5399     3.0885  -6.650 3.52e-11 ***
```

```
male           -49.0196      5.4840   -8.939  < 2e-16 ***
asian          -15.7985     17.2940   -0.914 0.361048
black            1.0940     10.2360    0.107 0.914890
othernonwhite  -54.1384     15.1274   -3.579 0.000351 ***
latino         -75.6905     13.0155   -5.815 6.75e-09 ***
momyrs           3.6746      0.9381    3.917 9.18e-05 ***
...
Multiple R-squared:  0.5132,     Adjusted R-squared:  0.5118
```

So, the kids seemed to be learning about 30 characters per month during the studied age range. Latino kids seem to start from a lower English base, possibly due to speaking Spanish at home. Consistent with the general notion that girls develop faster than boys, the latter have a lower base. Having a lot of older siblings also seems to be related to a lower base, possibly due to the child being one of several competing for the parents' attention. Having a mother with more education had a modest positive effect.

A word on the standard errors: As each child was measured multiple times as he/she aged, the observations are not independent, and the true standard errors are larger than those given.

Let's try median regression instead:

```
> rq(vocab ~ .,data=encc1)
...
Coefficients:
  (Intercept)           age      birthord          male
asian
  -574.685185     37.777778    -19.425926    -44.925926
-21.944444
        black othernonwhite        latino        momyrs
   -13.148148    -50.462963    -68.629630      3.638889
...
```

The robust results here are similar to what we obtained earlier, but with some modest shifts.

It is often worthwhile to investigate other quantiles than the median. Trying that for age only:

```
> plot(c(12,30),c(0,800),type = "n", xlab = "age", ylab = "vocab")
> abline(coef(rq(vocab ~ age,data=encc)))
> abline(coef(rq(vocab ~ age,data=encc,tau=0.9)))
> abline(coef(rq(vocab ~ age,data=encc,tau=0.1)))
```

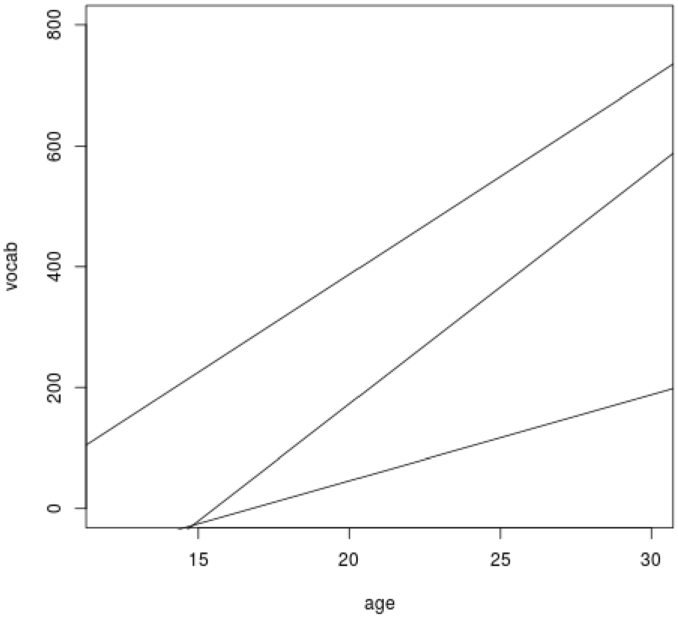

Figure 6.6: Vocabulary vs. Age

As seen in Figure 6.6, the middle-level children start out knowin much fewer words than the most voluble ones, but narrow the gap over time. By contrast, the kids with smaller vocabularies start out around the same level as the middle kids, but actually lose ground over time, suggesting that educational interventions may be helpful.

6.11 Improving Fit

The currency example seemed so simple at first, with a very nice adjusted R-squared value of 0.89, and with the *yen* seeming to have a clean linear relation with the *franc* and the *mark*. And yet we later encountered some troubling aspects to this data.

First we noticed that the adjusted R-squared value for the kNN fit was even better, at 0.98. Thus there is more to this data than simple linear relationships. Later we found that the last 18 data points, possibly more, have an inordinate influence on the $\widehat{\beta}_j$. This too could be a reflection of nonlinear relationships between the currencies. The plots exhibited some strange, even grotesque, relations.

So, let's see what we might do to *improve* our parametric model.

6.11.1 Deleting Terms from the Model

Predictors with very little relation to the response variable may actually degrade the fit, and we should consider deleting them. Tnis topic is treated in depth in Chapter 9.

6.11.2 Adding Polynomial Terms

Our current model is linear in the variables. We might add second-degree terms. Note that this means not only squares of the variables, but products of pairs of them. The latter may be important, in view of our comment in Section 6.7.3 that it might be useful to add a Canadian *dollar/mark* interaction term to our model.

6.11.2.1 Example: Currency Data

Let's add squared terms for each variable, and try the interaction term as well. Here's what we get:

```
> curr2 <- curr1
> curr2$C2 <- curr2$Canada^2
> curr2$M2 <- curr2$Mark^2
> curr2$F2 <- curr2$Franc^2
> curr2$P2 <- curr2$Pound^2
> curr2$CM <- curr2$Canada* curr2$Mark
> summary(lm(Yen ~ .,data=curr2))
...
Coefficients:
               Estimate Std. Error t value
(Intercept)  223.575386   1.270220 176.013
Can           -8.111223   1.540291  -5.266
Mark          50.730731   1.804143  28.119
Franc        -34.082155   2.543639 -13.399
Pound         -3.100987   1.699289  -1.825
```

```
C2            -1.514778   0.848240  -1.786
M2            -7.113813   1.175161  -6.053
F2            11.182524   1.734476   6.447
P2            -1.182451   0.977692  -1.209
CM             0.003089   1.432842   0.002
            Pr(>|t|)
(Intercept)  < 2e-16 ***
Can         1.82e-07 ***
Mark         < 2e-16 ***
Franc        < 2e-16 ***
Pound         0.0684 .
C2            0.0745 .
M2          2.24e-09 ***
F2          2.04e-10 ***
P2            0.2269
CM            0.9983
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1
...
Multiple R-squared:  0.9043,    Adjusted R-squared:  0.9032
```

Adjusted R-squared increased only slightly. And this was despite the fact that two of the squared-variable terms were “highly significant,” adorned with three asterisks, showing how misleading significance testing can be. The interaction term came out tiny, 0.003089. So, kNN is still the winner here.

6.11.2.2 Example: Programmer/Engineer Census Data

Let’s take another look at the Census data on programmers and engineers in Silicon Valley, first introduced in Section 1.11.1.

We run

```
> data(prgeng)
> pe <- prgeng  # see ?knnest
> # dummies for MS, PhD
> pe$ms <- as.integer(pe$educ == 14)
> pe$phd <- as.integer(pe$educ == 16)
> # computer occupations only
> pecs <- pe[pe$occ >= 100 & pe$occ <= 109,]
> pecs1 <- pecs[,c(1,7,9,12,13,8)]
> # predict wage income from age, gender etc.
```

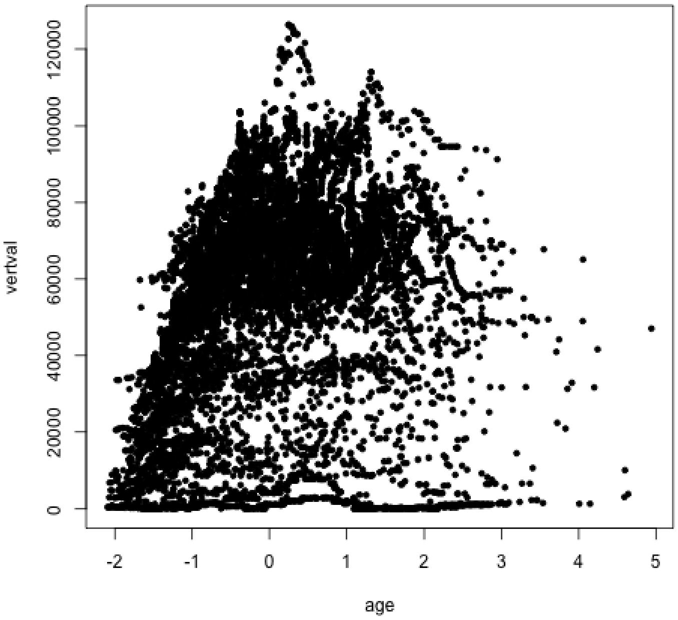

Figure 6.7: Mean Wage Income vs. Age

```
> # prepare nearest-neighbor data
> xdata <- preprocessx(pecs1[,1:5],150)
> kmin(pecs1[,6],xdata,nk=30)$kmin
[1] 5
> zout <- knnest(pecs1[,6],xdata,5)
> nonparvsxplot(zout)
```

we find that the **age** variable, and possibly **wkswrkd**, seem to have a quadratic relation to **wageinc**, as seen in Figures 6.7 and 6.8. So, let's try adding quadratic terms for those two variables. And, to assess how well this works, let's break the data into training and test sets:

```
> pecs2 <- pecs1
> pecs2$age2 <- pecs1$age^2
> pecs2$wks2 <- pecs1$wkswrkd^2
> n <- nrow(pecs1)
> trnidxs <- sample(1:n,12000)
```

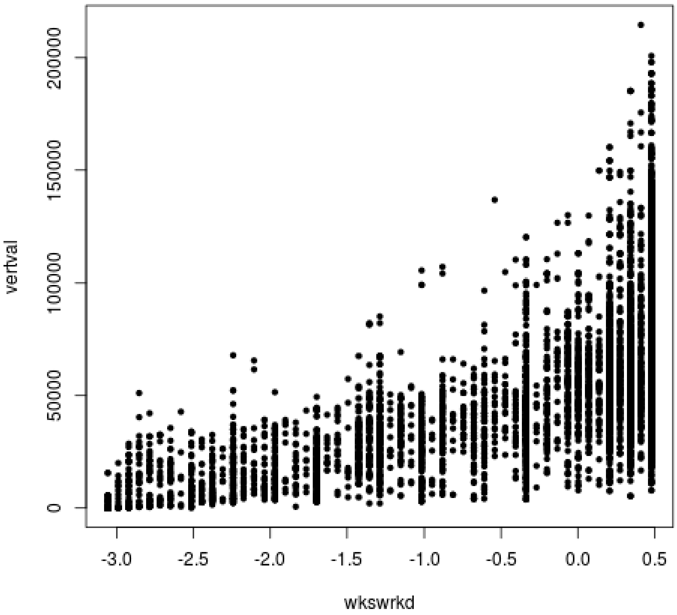

Figure 6.8: Mean Wage Income vs. Weeks Worked

```
> predidxs <- setdiff(1:n,trnidxs)
> lmout1 <- lm(wageinc ~ .,data=pecs1[trnidxs,])
> lmout2 <- lm(wageinc ~ .,data=pecs2[trnidxs,])
> lmpred1 <- predict(lmout1,pecs1[predidxs,])
> lmpred2 <- predict(lmout2,pecs2[predidxs,])
> ypred <- pecs1$wageinc[predidxs]
> mean(abs(ypred-lmpred1))
[1] 25721.5
> mean(abs(ypred-lmpred2))
[1] 25381.08
```

So, adding the quadratic terms helped slightly, about a 1.3% improvement. From a Prediction point of view, this is at best mild, There was also a slight increase in adjusted R-squared, from 0.22 (not shown) to 0.23 (shown below).

But for Description things are much more useful here:

```
> summary(lmout2)
...
Coefficients:
              Estimate Std. Error t value
(Intercept) -63812.415   4471.602 -14.271
age           3795.057    221.615  17.125
sex         -10336.835    841.067 -12.290
wkswrkd        598.969    131.499   4.555
ms           14810.929    928.536  15.951
phd          20557.235   2197.921   9.353
age2           -39.833      2.608 -15.271
wks2             9.874      2.213   4.462
            Pr(>|t|)
(Intercept)  < 2e-16 ***
age          < 2e-16 ***
sex          < 2e-16 ***
wkswrkd     5.29e-06 ***
ms           < 2e-16 ***
phd          < 2e-16 ***
age2         < 2e-16 ***
wks2        8.20e-06 ***
...
Multiple R-squared:  0.2385,    Adjusted R-squared:  0.2381
```

As usual, we should not make too much of the p-values, especially with a sample size this large (16411 for **pecs1**). So, all those asterisks don't tell us too much. But a confidence interval computed from the standard error shows that the absolute age-squared effect is at least about 34, far from 0, and it does make a difference, say on the first person in the sample:

```
> predict(lmout1,pecs1[1,])
       1
62406.48
> predict(lmout2,pecs2[1,])
       1
63471.53
```

The more sophisticated model predicts about an extra $1,000 in wages for this person.

Most important, the negative sign for the age-squared coefficient shows that income tends to level off and even decline with age, something that could be quite interesting in a Description-based analysis.

The positive sign for **wkswrkd** is likely due to the fact that full-time workers tend to have better jobs.

6.12 Classification Settings

Since we treat classification as a special case of regression, we can use the same fit assessment methods, though in some cases some adapting of them is desirable.

6.12.1 Example: Pima Diabetes Study

Let's illustrate with the Pima diabetes data set from Section 4.4.2.

```
> pima <- read.csv('../Data/Pima/pima-indians-diabetes.data')
```

It goes without saying that with any data set, we should first do proper cleaning.[6] This data is actually a very good example. Let's first try the **freqparcoord** package:

```
> library(freqparcoord)
> freqparcoord(pima[,-9],-10)
```

[6] And of course should have done so for the other data earlier in this chapter, but I decided to keep the first analyses simple.

Here we display that 10 data points (predictors only, not response variable) whose estimated joint density is lowest, thus qualifying as "unusual."

The graph is shown in Figure 6.9. Again we see a jumble of lines, but look at the big dips in the variables **BP** and **BMI**, blood pressure and Body Mass Index. They seem unusual. Let's look more closely at blood pressure:

```
> table(pima$BP)

  0  24  30  38  40  44  46  48  50  52  54  55
 35   1   2   1   1   4   2   5  13  11  11   2
 56  58  60  61  62  64  65  66  68  70  72  74
 12  21  37   1  34  43   7  30  45  57  44  52
 75  76  78  80  82  84  85  86  88  90  92  94
  8  39  45  40  30  23   6  21  25  22   8   6
 95  96  98 100 102 104 106 108 110 114 122
  1   4   3   3   1   2   3   2   3   1   1
```

No one has a blood pressure of 0, yet 35 women in our data set are reported as such. The value 24 is suspect too, but the 0s are wrong for sure. What about BMI?

```
> table(pima$BMI)

   0 18.2 18.4 19.1 19.3 19.4 19.5 19.6 19.9   20
  11    3    1    1    1    1    2    3    1    1
20.1 20.4 20.8   21 21.1 21.2 21.7 21.8 21.9 22.1
...
```

Here again, the 0s are clearly wrong. So, at the very least, let's exclude such data points:

```
> pima <- pima[pima$BP > 0 & pima$BMI > 0,]
> dim(pima)
[1] 729   9
```

(We lost 38 cases.)

Now, for our analysis, start with fitting a logit model, then comparing to kNN. First, what value of k should we use?:

```
> glmout <- glm(Diab ~ .,data=pima,family=binomial)
> xdata <- preprocessx(pima[,-9],150,xval=TRUE)
> kminout <- kmin(pima$Diab,xdata,lossftn=predwrong,nk=30)
> kminout$kmin
[1] 65
```

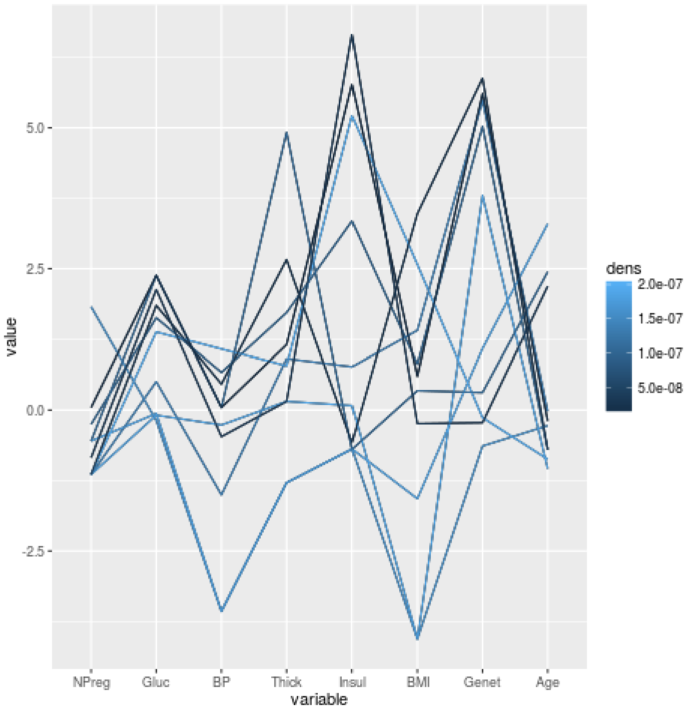

Figure 6.9: Outlier Hunt

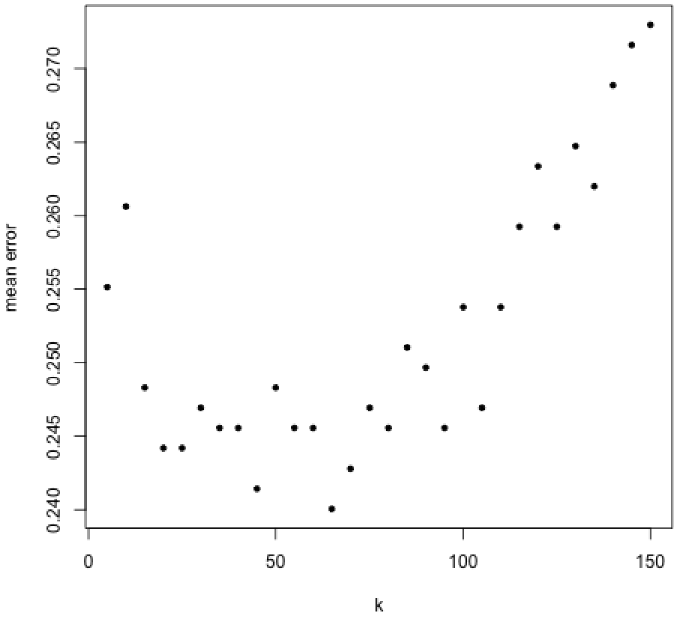

Figure 6.10: Best k for Pima

Note that here we used the **predwrong()** loss function, which computes the misclassification rate.

The "best" value of *k* found was 65, but the plot suggests that smaller values might be tried, as seen in Figure 6.10. Let's go with 50, and compare the parametric and nonparametric fits:

```
> kout <- knnest(pima$Diab,xdata,50)
> parvsnonparplot(glmout,kout)
```

The results of the plot are shown in Figure 6.11. There does appear to be some overestimation by the logit at very high values of the regression function, indeed all the range past 0.5. This can't be explained by the fact, noted before, that kNN tends to underestimate at the high end.

Note carefully that if our goal is Prediction, it may not matter much at the high end. Recall the discussion on classification contexts in Section 6.4.1. If the true population regression value is 0.8 and we estimate it to be 0.88, we still predict $Y = 1$, which is the same as what we would predict if we knew the true population regression function. Similarly, if the true regression value is 0.25 but we estimate it to be 0.28, we still make the proper guess for Y.

For Description, though, we should consider a richer model. Running **nonparvsxplot()** (not shown) suggests adding quadratic terms for the variables **Gluc**, **Insul**, **Genet** and especially **Age**. Adding these, and rerunning **knnest()** with **nearf = loclin** to deal with kNN's high-end bias, our new parametric-vs.-nonparametric plot is shown in Figure 6.12.

The reader may ask if now we have *under*estimation by the parametric model at the high end, but we must take into account the fact that with **nearf = loclin**, we can get nonparametric estimates for the regression function that are smaller than 0 or greater than 1, which is impossible in the classification setting. The perceived "underestimation" actually occurs at values at which the nonparametric figures are larger than 1.

In other words, we now seem to have a pretty good model.

6.13 Special Note on the Description Goal

If we are unable to improve the fit of our parametric model in a setting in which kNN seems to give a substantially better fit, we should be quite wary of placing too much emphasis on the values of the $\widehat{\beta}_j$. As we saw with the currency data, the estimated coefficients can be quite sensitive to unusual observations and so on.

This is not to say the $\widehat{\beta}_j$ are useless in such settings. On the contrary, they may be quite valuable. But they should be used with caution.

6.14 Mathematical Complements

6.14.1 The Hat Matrix

We'll use the notation of Section 2.3.2 here. The *hat matrix* is defined as

$$H = A(A'A)^{-1}A' \tag{6.10}$$

The name stems from the fact that we use H to obtain "Y-hat," the pre-

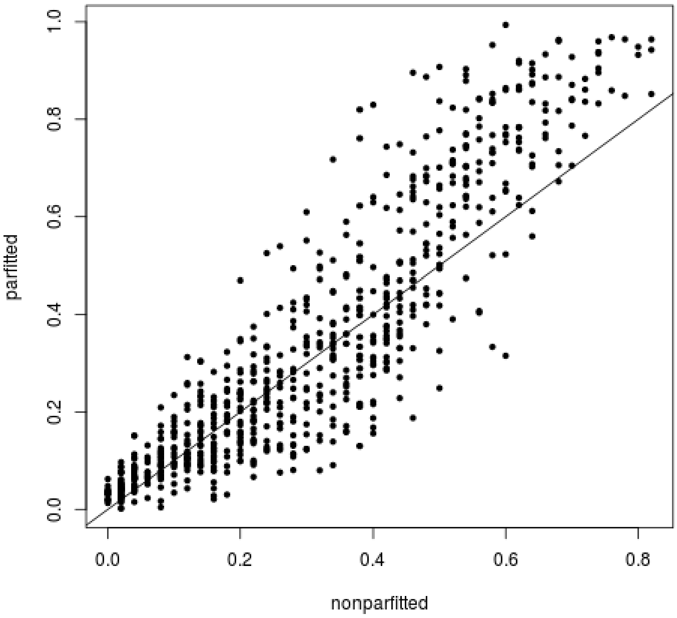

Figure 6.11: Estimation of Regression Values, Two Methods

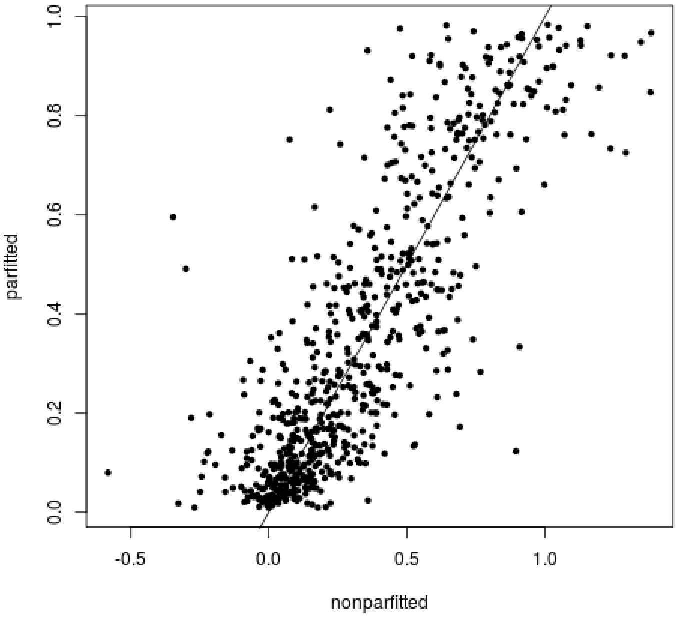

Figure 6.12: Estimation of Regression Values, Two Methods

dicted values for the elements of D:

$$\widehat{D}_i = \widehat{\mu}(\widetilde{X_i}') \ \widehat{\beta} \tag{6.11}$$

so

$$\widehat{D} = A\widehat{\beta} = A(A'A)^{-1}A'D = HD \tag{6.12}$$

Using the famous relation $(VW)' = W'V'$, it is easily verified that H is a symmetric matrix. Also, some easy algebra shows that H is *idempotent*, i.e.

$$H^2 = H \tag{6.13}$$

(The idempotencny also follows from the fact that H is a projection operator; once one projects, projecting the result won't change it.)

This leads us directly to the residuals:

$$L = D - A\widehat{\beta} = D - HD = (I - H)D \tag{6.14}$$

The diagonal elements

$$h_{ii} = H_{ii} \tag{6.15}$$

are known as the *leverage* values, another measure of influence like those in Section 6.8.1, for the following reason. Looking at (6.12), we see that

$$\widehat{D}_i = h_{ii} D_i \tag{6.16}$$

This shows us the effect of true value D_i on the fitted value $\widehat{D}_i$:

$$h_{ii} = \frac{\partial \widehat{D}_i}{\partial D_i} \tag{6.17}$$

So, h_{ii} can be viewed as a measure of how much influence observation i has on its fitted value. A large value might thus raise concern — but how large is "large"?

Let w_i denote row i of H, which is also column i since H is symmetric. Then

$$h_{ii} = H_{ii} = (H^2)_{ii} = w_i' w_i = h_{ii}^2 + \sum_{j=1, j \neq i}^{n} h_{ij}^2 \tag{6.18}$$

This directly tells us that $h_{ii} \geq 0$. But it also tells us that $h_{ii} \geq h_{ii}^2$, which forces $h_{ii} \leq 1$.

In other words,

$$0 \leq h_{ii} \leq 1 \tag{6.19}$$

This will help assess whether a particular h_{ii} value is "large."

As mentioned, there is a very large literature of this kind of analysis. The reader is referred to the many books on this topic, such as Atkinson and Riani (2000).

6.14.2 Martrix Inverse Update

The famous Sherman-Morrison-Woodbury formula says that for an invertible matrix B and vectors u and v

$$(B + uv')^{-1} = B^{-1} - \frac{1}{1 + v' B^{-1} u} B^{-1} uv' B^{-1} \tag{6.20}$$

In other words, if we have already gone to the trouble of computing a matrix inverse, and the matrix is then updated as above by adding uv', then we do not have to compute the new inverse from scratch; we need only modify the old inverse, as specified above.

Let's apply that to Section 6.8.1, where we discussed the effect of deleting an observation from our data set. Setting $B = A'A$, $u = \widetilde{X}_i$ and $v = -\widetilde{X}_i$, then the new version of $A'A$ after deleting observation i is

$$(A'A)_{(-i)} = A'A + uv' \tag{6.21}$$

That will be our value for the left-hand side of (6.20). Look what happens to the right-hand side:

$$1 + v'B^{-1}u = 1 - \widetilde{X}_i'(A'A)^{-1}\widetilde{X}_i \tag{6.22}$$

But that subtracted term is just h_{ii}! Now that's convenient.

Now, again in (6.20), note that

$$B^{-1}u = (A'A)^{-1}\widetilde{X}_i \tag{6.23}$$

and

$$v'B^{-1} = -\widetilde{X}_i'(A'A)^{-1} \tag{6.24}$$

After substuting all this in (6.20) to computer the new $(A'A)^{-1}$, we can find our new $\widehat{\beta}$ by post-multiplying by $A'_{(-i)}D_{-i}$, where the latter factors are the new versions of A' and D.

It should be noted, however, that this approach has poor roundoff error properties if $A'A$ is *ill-conditioned*, meaning that it is nearly singular. This in turn can arise if some of the predictor variables are highly correlated with each other, as we will see in Chapter 8.

6.14.3 The Median Minimizes Mean Absolute Deviation

Let's derive (6.8).

First, suppose a random variable W has a density f_W. What value m minimizes $E[(W - m)]$?

$$\begin{aligned}
E[(W-m)] &= \int_{-\infty}^{\infty} |t-m| \; f_V(t)dt && (6.25)\\
&= \int_{-\infty}^{m} (m-t) \; f_V(t)dt + \int_{m}^{\infty} (t-m) \; f_V(t)dt && (6.26)\\
&= mP(W<m) - \int_{-\infty}^{m} t \; f_V(t)dt + \int_{m}^{\infty} t \; f_V(t)dt - mP(W>m)
\end{aligned}$$

Differentiating with respect to m, we have

$$0 = P(W < m) + mf_V(m) - mf_V(m) - mf_V(m) - [P(W > m) + m(-f_V(m))] \tag{6.27}$$

In other words,

$$P(W < m) = P(W > m) \tag{6.28}$$

i.e. $m = \text{median}(W)$.

The extension to conditional median then follows the same argument as in Section 1.13.1.

6.15 Further Exploration: Data, Code and Math Problems

Exercises

1. The contributors of the Forest Fire data set to the UCI Machine Learning Repository, *https://archive.ics.uci.edu/ml/datasets/Forest+Fires*, describe it as "a difficult regression problem." Apply the methods of this chapter to attempt to tame this data set.

2. Using the methods of this chapter, re-evaluate the two competing Poisson-based analyses in Section 4.5.

3. Suppose the predictor vector X in a two-class classification problem has a joint density f_X. Define the "population" residual $r = Y - g(X)$, and its cumulative distribution function F_r.

(a) Express F_r as an integral involving f_X and the regression function $\mu(t)$.

(b) Define a sample analog of (a), and use it to develop a graphical fit assessment tool for parametric models such as the logistic, using R's empirical distribution function **ecdf()** and kNN.

4. Write an R function analogous to **influence()** for **quantreg** objects.

Chapter 7

Measuring Factor Effects

Throughout this book, we've noted the twin goals of regression and classification analysis, Prediction and Description. In many cases, our methodological approach has been aimed at Prediction. In this chapter, though, the theme will entirely be Description, with most (but not all) of the material being concerned with parametric models.

We will see, though that attaining this goal may require some subtle analysis. It will be especially important to bear in mind the following principle:

> The regression coefficient (whether sample estimate or population value) for one predictor variable may depend on which other predictors are present.

7.1 Example: Baseball Player Data

In Section 1.7.1.2, we found that the data indicated that older baseball players — of the same height — tend to be heavier, with the difference being about 1 pound gain per year of age. This finding may surprise some, since athletes presumably go to great lengths to keep fit. Ah, so athletes are similar to ordinary people after all.

We may then ask whether a baseball player's weight is also related to the position he plays. So, let's now bring the Position variable in our data into play. First, what is recorded for that variable?

```
> levels(mlb$Position)
```

```
[1] "Catcher"           "First_Baseman"
[3] "Outfielder"        "Relief_Pitcher"
[5] "Second_Baseman"    "Shortstop"
[7] "Starting_Pitcher"  "Third_Baseman"
```

o, all the outfield positions have been simply labeled "Outfielder," though pitchers have been separated into starters and relievers.

Technically, this variable, **mlb$Position**, is an R **factor**. This is a fancy name for an integer vector with labels, such that the labels are normally displayed rather than the codes. So actually catchers are coded 1, designated hitters 2, first basemen 3 and so on, but in displaying the data frame, the labels are shown rather than the codes.

The designated hitters are rather problematic, as they only exist in the American League, not the National League. Let's restrict our analysis to the other players:

```
> nondh <-
   mlb[mlb$Position != "Designated_Hitter",]
> nrow(mlb)
[1] 1034
> nrow(nondh)
[1] 1016
```

We've deleted the designated hitters, assigning the result to **nondh**. A comparison of numbers of rows show that there were only 18 designated hitters in the data set anyway.

In order to have a proper basis of comparison below, we should re-run the weight-height-age analysis:

```
> summary(lm(Weight ~ Height + Age,data=nondh))
...
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -187.6382    17.9447  -10.46  < 2e-16
Height         4.9236     0.2344   21.00  < 2e-16
Age            0.9115     0.1257    7.25 8.25e-13

(Intercept) ***
Height      ***
Age         ***
...
Multiple R-squared:  0.318,     Adjusted R-squared:  0.3166
```

Basically, no change from before. Now, for simplicity, let's consolidate into four kinds of positions: infielders, outfielders, catchers and pitchers. That means we'll need three dummy variables:

```
> poscodes <- as.integer(nondh$Position)
> infld <- as.integer(poscodes==3 | poscodes==6 |
   poscodes==7 | poscodes==9)
> outfld <- as.integer(poscodes==4)
> pitcher <- as.integer(poscodes==5 | poscodes==8)
```

Again, remember that catchers are designated via the other three dummies being 0.

So, let's run the regression:

```
> lmpos <- lm(Weight ~ Height + Age + infld + outfld +
   pitcher,data=nondh)
> summary(lmpos)
...
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -182.7216    18.3241  -9.972  < 2e-16
Height         4.9858     0.2405  20.729  < 2e-16
Age            0.8628     0.1241   6.952 6.45e-12
infld         -9.2075     1.6836  -5.469 5.71e-08
outfld        -9.2061     1.7856  -5.156 3.04e-07
pitcher      -10.0600     2.2522  -4.467 8.84e-06

(Intercept) ***
Height      ***
Age         ***
infld       ***
outfld      ***
pitcher     ***
...
Mult. R-squared:  0.3404, Adj. R-squared:  0.3372
...
```

The estimated coefficients for the position variables are all negative. For example, for a given height and age, pitchers are on average about 10.1 pounds lighter than catchers of the same height, while outfielders the figure is about 9.2 pounds. An approximate 95% confidence interval for the population value of the latter is

```
9.2 \pm 2 \times 1.8 = (5.6,12.8)
```

So, the image of the "beefy" catcher is borne out.

Note that the estimated coefficient for age shrank a little. In our original analysis, with just height and age as predictors, it had been 0.9115,[1], but now is only 0.8628. The associated confidence interval, (0.61,1.11), is still we away from 0, indicating weight increase with age, but the effect is now smaller than before. This is an example of the phenomenon mentioned at the outset of this chapter that the coefficient for one predictor may depend on what other predictors are present.

It also suggests that the age effect on weight is not uniform across playing positions. To investigate this, let's add interaction terms:

```
> summary(lm(Weight ~ Height + Age +
   infld + outfld + pitcher +
   Age*infld + Age*outfld + Age*pitcher ,data=nondh))
...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -168.5453    20.3732  -8.273 4.11e-16
Height         4.9854     0.2407  20.714  < 2e-16
Age            0.3837     0.3335   1.151   0.2501
infld        -22.8916    11.2429  -2.036   0.0420
outfld       -27.9894    11.9201  -2.348   0.0191
pitcher      -31.9341    15.4175  -2.071   0.0386
Age:infld      0.4629     0.3792   1.221   0.2225
Age:outfld     0.6416     0.4033   1.591   0.1120
Age:pitcher    0.7467     0.5232   1.427   0.1539

(Intercept) ***
Height      ***
Age
infld       *
outfld      *
pitcher     *
Age:infld
Age:outfld
Age:pitcher
...
Mult. R-squared:  0.3424,    Adj. R-squared:  0.3372
```

This doesn't look helpful. Confidence intervals for the estimated interaction coefficients include 0 but are wide. Thus there could be important

[1] This was the case even after removing the Designated Hitters, not shown here.

interaction effects, or they could be tiny; we just don't have a large enough sample to say much.

Note that the coefficients for the position dummies changed quite a bit, but this doesn't mean we now think there is a larger discrepancy between weights of catchers and the other players. For instance, for 30-year-old players, the estimated difference in mean weight between infielders and catchers of a given height is

```
−22.8916 + 30 \times 0.4629 = −9.0046
```

similar to the -9.2075 figure we had before.

7.2 Simpson's Paradox

The famous *Simpson's Paradox* should not be considered a paradox, when viewed in the light of a central point we have been discussing in this chapter, which we will state a little differently here:

> The regression coefficient (sample or population) for a predictor variable may change substantially when another predictor is added. In particular, its sign may change, from positive to negative or *vice versa*.

7.2.1 Example: UCB Admissions Data (Logit)

The most often-cited example, in a tabular context, is that of the UC Berkeley admissions data (Bickel, 1975). The issue at hand was whether the university had been discriminating against women applicants for admission to graduate school.

On the surface, things looked bad for the school — 44.5% of the male applicants had been admitted, compared to only 30.4% of the women. However, upon closer inspection it was found that the seemingly-low female rate was due to the fact that the women tended to apply to more selective academic departments, compared to the men. After correcting for the Department variable, it was found that rather than being victims of discrimination, the women actually were slightly favored over men. There were six departments in all, labeled A-F.

The data set is actually included in base R. As mentioned, it is stored in the form of an R table:

```
> ucb <- UCBAdmissions
> class(ucb)
[1] "table"
> ucb
, , Dept = A

          Gender
Admit      Male Female
  Admitted  512     89
  Rejected  313     19

, , Dept = B

          Gender
Admit      Male Female
  Admitted  353     17
  Rejected  207      8
...
```

In R, it is sometimes useful to convert a table to an artificial data frame, which in this case would have as many rows as there were applicants in the UCB study, 4526. The **regtools** function **tbltofakedf()** facilitates this:

```
> ucbdf <- tbltofakedf(ucb)
> dim(ucbdf)
[1] 4526    3
> head(ucbdf)
     [,1]       [,2]   [,3]
[1,] "Admitted" "Male" "A"
[2,] "Admitted" "Male" "A"
[3,] "Admitted" "Male" "A"
[4,] "Admitted" "Male" "A"
[5,] "Admitted" "Male" "A"
[6,] "Admitted" "Male" "A"
```

The first six rows are the same, and in fact there will be 512 such rows, since, as seen above, there were 512 male applicants who were admitted to Department A.

Let's analyze this data using logistic regression. With such coarsely discrete data, this is not a typical approach, but it will illustrate the dynamics of Simpson's Paradox.

First, convert to usable form, not R factors:

```
> ucbdf$admit <- as.integer(ucbdf[,1] == 'Admitted')
> ucbdf$male <- as.integer(ucbdf[,2] == 'Male')
# save work by using the 'dummies' package
> library(dummies)
> dept <- ucbdf[,3]
> deptdummies <- dummy(dept)
> head(deptdummies)
     deptA deptB deptC deptD deptE deptF
[1,]     1     0     0     0     0     0
[2,]     1     0     0     0     0     0
[3,]     1     0     0     0     0     0
[4,]     1     0     0     0     0     0
[5,]     1     0     0     0     0     0
[6,]     1     0     0     0     0     0
> ucbdf1 <- cbind(ucbdf,deptdummies[,-6])[,-(1:3)] # only 5 dummies
> head(ucbdf1)
        V1   V2 V3 admit male deptA deptB deptC
1 Admitted Male  A     1    1     1     0     0
2 Admitted Male  A     1    1     1     0     0
3 Admitted Male  A     1    1     1     0     0
4 Admitted Male  A     1    1     1     0     0
5 Admitted Male  A     1    1     1     0     0
6 Admitted Male  A     1    1     1     0     0
  deptD deptE
1     0     0
2     0     0
3     0     0
4     0     0
5     0     0
6     0     0
```

Now run the logit, first only with the **male** predictor, then adding the departments:

```
> glm(admit ~ male,data=ucbdf1,family=binomial)
...
Coefficients:
(Intercept)         male
    -0.8305       0.6104
...
> glm(admit ~ .,data=ucbdf1,family=binomial)
...
Coefficients:
```

```
(Intercept)        male       deptA
   -2.62456    -0.09987     3.30648
      deptB       deptC       deptD
    3.26308     2.04388     2.01187
      deptE
    1.56717
...
```

So the sign for the **male** variable switched from positive (men are favored) to slightly negative (women have the advantage). Needless to say this analysis (again, in table form, not logit) caused quite a stir, as the evidence against the university had looked so strong.

By the way, note that the coefficients for all five dummies were positive, which reflects the fact that all the departments A-E had higher admissions rates than department F:

```
> apply(ucb,c(1,3),sum)
          Dept
Admit        A   B   C   D   E   F
  Admitted 601 370 322 269 147  46
  Rejected 332 215 596 523 437 668
```

7.2.2 A Geometric Look

To see the problem geometrically, here is a variation of another oft-cited example: We have scalar variables Y, X and I, with:

- $I = 0$ or 2, w.p. 1/2 each
- $X\ N(10 - 2I, 0.5)$
- $Y = X + 3I + \epsilon$, with $\epsilon\ N(0, 0.5)$

So, the population regression function with two predictors is

$$E(Y \mid X = t, I = k) = t + 3k \tag{7.1}$$

With just X as a predictor, we defer this to the exercises at the end of this chapter, but let's simulate it all:

```
> n <- 1000
> i <- sample(c(0,2),n,replace=TRUE)
> x <- rnorm(n,mean=0,sd=0.5)
> x <- x + 10 - 2*i
> eps <- rnorm(n,sd=0.5)
> y <- x + 3*i + eps
> plot(x,y)
> abline(coef(lm(y ~ x)))
> idxs0 <- which(i == 0)
> abline(coef(lm(y[idxs0] ~ x[idxs0])))
> idxs1 <- setdiff(1:n,idxs0)
> abline(coef(lm(y[idxs1] ~ x[idxs1])))
> text(9.0,13.8,'reg. line, I = 1')
> text(7.0,8.5,'reg. line, I = 0')
> text(8.2,11.4,'reg. line, no I')
```

The result, shown in Figure 7.1, dramatizes the problem. If we I as a predictor, we positive slopes for the regression function (slope*s* plural, as there are two possible values of I. But if do not know/use I, the slope is negative.

7.2.3 The Verdict

Simpson's is not really a paradox — let's just call it Simpson's Phenomenon — but is crucially important to keep in mind in applications in which Description is the goal. And the solution to the "paradox" is to think twice before deleting any predictor variables.

Ironically, this last point is somewhat at odds with the theme of Chapter 9, in which we try to pare down the number of predictors. When we have correlated variables, such as Gender and Department in the admissions data, it might be tempting to delete one or more of them on the grounds of "redundancy," but we first should check the effects of deletion, e.g. sign change.[2]

On the other hand, this is rather consistent with the method of *ridge regression* in Chapter 8. That approach attempts to ameliorate the effects of correlated predictor variables , rather than resorting to deleting some of them.

Once again, we see that regression and classification methodology does not

[2] In the admissions data, the correlation, though substantial, would probably jot warrant deletion in the first place, but the example does illustrate the dangers.

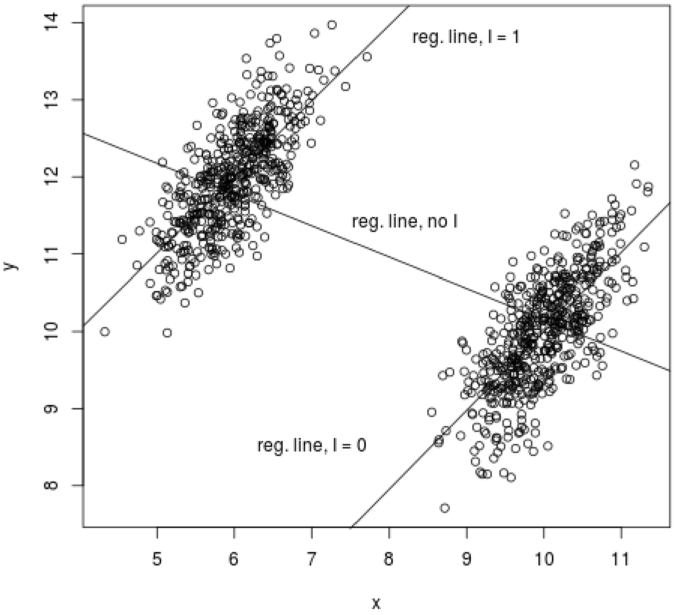

Figure 7.1: Geometric View of Simpson's Paradox

always offer easy, pat solutions.

7.3 Comparing Groups in the Presence of Covariates

Recall the my old consulting problem from Section 1.9.1:

> 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.

How do we deal with such situations?

7.3.1 ANCOVA

There is a classical statistical method named Analysis of Covariance, used to compare several groups in terms of a variable Y, in the presence of covariates. It assumes:

(a) Conditional mean response is a linear function of the covariates.

(b) Coefficients of the predictors are the same across groups, except for the constant term β_0.

(c) Conditional variance is constant, both within and between groups.

The difference between groups is then taken to be the difference between constant terms.

But wait a minute, you ask, isn't this just what we've been doing with dummy variables? The answer of course is yes. See the next section.

7.3.2 Example: Programmer/Engineer 2000 Census Data

Consider again the census data on programmer and engineer salaries, particularly the analysis in Section **??**. Considering the dummy variables Gender, MS and PhD, we have six subgroups that could be compared.[3] Our use here of **lm()**, plus the fact that we don't have any interaction terms involving the dummies, implies assumptions (a)-(c) in the last section.

The constant term for any group follows from such considerations. For example, the one for the group consisting of female PhDs, the constant term is

$$63812.415 - 10336.835 + 20557.235 = -663592 \tag{7.2}$$

We could then compare the six groups in this manner.

7.3.3 Answering Other Subgroup Questions

But we could go further. The coefficients of the dummy variables are obviously quite useful, but they are just *marginal* quantities, i.e. they measure the effect on mean response of a *one-unit* increase in a predictor, such as a one-year increase in age. It may also be useful to study *overall* effects within groups.

In the programmer/engineer data, the estimated coefficient for PhD was \$20557.235. But the distribution of the various predictors of those with doctorates will likely be substantially different from the data as a whole. The doctorate holders are more likely to be older, more likely to be male and so on. (Indeed, they may also be more likely to be unemployed.)

So, how can we assess how the PhDs as a group are doing? For example, in purely financial terms, how much did having a PhD impact their wage income?

We can answer such questions by appealing to the Law of Iterated Expectations,

$$EV = E[E(V \mid U)]M \tag{7.3}$$

[3] The nature of the data definition is such that the MS and PhD variables can't simultaneously be 1. Note that ordinarily we would be interested in the dummies individually, rather than defining the subgroups, but for the sake of illustration let's suppose the groups are of interest.

First, let's establish a basis for comparison:

```
> library(regtools)
> data(prgeng)
> pe <- prgeng
> pe$ms <- as.integer(pe$educ == 14)
> pe$phd <- as.integer(pe$educ == 16)
> pecs <- pe[pe$occ >= 100 & pe$occ <= 109,]
> pecs1 <- pecs[,c(1,7,9,12,13,8)]
> pecs1doc <- pecs1[pecs1$phd,]
> mean(pecs1doc$wageinc)
[1] 75000
```

So we estimate the mean wage income of all PhDs to be $75,000. But what if they didn't have PhDs? What if they had stopped their education at a Master's degree?

We can answer that question using (7.3). We take our subgroup data, **pecs1doc**, change their status from PhD to MS, then average our estimated regression function over this subgroup:

```
> pecs1doc$ms <- 1
> pecs1doc$phd <- 0
> lmout <- lm(wageinc ~ .,data=pecs1)
> prout <- predict(lmout,pecs1doc[,-7])
> mean(prout)
[1] 77395.88
```

Oh, this is a little disturbing. If these PhDs had not pursued a doctorate, they actually would have been making about $2400 *more*, rather than the $20,000 *less* that the coefficient of the PhD dummy variable had suggested.

Of course, this then indicates a need for further analysis. For example, was the discrepancy due to the PhDs working in lower-paying sectors of the economy, such as education or civil service? And we should double-check the accuracy of our linear model, and so on. But it is clear that this approach can be used for lots of other interesting investigations.

7.4 Unobserved Predictor Variabless

In Statistical Heaven, we would have data on all the variables having substantial relations to the response variable. Reality is sadly different, and often we feel that our analyses are hampered for lack of data on crucial variables.

Statisticians have actually developing methodology to deal with this problem. Not surprisingly, the methods have stringent assumptions, and they are hard to verify. But they should be part of the toolkit of any data scientist, either to use where appropriate or at least understand when presented with such analyses done by others. The following sections will provide brief introductions to such methods.

7.4.1 Instrumental Variables (IV)

This one is quite controversial. It's primarily used by economists, but has become increasingly popular in the social and life sciences.

Suppose we are interested in the relation between Y and two predictors, $X^{(1)}$ and $X^{(2)}$. We observe Y and $X^{(1)}$ but not $X^{(2)}$. We believe that the two population regression functions (one predictor vs. two) are well approximated by the linear model:[4]

$$E(Y \mid X^{(1)}) = \beta_{01} + \beta_{11} X^{(1)} \tag{7.4}$$

$$E(Y \mid X^{(1)}, X^{(2)}) = \beta_{02} + \beta_{12} X^{(1)} + \beta_{22} X^{(2)} \tag{7.5}$$

We are primarily interested in the role of $X^{(1)}$, i.e. the value of β_{12}. However, as has been emphasized so often in this chapter, generally

$$\beta_{11} \neq \beta_{12} \tag{7.6}$$

A commonly offered example concerns a famous economic study regarding the returns to education. Here Y is weekly wage and $X^{(1)}$ is the number of years of schooling. The concern was that this analysis doesn't account for "ability"; highly-able people might pursue many years of education, and thus get a good wage due to their ability, rather than the education itself. If a measure of ability were included in our data, we could simply use it as a covariate and fit the model (7.5), but no such measure was included in the data.[5]

[4] The second model does not imply the first. What if $X(2) = X^{(1)\ 2}$, for instance?

[5] Of course, even with better data, "ability" would be hard to define. Does it mean IQ (of which I am very skeptical), personal drive or what?

The *instrumental variable* approach involves using a variable that is intended to remove from $X^{(1)}$ the portion of that variable that involves ability. If this works — a big "if" — then we will be able to measure the effect of years of schooling without the confounding effect of ability. The instrumental variable, or simply the *instrument*, is observable.

7.4.1.1 The IV Method

Let Z denote our instrument. Its definition consists of two conditions (let ρ denote population correlation):

(a) $\rho(Z, X^{(1)}) \neq 0$

(b) $\rho(Z, X^{(2)}) = 0$

In other words, the instrument is uncorrelated with the unseen predictor variable, but *is* correlated with the response variable.[6]

In the years-of-schooling example, the instrument is distance from a college. The rationale here is that, if there are no nearby postsecondary institutions, the person will find it difficult to pursue a college education, and may well decide to forego it.[7] That gives us condition (a), and it seems reasonable that this variable should be unrelated to ability.[8] In this manner, we hope to capture that part of $X^{(1)}$ that is unrelated to ability.

Where do these conditions (a) and (b) come from mathematically? Let's first do a (population) calculation:

$$Cov(Z,Y) = \beta_{12} Cov(Z, X^{(1)} + \beta_{22} Cov(Z, X^{(2)}) = \beta_{12} Cov(Z, X^{(1)}) \quad (7.7)$$

and thus

$$\begin{aligned} \beta_{12} &= \frac{Cov(Z,Y)}{Cov(Z,X^{(1)})} & (7.8) \\ &= \frac{\rho(Z,Y)\ \sigma(Y)}{\rho(Z,X^{(1)})\ \sigma(X^{(1)})} & (7.9) \end{aligned}$$

[6] Our focus here is on linear models. In nonlinear cases, we must ask for independence rather than merely lack of correlation.

[7] The study was based on data from 1980, when there were fewer colleges in the U.S. than there are now.

[8] This could be debated, of course. See Section 7.4.1.5.

where for any random variable W, $\sigma(W)$ denotes its population standard deviation.

Since we can estimate $Cov(Z,Y)$ and $Cov(Z,X^{(1)})$ from our sample, we can thus estimate the parameter of interest, β_{12} — in spite of not observing $X^{(2)}$.

This is wonderful! Well, wait a minute...is it too good to be true? Well, as noted, the assumptions are crucial, such as:

- We assume the linear models (7.4) and (7.5). The first can be assessed from our data, but the second cannot.

- We assume condition (b) above, i.e. that our instrument is uncorrelated with our unseen variable. Often we are comfortable with that assumption — e.g. that distance from a college is not related to ability — but again, it cannot be verified.

- We need the instrument to have a fairly substantial correlation to the observed predictor, i.e. $\rho(Z,X^{(1)})$ should be substantial. If it isn't, then we have a small or even tiny denominator in (7.9), so that the sample variance of the quotient — and thus of our $\widehat{\beta}_{12}$ will be large, certainly not welcome news.

7.4.1.2 2 Stage Least Squares:

Another way to look at the IV idea is *2 Stage Least Squares* (2SLS), as follows. Recall the phrasing used above, that the instrument

> is a variable that is intended to remove from $X^{(1)}$ the portion of that variable that involves ability.

That suggests that regressing $X^{;(1)}$ on Z.[9]

Let's see what happens. Using (7.5), write

$$E(Y \mid Z) = \beta_0 + \beta_{11}E(X^{(1)}|Z) + \beta_{12}E(X^{(2)}|Z) \tag{7.10}$$

[9] The term "regress V on U means to model the regression function of V given U.

By assumption, Z and $X^{(2)}$ are uncorrelated. If they are also bivariate normally distributed, then they are independent. Assuming this, we have

$$E(X^{(2)}|Z) = E[X^{(2)}] \tag{7.11}$$

In other words,

$$E(Y \mid Z) = c + \beta_{11} E(X^{(1)}|Z) \tag{7.12}$$

for a constant $c = \beta_0 + E[X^{(2)}]$.

But $E(X^{(1)}|Z)$ is the regression of $X^{(1)}$ on Z. In other words, the process is as follows:

> **2 Stage Least Squares:**
>
> - First regress $X^{(1)}$ on the instrument Z, saving the fitted values.
> - Then regress Y on those fitted values.
> - The resulting estimated slope will be $\widehat{\beta}_{11}$, the estimate we are seeking.

In other words, the IV method can be viewed as an application of 2SLS, with the predictor variable in the first stage being our instrument.

In terms of R, this would mean

```
lmout <- lm(x1 ~ z)
estreg <- predict(lmout)
b11hat <- coef(lm(y ~ estreg))[2]
```

However, there are more sophisticated R packages for this, which do more than just find that point estimae $\widehat{\beta}_{11}$, as we will see in the next section.

7.4.1.3 Example: Price Elasticity of Demand

One of the popular R functions for IV computation is **ivreg()**, in the **AER** package. That package's prime example deals with the effects of price on cigarette consumption. (Note: Our analysis here will be a simplified version of the example in the package.)

In the example, they first compute relative price, **rprice**, to adjust for inflation. They also compute an instrument, **tdiff**, as follows.

The problem is that price and demand are interrelated; if demand increases, the price will likely go up too. So, the unseen variable here is that part of demand that comes from mutual interaction of price and demand. But part of price is also determined by tax on the product, which the authors use as their instrument:

```
> data("CigarettesSW", package = "AER")
> cgd <- CigarettesSW
> cgd$rprice <- with(cgd, price/cpi)
> cgd$tdiff <- with(cgd, (taxs - tax)/cpi)
```

Now, we should check that this instrument will be useful. As noted earlier, it should have substantial correlation with its partner variable. Let's check:

```
> cor(cgd$rprice,cgd$tdiff)
[1] 0.7035012
```

Good, so let's run the IV model:

```
> ivout <- ivreg(packs ~ rprice | tdiff, data=cgd)
> summary(ivout)
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 219.5764    16.9894  12.924  < 2e-16
rprice       -1.0195     0.1559  -6.539  3.2e-09

(Intercept) ***
rprice      ***
...
Multiple R-Squared: 0.4905,     Adjusted R-squared: 0.4851
...
```

Not surprisingly (and gratifyingly to government policymakers who wish to reduce cigarette consumption), there does appear to be a substantial negative affect of a price increase on demand – *apart* from the effect that demand itself has on price.

Note the syntax of **ivreg()**. In the formula,

```
packs ~ rprice | tdiff
```

the response and predictor variable go before the vertical line, and the instrument is listed after it.

Just to check that we're using **ivreg()** correctly, let's use 2SLS:

```
> summary(lm(packs ~ predict(lm(rprice~tdiff)),data=cgd))
...
Coefficients:
                            Estimate Std. Error
(Intercept)                 219.5764    20.8631
predict(lm(rprice ~ tdiff))  -1.0195     0.1915
                            t value Pr(>|t|)
(Intercept)                  10.525  < 2e-16 ***
predict(lm(rprice ~ tdiff))  -5.325 6.88e-07 ***
...
Multiple R-squared:  0.2317,     Adjusted R-squared:  0.2235
...
```

Yes, it matches. Note, though, that using 2SLS "manually" like this is not desirable, because the standard errors and so on that are emitted by the second **lm()** call won't be correct; **ivreg()** does use 2SLS internally, but it make the proper corrections. For example, the standard error for price reported by 2SLS above is 0.1915, but actually should be 0.1559, according to **ivreg()**.

One more thing: Why not simply use the instrument directly as a predictor? We certainly could do that:

```
...
Coefficients:
(Intercept)       rprice        tdiff
   223.9610      -1.0676       0.2004
```

Indeed, in this instance, the estimated coefficients for the price variable were almost identical. But in general, this will not be the case, and in settings in which the quantity β_{12} is of central interest, the IV method can be useful.

7.4.1.4 Multiple Predictors

What if we have several observable predictors?[10] The same analysis shows that we need one instrument for each one. How does the estimation then

[10] We are still assuming just one unobserved predictor, which might be viewed as several of them combined.

work?

Suppose our population model is

$$E(Y \mid X^{(1)}, ..., X^{(k+1)}) = \beta_{0,k+1} + \beta_{1,k+1} X^{(1)} + ... + \beta_{k+1,k+1} X^{(k+1)} \quad (7.13)$$

with $X^{(1)}, ..., X^k$ observed but X^{k+1} unobserved. The extension of (7.7) will then be

$$Cov(Y, Z_i) = \beta_{1,k+1} Cov(X^{(1)}, Z_i) + ... + \beta_{k,k+1} Cov(X^{(k)}, Z_i) \quad (7.14)$$

This sets up k linear equations in k unknowns, which can be solved for the estimated $\beta_{j,k+1}$. This then is a Method of Moments estimator, so we can also obtain standard errors.

In many cases, it is felt that a predictor is unrelated to the unobserved variable, and that predictor will serve as its own instrument. In such a situation, the fitted values reduce to the values of the predictor itself.

7.4.1.5 The Verdict

In our years-of-schooling example (Section 7.4.1.1), it was mentioned that the assumption that the distance variable was unreleated to ability was debatale. For example, we might reason that able children come from able parents, and able parents believe college is important enough that they should live near one. This is an example of why the IV approach is so controversial.

Nevertheless, the possible effect of unseen variables itself can make an analysis controversial. IVs may be used in an attempt to address such problems. However, extra care is warranted if this method is used.

7.4.2 Random Effects Models

Continuing our theme here in Section 7.4 of approaches to account for unseen variables, we now turn briefly to *mixed effects models.* Consider a usual linear regression model for one predictor X,

$$E(Y \mid X = t) = \beta_0 + \beta_1 t \quad (7.15)$$

for unknown constants β_0 and β_1 that we estimated from the data.

Now alter the model so that β_0 is random, each unit (e.g. each person) having a different value, though all having a common value of β_1. We might observe people over time, measuring something that we model as having a linear time trend; the slope of that trend is assumed the same for all people, but the starting point β_0 is not.

We might write our new model as

$$E(Y \mid X = t) = \beta_0 + B + \beta_1 t \tag{7.16}$$

where α is a random variable having mean 0. Each person has a different value of B, with the slopes for people now being a random variable with mean β_0 and variance σ_a^2.

It is more common to write

$$Y = \beta_0 + \alpha + \beta_1 X + \epsilon \tag{7.17}$$

where $E\epsilon$ has mean 0 and variance σ_e^2. The population values to be estimated from our data are β_0, β_1, σ_a^2 and σ_e^2. Typically these are estimated via Maximum Likelihood (with the assumptions that α and ϵ have normal distributions, etc.), though the Method of Moments is possible too.

The variables α and ϵ are called *random effects* (they are also called *variance components*), while the $\beta_0 + \alpha + \beta_1 X$ portion of the model is called *fixed effects*. This phrasing is taken from the term *fixed-X regression*, which we saw in Section 2.2; actually, we could view this as a random-X setting, but the point is that even then we are not estimating the distribution of X. Due to the presence of both fixed and random effects, the term *mixed-effects model* is used.

7.4.2.1 Example: Movie Ratings Data

The famous Movie Lens data (`http://files.grouplens.org/datasets/movielens/` provide ratings of many movies by many users. We'll use the 100,000-rating data here, which includes some demographic variables for the users. The R package **lme4** will be our estimation vehicle.

First we need to merge the ratings and demographic data:

```
> ratings <- read.table('u.data')
> names(ratings) <- c('usernum','movienum','rating','transID')
> demog <- read.table('u.user',sep='|')
> names(demog) <- c('usernum','age','gender','occ','ZIP')
> u.big <- merge(ratings,demog,by.x=1,by.y=1)
> u <- u.big[,c(1,3,5,6)]
```

We might speculate that older users are more lenient in their ratings. Let's take a look:

```
> z <- lmer(rating ~ age+gender+(1|usernum),data=u)
> summary(z)
...
Random effects:
 Groups   Name        Variance Std.Dev.
 usernum  (Intercept) 0.175    0.4183
 Residual             1.073    1.0357
Number of obs: 100000, groups:  usernum, 943

Fixed effects:
             Estimate Std. Error t value
(Intercept)  3.469074   0.048085   72.14
age          0.003525   0.001184    2.98
genderM     -0.002484   0.031795   -0.08

Correlation of Fixed Effects:
        (Intr) age
age     -0.829
genderM -0.461 -0.014
```

First, a word on syntax. Here our regression formula was

```
rating ~ age + gender + (1|usernum)
```

Most of this looks the same as what we are accustomed to in **lm()**, but the last term indicates the random effect. In R formulas, '1' is used to denote a constant term in a regression equation (we write '-1' in our formula if we want no such term), and here '(1—usernum)' specifies a random effects constant term which depends on **usernum** but *is unobserved.*

So, what is the answer to our speculation about age? Blind use of significance testing would mean announcing "Yes, there is a significant positive relation between age and ratings." But the effect is tiny; a 10-year difference in age would mean an average increase of only 0.03525, on a ratings scale of 1 to 5. There doesn't seem to be much different between men and women either.

The estimated variance of α, 0.175, is much smaller than that for ϵ, 1.073.

Of course, much more sophisticated analyses can be done, adding a variance component for the movies, accounting for the different movie genres and so on.

Chapter 8

Shrinkage Estimators

Chapter 9

Dimension Reduction

Chapter 10

Smoothing-Based Nonparametric Estimation

In previous chapters, we often made use of *k-Nearest Neighbor* (kNN) regression analysis. This considered a *nonparametric* approach, because we do not assume any parametric model for the regression function, $\mu(t) = E(Y|X = t)$.

In addition, kNN is known as a *smoothing* method. In taking the average of Y values near the point $X = t$, we are "smoothing" those values. A very close analogy is that of photo editing software, such as the open-source GNU Image Manipulation Program (GIMP). An image may have some erroneous pixels here and there, so we replace every pixel by the mean of its neighbors, thus "smoothing out" the image. That way we capture the overall trend of the image in that little section.

We might do something a bit more sophisticated, such as using the median neighbor value, rather than the mean. This would likely eliminate stray pixels altogether, although it would likley give a somewhat rougher image.

And that brings up another question: How much smoothing should we do, i.e. how many nearest neighbors should we use? The more neighbors we use, the smoother our image — but the more we lose some of the fine details of an image. In an image of a face, for instance, if our neighborhood of an eye is so large that it also includes the chin, then we've homogenized the entire face, and have a useless image (unless we are trying to hide the person's identity!). Too little smoothing, on the other hand, will mean that those rogue pixels may still stand out.

So, we wish to find a "happy medium" value for our number of neighbors. This is easier said then done, and we will look into this in some detail in this chapter.

We will also look at variations of kNN, such as *kernel* methods. These are in a sense the opposite of kNN. The latter looks at a fixed number of neighbors, but the radius of the neighborhood is random, due to our having a random sample from a population. With kernel-based estimation, we fix the radius of the neighborhood, which means the number of neighbors will be random.

In Chapter 11, we will study another kind of nonparametric regression analysis, not smoothing-based. Actually, we will refer to it as *quasi-parametric* rather than nonparametric, but that can wait. Let's take a look at smoothing methods now.

10.1 Kernel Estimation of Regression Functions

WE COVER THIS BECAUSE (A) IT IS ANOTHER WIDELY USED TECH, AND (B) IT IS EASIER TO TREAT MATHEMATICALLY

10.1.1 What the Theory Says

10.2 Choosing the Degree of Smoothing

XVAL, PLUGIN ETC.a

10.3 Bias Issues

LINEAR SMOOTHING, LOESS ETC.

10.4 Convex Regression

MAYBE

10.4.1 Empirical Methods

Chapter 11

Boundary-Based Classification Methods

Chapter 12

Regression and Classification in Big Data

Chapter 13

Miscellaneous Topics