

Statistical Rethinking

A MOSTLY BAYESIAN COURSE
IN
MOSTLY NON-BAYESIAN STATISTICS

Richard McElreath

©2011

Contents

Preface	7
How to use this book	9
Acknowledgments	13
Chapter 1. Three Tyrannies and Three Frontiers	15
1.1. Truth and Reconciliation	15
1.2. Three Tyrannies	17
1.3. Three Frontiers	27
1.4. The R Environment for Statistical Computing	28
1.5. Statistical Rethinking	29
Chapter 2. Models, Estimation and Confidence	31
2.1. Some kind of intro	31
2.2. Hypotheses and Models	31
2.3. Bayes' theorem: A conditioning engine	36
2.4. Grid Approximation	46
2.5. Justifying Maximum Likelihood	49
2.6. The Bayesian Method of Maximum Likelihood	58
2.7. Confidence: More Than the Maximum	74
2.8. Confidence Intervals Are Not Significance Tests	100
Chapter 3. The Tyranny of Fisher	103
3.1. Defining the P -value	105
3.2. Hopeful Illusions	108
3.3. Comparing P to the Naive Posterior	111
3.4. The need to develop the alternative model	127
3.5. How to Interpret P -Values, If You Must	128
3.6. Defenses of P -values	135
3.7. Checking assumptions	136
Chapter 4. Building Models	137
4.1. A Normal World	137
4.2. From Parameters to Linear Models	142
4.3. Transformation and Log-Normal Regression	176
4.4. Linear Non-linear Models	184
4.5. Multivariate Models	191
4.6. Categorical Variables	216

4.7. Centering Variables	223
4.8. Ordinary Least Squares	223
4.9. Goodbye Sigma, Hello Tau	227
 Chapter 5. Comparing Models	233
5.1. Firing the Weatherperson	233
5.2. Model Families	236
5.3. The Problem with Parameters	238
5.4. Bayes' Theorem and Model Families	251
5.5. BIC	258
5.6. AIC	260
5.7. Comparing AIC and BIC	267
5.8. Using BIC and AIC	272
5.9. Comparing Estimates	289
5.10. Translating Hypotheses into Model Families	294
5.11. Fitting All Possible Models	294
5.12. Common Questions	296
 Chapter 6. Interactions	297
6.1. Double Jeopardy	297
6.2. Building an Interaction	301
6.3. Anatomy of the Linear Interaction	312
6.4. Continuous Interactions and the Value of Centering	315
6.5. Higher-order interactions	328
6.6. Multicollinearity returns	328
 Chapter 7. Generalized Linear Models I: Meet the Family	329
7.1. The Tyranny of Gauss	329
7.2. Generalized Linear Models	338
7.3. The Exponential Distribution	344
7.4. Gamma	346
 Chapter 8. Generalized Linear Models II: Counts	347
8.1. Fingers and Toes	347
8.2. Binomial	347
8.3. Poisson	361
8.4. Multinomial	367
 Chapter 9. Generalized Linear Models III: Monsters and Mixtures	369
9.1. The Golem	369
9.2. Ordered Categorical Outcomes	369
9.3. Ranked Outcomes	386
9.4. Variable Probabilities: Beta-binomial	386
9.5. Variable Rates: Gamma-Poisson	399
9.6. Variable Process: Zero Inflated Outcomes	406

Chapter 10. Multilevel Models	407
10.1. The musician who forever forgets	407
10.2. What are multilevel models good for?	408
10.3. Multilevel tadpoles	410
10.4. Varying effects and the bias-variance tradeoff	415
10.5. Everything can vary and probably should	424
10.6. Wrestling with lemurs	431
10.7. The importance of variance priors	437
10.8. Centering Strikes Back	442
10.9. How many parameters?	442
Chapter 11. Markov Chain Monte Carlo Estimation	449
11.1. Fortuna Strikes Back	449
11.2. Good King Markov and His Island Kingdom	450
11.3. Multilevel Markov chains	469
11.4. BUGS, JAGS and DAGs	478
Chapter 12. Missing Values and Other Errors	481
Endnotes	483
Bibliography	493
Index	497

Preface

This book is a course in Bayesian statistics, but the statistics aren't very Bayesian. It focuses on the typical tools of early 21st century statistical modeling: regression using linear models, generalized linear models (GLM's), and generalized linear mixed models (GLMM's). These are the tools that every PhD in the social and natural sciences needs to know. These tools are really frameworks, extremely productive and useful frameworks. The cost of these tools is that they are usually quite causally distant from the systems they describe. But the benefit is that they are relatively user friendly and widely available.

These are not normally seen as Bayesian tools. They are almost always estimated using maximum likelihood or approximations of it. But it is easy to see these tools as valuable parts of Bayesian inference about evidence, even when estimated by raw likelihood calculations. Most of statistical inference is Bayesian, because most of our tools can be seen as special cases of Bayes' theorem. For example, maximum likelihood is a special use of Bayes' theorem, in which we assume a flat prior probability density and search only for the parameter value for which the posterior has a maximum. The Bayesian posterior can then be estimated, provided the sample size is sufficient, by using the variance-covariance matrix to define a multivariate normal posterior density. This is most definitely not how Fisher justified maximum likelihood inference, but it is how this course justifies it.

Why bother re-teaching regression in this way? The value of foregrounding the Bayesian interpretations is that it both makes statistics coherent and renders the later introduction of useful priors more natural. But instead the typical statistics book introduces Bayesian methods at the end, and most teachers omit them, because of time constraints. This is a shame, because Bayes' theorem helps us understand how different methods relate to and complement one another. In other words, if we don't start with Bayes' theorem, we have to accept a kind of statistics farrago—a confused and confusing mixture of approaches and habits. That is how statistics is usually taught. I suffered through the farrago myself. I don't wish to inflict that pain on my own students.

Many of the students I teach and advise are seeking PhD's in population biology and ecology, and so many of them are familiar with another unifying theorem called the Price equation. My first book spent considerable time teaching this equation.¹ It served a similar conceptually unifying role in that book to what Bayes' theorem does in this one. The Price equation is an identity that allows all evolutionary models to be related to one another. Being so general however, in order to make causal interpretations from it, one has to make assumptions, which reduces the generality of the results. Indeed, I don't think the Price equation itself actually produces "results." Instead, models derived from the Price equation produce results. Still, the value of Price's equation lies partly in providing a universal way to understand which approximations differentiate models and their results. All of them are special cases of the Price identity. In that sense, all of evolutionary biology is "Price-ian," in the same sense that much of statistics is "Bayesian," or more accurately, Laplacian.²

The other major reason to be Bayesian about "non-Bayesian" statistics is because it makes it *easier* to explain concepts. Usually Bayesian statistics has a kind of aura of difficulty about it. Books on Bayesian statistical inference are full of integrals and other symbols that make people run away. But the difficulties of teaching things like frequentist confidence intervals convinced me that the Bayesian framing is actually more intuitive. I think this intuition is why scientists tend to mistakenly interpret frequentist estimates and intervals in Bayesian terms, something I call the *inverse illusion*. They really really want to make inferences about the posterior (inverse probabilities), so much that they unconsciously interpret frequentist tools as if they were Bayesian. Indeed, many statistics books implicitly define P -values as inverse probabilities, and many scientists therefore believe it.³

Instead of trying to teach people weird frequentist interpretations, and finding our students sliding towards crypto-Bayesian interpretations, why not just ignore frequentist interpretations to begin with? There are benefits to this approach. One benefit of being explicit about inverse probability is that teaching students why a P -value is not the "probability of the null hypotheses being true" arises naturally, once they know Bayes' theorem. Under the frequentist injunction that we not talk about the probability of a hypothesis, this cannot easily be made clear. More broadly, many standard tasks in model interpretation—such as estimating confidence intervals and regions—are made much easier, once you

teach students to sample from the posterior density. It makes a mathematical problem into an empirical problem. This empowers the user, rather than the software.

As a result of this strategy, you won't find this book tying itself in knots over the proper interpretation of a frequentist confidence interval and how it differs from a credible interval. Practicing scientists are perfectly capable of learning the Bayesian framing from the start, as well as which assumptions are needed to get them to their posterior estimates. Practicing scientists don't care about philosophy; they care about results. Bayesian approaches produce results in important cases for which classical approaches can barely get started.⁴ We can either teach convoluted frequentist interpretations that they will distort in their memories, or we can teach them credible Bayesian interpretations of familiar tools, so they can appreciate when these tools fail and why inverse probability is useful.

Of course, we also much teach that inverse probability isn't everything. Throughout the book, I emphasize checking model assumptions and predictions. It makes no sense to falsify the model, because all models are fictions. But it does make sense to evaluate the performance of the model against the data. These evaluations often make use of Bayesian inferences, but they aim at evaluating the usefulness of the model, rather than merely comparing models to one another. Our models live in small worlds of deduction, within which the Bayesian approach has some important advantages. But science lives in the large world, in which no method—Bayesian or otherwise—can claim to solve all problems. Thus the emphasis is always on shifting focus back and forth between the small world of posterior probability and the large world of model criticism. However, the Bayesian approach also has a key advantage in this area, because Bayesian statistical estimates are always model based and so always generative of predictions. In contrast, many good statistical approaches, such as Generalized Estimating Equations (GEE), have no probability model and so cannot produce predictions.

How to use this book

This book has a lot of code in it, integrated fully into the main text. The reason for this is that doing model-based statistics in the 21st century really requires programming, of at least a minor sort. The code is not optional. In many places, I have removed mathematical demonstrations and replaced them with calculations, implemented in R code, that allow the reader to better appreciate the consequences of a procedure.

Everyplace, I have erred on the side of including too much code, rather than too little. In my experience teaching scientific programming, novices learn more quickly when they have working code to modify, rather than needing to write an algorithm from scratch. My generation was probably the last to have to learn some programming to use a computer, and so coding has gotten harder and harder to teach as time goes on. My students are very computer literate, but they have no idea what computer code looks like.

What the book assumes. This book does not try to teach the reader to program, in the most basic sense. It assumes that you have made a basic effort to learn how to install and process data in R. In most cases, an online crash course in R programming will be enough. I know many people have found Emmanuel Paradis' *R for Beginners* helpful. You can find it and many other beginner guides here:

<http://cran.r-project.org/other-docs.html>

To make use of this book, you should know already that `y<- 7` stores the value 7 in the symbol `y`. You should know that symbols which end in parentheses are functions. You should recognize a loop and understand that commands can be embedded inside other commands (recursion). Knowing that R *vectorizes* a lot of code, instead of using loops, is important. But you don't have to yet be confident with R programming.

Inevitably you will come across elements of the code in this book that you haven't seen before. I have made an effort to explain any particularly important or unusual programming tricks in my own code. In fact, this book spends a lot of time explaining code. I do this because students really need it. Unless they can connect each command to the recipe and the goal, when things go wrong, they won't know whether it is because of a minor or major error. The same issue arises when I teach mathematical evolutionary theory—students and colleagues often suffer from rusty algebra skills, so when they can't get the right answer, they often don't know whether it's because of some small mathematical misstep or instead some problem in strategy. The protracted explanations of code in this book aim to build a level of understanding that allows the reader to diagnose and fix problems.

Using the code. Code examples in the book are marked by a shaded box, and output from example code is often printed just beneath a shaded box, but marked by a fixed-width typeface. For example:

```
print( "All models are wrong, but some are useful." )
```

R code
0.1

```
[1] "All models are wrong, but some are useful."
```

The intention is that the reader follow along, executing the code in the shaded boxes and comparing their own output to that printed in the book.

Next to each snippet of code, you'll find a number that you can search for in the accompanying code snippet files, available from the author's website (Not Yet Implemented). This means you shouldn't have to spend time typing the code, but can just copy it from these files into R and keep moving. I really really really want you to execute the code, because just as one cannot learn martial arts by watching Bruce Lee movies, you can't learn to program statistical models by only reading a book. You have to get in there and throw some punches and, likewise, take some hits.

The command line is the best tool. Programming at the level needed to perform 21st century statistical inference is not that complicated, but it is unfamiliar at first. Why not just teach the reader how to do all of this with a point-and-click program? There are big advantages to doing statistics with text commands, rather than pointing and clicking on menus. Foremost among them is that each analysis documents itself, so that years from now you can come back to your analysis and replicate it exactly. You can re-use your old files and send them to colleagues. Pointing and clicking, however, leaves no trail of breadcrumbs. A file with your R commands inside it does. Once you get in the habit of planning, running, and preserving your statistical analyses in this way, it pays for itself many times over.

We don't use the command line because we are hardcore or elitist. We use the command line because it is better. Unlike the point-and-click interface, you do have to learn a basic set of commands to get started with a command line interface. However, the costs later are reduced. With point-and-click, you pay down the road, rather than initially. You'll see that the command line makes some things that are prohibitively difficult for point-and-click interfaces, like recoding large data tables or running 10-thousand resampled regressions, trivial.

How you should work. But I would be cruel, if I just told the reader to use a command-line tool, without also explaining something about how to do it. You do have to relearn some habits, but it isn't a major change. For readers who have only used menu-driven statistics software

before, there will be some significant readjustment. But after a few days, it will seem natural to you. For readers who have used command-driven statistics software like Stata and SAS, there is still some readjustment ahead. I'll explain the overall approach, first. Then I'll say why even Stata and SAS users are in for a change.

First, the sane approach to scripting statistical analyses is to work back-and-forth between two applications: (1) a *plain text editor* of your choice and (2) the R program itself. A plain text editor is a program that creates and edits simple formatting-free text files. Common examples include Notepad (in Windows) and Text Edit (in Mac OS X) and Emacs (in most *NIX distributions, including Mac OS X). You will use the plain text editor to keep a running log of the commands you feed into the R application for processing. You absolutely do not want to just type out commands directly into R itself. Instead, you want to either copy-and-paste lines of code from your plain text editor into R, or instead read entire script files directly into R. You will always of course enter commands directly into R as you explore data or debug or merely play. But your serious work should be implemented through the plain text editor. There are several practical reasons for this.

First, editing commands in R is awkward. Inevitably, you will make typos. Editing a complex line of code in R to find and fix a single misplaced character is a pain. In contrast, in your plain text editor, you can easily move the cursor to any position to edit, without having to fight with the R command line at the same time.

Second, you can see the whole picture in the text editor, because it separates input from output. The text editor holds the plan of action. You want to plan your strategy and build the commands to implement it there, free from the clutter of intermediate outputs and warning messages and other vomitus of the R application itself. While it is possible to save a log of commands input into R, this hardly helps with planning.

Third, you want a secure and portable record of your work. Once you solve a scripting problem once, you can consult your previous scripts to remember how you did it. And when a colleague asks you how you did an analyses, you can just email them the script. The only field I know of in which such a practice is actually common is economics (I believe AER enforces this policy?), but it should be the norm everyplace. The format should be portable, so that any colleague can open it on any compute system. Plain text files fit this bill. MS Word files do not. Additionally, a complex text processor like MS Word actually gets in the way, because it forces formatting and spell checking that has no effect on coding. You want a plain text editor to show every space you type.

You can add comments to your R scripts to help you plan the code and remember later what the code is doing. To make a comment, just begin a line with the # symbol. To help clarify the approach, below I provide a very short complete script for running a linear regression on one of R's built-in sets of data. Even if you don't know what the code does yet, hopefully you will see it as a basic model of clarity of formatting and use of comments.

```
# Load the data:  
# car braking distances in feet paired with speeds in km/h  
# see ?cars for details  
data(cars)  
  
# fit a linear regression of distance on speed  
m <- lm( dist ~ speed , data=cars )  
  
# estimated coefficients from the model  
coef(m)  
  
# 95% confidence intervals of the estimates  
confint(m)  
  
# plot residuals against speed  
plot( resid(m) ~ speed , data=cars )
```

R code
0.2

Finally, even those who are familiar with scripting Stata or SAS will be in for some readjustment, with R. Programs like Stata and SAS have a slightly different paradigm for how information is processed. In those applications, procedural commands like PROC GLM are issued in imitation of menu commands. These procedures produce a mass of default output that the user them sifts through. R does not behave this way. Instead, R forces the user to decide which bits of information she wants. One fits a statistical model in R and then must issue later commands to ask questions about it. This more interrogative paradigm will become familiar through the examples in the text. But be aware that you are going to take a more active role in deciding what questions to ask about your models.

Acknowledgments

...

1

Three Tyrannies and Three Frontiers

1.1. Truth and Reconciliation

I like to begin my graduate statistics courses by having students answer a short quiz, always containing the question:

What is the definition of a P -value?

Typically, one or two people in a class of about 20 PhD students, all of whom have excelled in previous courses in statistics and many of whom have published statistical analyses in top-tier journals, can answer this question correctly. When I reveal the correct definition (see Chapter 3) and the fact that few of the students in the room could provide it, there is widespread surprise. Some students were aware of their own gaps in knowledge but believed they were the exception. Sometimes called the *impostor syndrome*, this kind of pluralistic ignorance is common among academics at all ranks. Judging from surveys, even many teachers of statistics cannot correctly define a P -value.⁵ If the reader isn't confident of the correct definition, you are in good company. Luckily, once we establish our collective ignorance, the learning in earnest can begin.

Since science is a social phenomenon, individual scientists should not be held individually responsible for believing collective illusions nor for performing the damaging rituals of their culture. The responsibility is collective. Too often, statisticians adopt the role of a scolding authority. Practicing scientists were taught, by prestigious PhD programs, that a P -value is the probability that the null hypothesis is true, so we can hardly blame them for believing it. It is our job to correct such misunderstandings, especially since they can have damaging consequences for knowledge. But we need to foster a social environment in which scientists feel comfortable admitting their lack of understanding of statistical practice. It is partly out of fear of looking stupid that scientists don't ask deeper questions about their statistics. It is partly out of fear of seeming dumber than their colleagues that scientists don't often enough ask for help with their quantitative evidence. These anxieties of course exist to

some extent in all domains of scientific society. But they are magnified in the case of statistics, due to the dread and aura of genius surrounding mathematics.

Let's have a good cry and a loud shout and solid night's sleep and then get on with the project of rethinking the norms of statistical practice. There is sin enough to go around, from the educators who continue teaching statistical practices that they don't believe in, to the journal editors who demand statistical practices long discredited within statistics, to the scholars who are too afraid do anything different, to the reviewers who punish any paper not reporting P -values. At fault also are the numerous statistics textbooks that refuse to take a strong position against P -values, even though their authors understand the philosophical and sociological flaws with null-hypothesis significance testing.

Sharing the blame are degree programs happy to outsource their teaching of statistics to Statistics departments in which applied statistics is something of a foreign country. It is often overlooked that most instructors of introductory statistics courses do not themselves do science. So even though many of these instructors care deeply about student outcomes, they are often poorly qualified to teach applied statistics. Every field of science has its own kinds of evidence and traditions of analysis and standards of proof. Every different speciality has its own concerns with bias and measurement. An experimental physicist and a plant geneticist work with very different kinds of measurement and models. A theoretical statistician cannot be expected to have mastered these distinctions, especially when the incentives in their field encourage fancy theoretical results over applied work and collaboration with scientists.

Whatever the reasons, undergraduate courses in statistics tend to present a mythological package of procedures called nothing other than "Statistics." Approaches such as Neyman-Pearson hypothesis testing— α and β error rates—and Fisherian P -values are taught blended together. Those approaches are logically incompatible, but we can be forgiven for thinking otherwise, because the textbooks tend to present a monolithic, deductive "Statistics" rather than delve into the contentious and entertaining and educational history of statistics. My hunch is that students are partly to blame for this, because any topic involving mathematics leads many to think there is a single correct approach, and that it can be found deductively. And so we have encouraged educational narratives that present a mythology of a progressive, uncontroversial body of statistics. The real history of probability theory and statistical inference is much more entertaining, if less progressive and mythological.⁶

And of course, any textbook that does not teach P -values will find a much smaller market than one that revolves around them. The book you are currently reading is not shy, as you have already seen. It is very opinionated, and one of its opinions is that P -values, as they are usually employed, have no place in applied statistics. But I do not wish to blame those who have used P -values, as if only dumb people would do such a thing. Instead, there has been an impersonal, distributed conspiracy of forces that has lead to the widespread use and misunderstanding of P -values. Certainly Ronald Fisher himself would not approve of the way P -values are now used. The way to expose this accidental conspiracy is to talk about it. We all share in the blame and we will all share in the cure.

We need to encourage statistical thinking, rather than statistical ritual. And that is made much easier when statistical inference is unified, rather than comprising a farrago of tests and estimators. Bayesian inference points to one way to provide such a unification. And it is by now obviously to nearly everyone that Bayesian thinking is powerful, making possible the estimation and interpretation of problems that frustrate non-Bayesian approaches. But there is usually much to unlearn, before students and scientists can learn the Bayesian approach.

1.2. Three Tyrannies

This book doesn't have all the answers. It promotes the view, above all else, that statistical analysis is just like every other part of science: it is part of a conversation about the uncertain connections between experience and belief. As a consequence, there are many good and complementary ways to analyze quantitative evidence. But there are also many more bad ways. Commonplace habits like the failure to report effect sizes and misinterpretation of P -values actively retard understanding. We can do better.

What is wrong with commonplace statistical practice? There are three contemporary tyrannies, patterns of counter-productive ritual thought and practice that dominate large constituencies in the sciences.

The Tyranny of Popperism: The perspective that the only proper goal of science, and therefore statistical practice, is to disprove hypotheses.

The Tyranny of Fisher: The near-exclusive reliance upon and nearly universal misunderstanding of null hypothesis significance testing, P -values.

The Tyranny of Gauss: The dominance of models and tests based on normal distributions, often when they are highly inappropriate, or the resort to “non-parametric” randomization tests otherwise.

None of these tyrannies rule over all scientists, but each is common someplace in the sciences. None of these tyrannies can honestly be blamed on Karl Popper, Ronald Fisher or Carl Friedrich Gauss. Still, it makes sense to use their names, because each made important contributions that have contributed, probably to their horror, to each pattern. No careful reading of Popper, Fisher, nor Gauss would lead one directly to these tyrannical patterns of thought and behavior. But in the case of the first two tyrannies, scientists commonly invoke Popper and Fisher to justify their behavior. Therefore I do not label these tyrannies with the names of Popper, Fisher and Gauss out of disrespect, but rather to direct the reader’s attention to the ritual practices that the contributions of these great thinkers have become trapped within.

Let’s spend a little time now defining each tyranny. In later chapters, we’ll get into each in much more detail.

1.2.1. The Tyranny of Popperism. A first tyranny arises from interpretations of Karl Popper’s philosophy of falsificationism. Most scientists, if they know any philosophy of science at all, have heard of Karl Popper. As he has been translated among practicing scientists, Popper argued that the proper goal of science is to falsify models. Therefore, the reasoning goes, the purpose of inferential statistics is to falsify models, because otherwise statistics would not advance science. The procedures we use to pursue this falsification are called “statistical tests.” Models that fail such tests are falsified, while those that do not fail remain unproven. Popper’s own views were or became much more subtle than this, and so I call this view *Popperism* to indicate that Popper himself did not quite endorse it.⁷

A compelling story, used by Popper himself, that encourages this view concerns the color of swans. Before Europeans voyaged to Australia, all swans that any European had ever seen or read about had white feathers. This lead to the belief that all swans are white. Let’s call this a formal hypothesis:

$$H_0: \text{All swans are white.}$$

When Europeans reached Australia, however, they encountered swans with black feathers. This evidence seemed to instantly prove H_0 to be false. Indeed, not all swans are white. Some are certainly black, according to all observers. The key insight here is that, before voyaging

to Australia, no number of observations of white swans could prove H_0 to be true. However it required only one observation of a black swan to prove it false.

This is a seductive story. If we can believe that important scientific hypotheses can be stated in such forms, then we have a powerful method for improving the accuracy of our theories: look for evidence that disconfirms our hypotheses. I do believe that seeking disconfirming evidence is important, but it cannot be as powerful as the swan story makes it appear. Most of the problems scientists confront are not so logically discrete. Instead, we most often face two simultaneous problems that make the swan fable misrepresentative. First, observations are prone to error, especially at the boundaries of scientific knowledge. Second, most hypotheses are quantitative, concerning degrees of existence, rather than discrete, concerning total presence or absence. Let's briefly consider each of these problems.

All observers will agree under most conditions that a swan is either black or white. There are few intermediate shades, and most observers' eyes work similarly enough that there will be little, if any, disagreement about which swans are white and which are black. But this kind of example is hardly commonplace in science, at least in mature fields. Instead, we routinely confront contexts in which we are not sure if we have detected a disconfirming result. At the edges of scientific knowledge, the ability to measure a hypothetical phenomenon is often in question as much as the phenomenon itself.

Here are two varied, recent examples. In 2005, a team of ornithologists from Cornell claimed to have evidence of an individual Ivory-billed Woodpecker (*Campephilus principalis*), a species thought extinct. The hypothesis implied here is:

$$H_0: \text{The Ivory-billed Woodpecker is extinct.}$$

It would only take one observation to falsify this hypothesis. However, many doubted the evidence. Despite extensive search efforts and a \$50,000 cash reward for information leading to a live specimen, no evidence satisfying all parties has yet (by 2011) emerged. Even if good physical evidence does eventually arise, this episode should serve as a counter-point to the swan story. Finding disconfirming cases is complicated by the difficulties of observation. Black swans are not always really black swans, and sometimes white swans are really black swans. There are mistaken confirmations (false positives) and mistaken disconfirmations (false negatives). Against this background of real measurement

difficulties, scientists who already believe that the Ivory-billed Woodpecker is extinct will always be suspicious of a claimed event of falsification. Those who believe it is still alive will tend to count the vaguest evidence as falsification.

Another recent example, this one from physics, focuses on the detection of faster-than-light (FTL) neutrinos.⁸ In September 2011, a large and respected team of physicists announced detection of neutrinos—small neutral sub-atomic particles able to pass easily and harmlessly through most matter—that arrived from Switzerland to Italy in slightly faster-than-lightspeed time. The dominant reaction from the physics community was not “Einstein was wrong!” but instead “How did the team mess up the measurement?” You might think measuring speed is a simple matter of dividing distance by time. It is, at the scale and energy you live at. But with a fundamental particle like a neutrino, if you measure when it starts its journey, you stop the journey. The particle is consumed by the measurement. So more subtle approaches are needed. The detected difference from lightspeed, furthermore, is quite small, and so even the latency of the time it takes a reading to travel from a control room can be orders of magnitude larger. And since the “measurement” in this case is really an estimate in a statistical model (see further down as well as in next chapter), all of the assumptions of the model are now suspect. As I write this in late 2011, the physics community is very confident that the FTL neutrino result is measurement error. Neutrinos clocked from supernova events are consistent with Einstein, and those distances are much larger and so would reveal differences in speed much better. But if the FTL neutrino measurement is never replicated exactly by another lab, we may never figure out what the error was.

In both the woodpecker and neutrino dramas, the key dilemma is whether the falsification is real or spurious. Measurement is complicated in both cases, but in quite different ways, rendering both true-detection and false-detection plausible. Popper himself was aware of this limitation inherent in measurement, and it may be one reason that Popper himself saw science as being broader than Popperism. But the probabilistic nature of evidence rarely appears when practicing scientists discuss the philosophy and practice of falsification.⁹ My reading of the history of science is that these sorts of measurement problems are the norm, not the exception. For an accessible history of some measurement issues in the development of physics and biology, including early experiments on relativity and abiogenesis, I recommend Collins and Pinch’s *The Golem*.¹⁰

The other kind of problem for the swan story is that most interesting scientific hypotheses are not of the kind “all swans are white” but rather of the kind:

$$H_0: \text{Some swans are white.}$$

Or maybe:

$$H_0: \text{Black swans are rare.}$$

Now what are we to conclude, after observing a black swan? The task here is not to disprove or prove a vague hypothesis of this kind, but rather to estimate and explain the distribution of swan coloration as accurately as we can. Even when there is no measurement error of any kind, this problem will prevent us from applying the swan story to our science. You might object that the hypothesis above is just not a good scientific hypothesis, because it isn’t easy to disprove. But if that’s the case, then most of the important questions about the world are not good scientific hypotheses. In that case, we should conclude that the definition of a “good hypothesis” isn’t doing us much good. Now, nearly everyone agrees that it is a good practice to design experiments and observations that can differentiate competing hypotheses. But in many cases, the comparison must be probabilistic, a matter of degree, not kind.¹¹

The greatest practical problem with the simple falsification view, the Tyranny of Popperism, is that when it comes to quantitative evidence, we never use hypotheses directly but instead evaluate *models*. Models are mathematical specifications of hypotheses. They are narrower, almost always, than the hypotheses that inspire them. The issue with models is that, while they are very useful in the conduct of science, they are all false. We know they are false, because we construct them to be simpler than the complex reality we hope to explain. Therefore, if you think about it for a moment, falsifying hypotheses cannot be the goal of statistical analysis, because all models are false, and we know this before we have even collected one datum. However, models are certainly very useful. They don’t have to be exactly true to make handy predictions or shed light on causal relationships. But models cannot be falsified anymore than the play *A Midsummer Night’s Dream* can be.

What are we supposed to do then, if we can’t deductively falsify most hypotheses? What value is statistical analysis, if all statistical models are false? There are several satisfying ways to answer these questions. The one this book focuses on is to work towards accumulating models that make better predictions. That is, a statistical thinker should prefer the statement

The goal of statistical analysis is to help us decide which models are more useful.

over

The goal of statistical analysis is to help us decide which models are false.

All models are false. Only some of them are useful.¹²

A final complaint about Popperism is that it may reinforce the false belief that “the absence of evidence is not evidence of absence.”¹³ The only sense in which this claim is correct is when we have made no observations whatsoever. Then we shouldn’t draw any conclusions yet. But the usual sense is much broader and more dangerous: the claim is that we are not supposed to infer anything from the failure to find something. We are allowed however to make inferences when we do find something. You might see how this follows from the black swan story: Not finding a black swan does not mean they do not exist, but finding one confirms that they do. In paleontology, for example, the failure to find any rabbits in the Precambrian period¹⁴ is not logically evidence that rabbits did not exist at that time. But after many years of excavating Precambrian deposits, we should feel pretty confident that rabbits did not exist in the Precambrian. When I challenge this aphorism, that absence of evidence is not evidence of absence, my colleagues are almost always surprised. But even a casual familiarity with probability theory reveals that the repeated failure to observe something does provide information and the data generating process. Again, the mistake is to think about discrete proof rather than about probabilities. Of course we must take proper account of the sampling model and such, but none of that changes the fact that the failure so far to find any fossil rabbits from the Precambrian is pretty good evidence that they were absent in that period. Of course not all failures to find evidence will have the same inferential power. But it is manifestly fallacious to claim that nothing can be inferred from absence of evidence. If that were true, then few statistical procedures would work.

Does this philosophy stuff really matter? I think it does, if only because it leads scientists to neglect more powerful forms of analysis. The Tyranny of Popperism, especially as it is blended with the next tyranny, has the unfortunate side-effect of discouraging the careful specification of research hypotheses. If the goal is to disprove hypotheses, the thinking goes, then all we have to do to support our preferred hypothesis is to disprove other hypotheses. It’s a kind of last man standing version of epistemology that we should all reject. Showing that a bunch of models

fail in prediction doesn't mean that our favorite model succeeds. Likewise, showing that some null hypothesis can predict some result reasonably well does not mean that an alternative model is not better. Models are all false, and they must all compete on equal ground, demonstrating their utility in explicitly defined domains of experience. I am confident that Popper would have agreed with that statement, even if Popperism does not.

1.2.2. The Tyranny of Fisher. The Tyranny of Popperism, applied to quantitative reasoning, has perhaps encouraged another tyranny of statistics, the Tyranny of Fisher. If the proper goal of science is to falsify hypotheses, then we need a way for statistics to accomplish that goal. The Tyranny of Popperism seems to justify then the Tyranny of Fisher, which has a two part logic. First, reject a null hypothesis if the P -value is less than 5%. Second, always do this. Too often, to get published in most scientific journals that report empirical research, one has to compute P -values and show that they are less than 0.05. While editors and reviewers are increasingly respectful of those, like myself, who rebel against the Tyranny of Fisher, it is still dominant in both the natural and social sciences.

While he was himself critical of many of the aspects of the procedure as it is now practiced, Ronald Fisher is properly credited as the most influential of those statisticians who promoted the use of P -values, along with thresholds of "significance," for rejecting hypotheses. I spend an entire chapter (Chapter 3) on the topic of why P -values have little role to play in statistical thinking, at least in the role they currently play. The argument is hardly new; statisticians have been complaining about the abuse of P -values for decades now. Here, I only want to provide an abstract of the argument. There are two major aspects of the Tyranny of Fisher: (1) P -values are most often misunderstood, and (2) even when interpreted correctly, P -values are a bad way to choose parameter values or to compare models.

Several empirical studies have now found that most scientists, and even teachers of statistics, believe one or more false statements about significance testing.¹⁵ The most common mistake I encounter in my own areas of research is the mistaken definition that a P -value is "the probability that the null hypothesis is true" or, in the softly-worded versions my colleagues cough up when I confront them, "the chance of wrongly rejecting the null hypothesis" or even "the probability that the results are due to chance." None of these definitions are correct, but they share a common error: they reverse the hypothesis and the data inside a conditional probability.

Many of the other false beliefs about significance testing can be sourced to this one illusion. I call this *the inverse illusion*, because it gets the definition exactly backwards (inverts it) and unknowingly pretends to be Bayesian (the use of inverse probability). The actual definition of a P -value, and the quantity statistical software tries to estimate, is:

The probability of obtaining an observation or more extreme observation, assuming that the null hypothesis is true.

We will be mathematically precise about this definition in a later chapter. The key point is that this is the probability of data, under the assumption that some model is true. People tend to believe instead that P is the inverse of this, the probability of the model, given the data. We want to know whether the null hypothesis is true, but we have to assume it is true in order to calculate P . This complaint is not just the shrill cry of mathematicians complaining about the details. The procedures used to estimate P -values have a definite logic to them, and by getting the definition wrong, it leads scientists to believe that their statistics are more powerful than they actually are. The ritual is comforting, but it is a false comfort.

But even if we could correct everyone's understanding of the proper meaning of P -values—and statisticians have been trying for decades—significance tests as currently practiced would still not be useful in statistical thinking. The reason is that, once properly defined, the P -value provides unreliable help for deciding which hypothesis to prefer. There are several reasons for this, which we'll explore in detail in Chapter 3. Foremost, it is not clear at all why we should care about the probability of some unobserved data, in the first place. A P -value is calculated by imagining data that we have not observed and then asking how often the null hypothesis would predict these unobserved data. Don't we care instead about the probability of the hypothesis, holding the observed data constant?¹⁶

I do think that consideration of the patterns of observations implied by a model are useful. For example, no amount of Bayesian inference about probabilities of parameters is going to tell us that a Gaussian model of counts is hazardous. Bayes' theorem suffers from the same garbage-in-garbage-out syndrome of all theorems. However, plotting the implied data from such a model will immediately reveal its peculiar predictions, like negative counts. But what I am describing here is the use of implied predictions of a model, after some other method has allowed us to choose parameter values. P -values are a crummy way to choose parameter values. But something in their spirit may be useful in checking the

rationality of a model against key evidence. I will have much more to say about this distinction between choosing parameter values and checking models, later in the book.

For now, such considerations lead us to the final Tyranny, the over-reliance on normal distributions.

1.2.3. The Tyranny of Gauss. Before the advent of cheap fast computing, the only statistical analyses that were really practical to perform were those that relied on normal, or Gaussian, distributions. The normal distribution is easy to calculate with, and so a great deal of useful mathematics can be proved about it and done with it. These virtues help obviate the need for intensive computer calculations. As a result, many classic statistical procedures rely upon normal distributions or approximations to them. This was especially true of multivariate models, which when specified entirely using normal distributions can be fit to data with a few matrix operations. These operations can be numerically taxing, and so before fast affordable computing, they were impractical. In the 1950's, it could take days to compute an ordinary regression model with two explanatory variables. But it could be done. However, if the outcome variable is non-normal, there is no longer any universal formula that can be used to find parameter estimates. And so much more taxing algorithms are needed to perform them. Even then, there is no guarantee of finding a solution. When computers were slow, even in the 1970's, models like logistic mixed-effects regressions were impractical. They could be defined, but they could not be practically used. The Gaussian distribution usually defined the limits of what was practically computable. When it could not be used, analysts fell back on distribution-free approaches, so-called non-parametric tests. Such tests are easy to compute, but have very little power and allow for very few kinds of hypotheses.

Now that most people have mobile phones vastly faster and with more capacity than the computers that sent people to the Moon, the situation is quite different. Yet scientific culture can change very slowly, and what used to be Gauss' Magical Toolbox is now more properly the Tyranny of Gauss. Approximations to and models based upon the normal distribution are a necessary and productive part of statistical research. However, the Tyranny of Gauss is not the use of these tools, but rather over-reliance upon them, ignorance of models that do not assume normal distributions, and the fall back position of employing "non-parametric" tests. Like with the previous two tyrannies, not all scientists are victims of the Tyranny of Gauss. But many are. There are

two main types of victims: (1) those who use Gaussian models like analysis of variance (ANOVA) and linear regression (OLS) for everything and (2) those who recognize a proper domain for Gaussian models and so employ non-parametric tests, instead of non-Gaussian regressions (generalized linear models, for example). Neither of these victim categories should be blamed, because we shouldn't blame victims.

A transparent example of the first problem arises when people analyze count data. Counts are positive integers—1,2,3,4, and so on—and zero (0). This kind of data arises all the time, for example when archaeologists count stone tools or when demographers count babies or when astronomers count solar flares. Analyzing count data with a model that assumes a normal distribution generates a number of inferential problems. We will detail some of these problems in Chapter 8. Perhaps the easiest example to understand, and one that is very widespread in archaeology, is the conversion of counts to proportions. These proportions are then treated as outcome variables in standard Gaussian models. What is so bad about this procedure? Proportions are ratios of the count of a particular kind of thing to the total count of all kinds of things. Consider two such ratios: 1/2 and 10/20. Both of these ratios are equal to one-half, 0.5. But we should be much more confident in the estimate of 0.5 in the second case, because 5-times as much sampling was done. Any probability statements derived from a Gaussian model of proportions will be truly bizarre, because the model has no way of knowing how much sampling really went into the data. The analyst told the model that each proportion is a single sample, and so the model abides. Additional hazards arise when the proportions are near zero (0) or one (1). Normal distributions will happily predict negative values or values greater than one, even though as proportions those values are impossible. The immediate importance of this fact is that estimates will be distorted. Predictions will be nonsensical. All of this is a shame, because today nearly every available statistical package can fit multivariate count models.

Many scientists were trained to recognize when models based upon normal probabilities are misleading. They would never model proportions as above. Unfortunately, they were instead trained to use a toolbox of “non-parametric”—usually distribution free—statistical tests. Such tests have been especially popular in fields with chronically small sample sizes, like field zoology. These tests—rank correlations, sign tests, Wilcoxon, Kruskal-Wallace, and many others—are the other symptom of the Tyranny of Gauss. None of these procedures has anything to offer, in light of more modern, distribution-based models available on every

desktop computer. Usually all that a non-parametric test can do is estimate a P -value, further subjugating the analyst to the Tyranny of Fisher. So even if better models were not now available, non-parametric tests wouldn't help the statistical thinker very much.

1.3. Three Frontiers

Better approaches are now widely available, approaches that focus the mind on a model instead of a procedure. None of these "frontiers" is really new, but their widespread use is new, and PhD students learn quickly that they are expected to know something about them.

1.3.1. Bayesian statistics. The first frontier is the insurgence of Bayesian statistics.

Bayesian statistics is usually thought of as an advanced topic, but this is a shame. It is easier to teach statistical inference with a Bayesian foundation. Classical statistical concepts, like frequentist confidence intervals and P -values, are hard to teach. The analogous Bayesian tools are conceptually easier. So I adopt a Bayesian framing at the start here, to make the course easier, not harder. As a bonus, all statistical methods—all of them, including frequentist tools—are special cases of Bayesian inference. And so when you learn a little Bayes, you meet a unifying approach to learning about the world.

The word *Bayesian* means a lot of different things. Being Bayesian in the 20th century often meant getting into philosophical arguments about random variables and subjectivity. Being Bayesian in the 21st century often means instead leveraging Bayesian tools to help us fit models that cannot be easily estimated otherwise, or to patch up some of the erratic behavior of maximum likelihood. No one any longer denies that Bayesian methods work, because now our computers are powerful enough to unleash them. So that's the goal we work towards in this course, unleashing their power. We'll start with Bayesian concepts and then learn the very common and useful approach of maximum likelihood inference as a special case of Bayesian inference. Then when we reach MCMC and Gibbs Sampling (think BUGS) in the final week, it won't seem so weird. Bayes has been there all along.

1.3.2. Multilevel models. Natural relation between hierarchical models and Bayesian statistics. First, conceptual relation of embedding distributions in distributions. Second, procedural advantage for estimation, using Gibbs Sampling. How to talk about nuisance parameters? Seems like too confusing a topic at this stage in the book.

Multilevel models—aka hierarchical, mixed effects or random effects models—are becoming *de rigueur* in the biological sciences, and for good reasons. These models allow for much more satisfying models of evolutionary hypotheses, and they allow for powerful analysis of highly unbalanced and clustered data. Pseudo-replication and a bunch of other classical neuroses seem to crumble before this kind of model.

The parallel development of *generalized* linear models (really non-linear models) of measures and counts (gamma, beta, exponential, poisson, etc.) has had similarly powerful effects on our ability to statistically model natural phenomena. The classical world of strictly additive models of normal distributions and low-power non-parametric procedures is fading fast.

This is all good. But progress in software has outpaced progress in community knowledge, and it seems like many people who use these models do not yet understand them. We can easily and quickly fit complex, multilevel models to our data, but understanding these models is another affair. These models are much easier to understand, once you know a little Bayes, and so teaching mixed models benefits from a Bayesian foundation. The goal is to teach the student how to specify, fit and interpret these models.

1.3.3. Model comparison. Both Bayesian inference and mixed models are not really new—Bayesian inference is much older than frequentist inference. For example, Laplace used Bayes’ theorem in the early 1800’s to estimate the mass of Jupiter to within 1% of the modern estimate. These approaches are now on the rise partly because of cheap computational power and advances in software, making it possible for anyone to fit a GLMM or run a BUGS model.

But formal model comparison is a different story. Beginning seriously in the 1960’s and 1970’s, statisticians have developed a buffet of tools to use for comparing structurally different models of natural phenomena: AIC, BIC, and their many spawn. These metrics can be very confusing for people with training in classical statistics. All of them are, in principle, estimates of Bayesian posterior probabilities of models. So again, learning a little Bayes makes it easier to understand their strengths and weaknesses.

Main problem to address is overfitting.

1.4. The R Environment for Statistical Computing

Maybe make this a fourth frontier? R is important because it standardizes the platform on which statistics is done, accelerates innovation,

encourages collaboration, and forces the user to make analytical decisions.

1.5. Statistical Rethinking

1.5.1. Models instead of procedures.

1.5.2. Embrace and propagate uncertainty.

1.5.3. Small worlds and large worlds.

2

Models, Estimation and Confidence

This chapter is about how quantitative evidence can be linked to hypotheses. Models are a necessary bridge between measurement and hypothesis. Probability theory suggests that models can be compared using posterior probability. Maximum likelihood, and the calculation of confidence intervals, are built up as special cases of Bayesian inference from posterior probability. This chapter also introduces the pseudo-empirical approach of sampling models from posterior probability and using these samples to more-intuitively calculate confidence intervals and perform model checks.

R packages used in this chapter: `rethinking`, `bbmle`.

2.1. Some kind of intro

Fortuna vs Sapientia? Traditional view is that chance and wisdom are opposites. Modern statistical inference harnesses chance to produce wisdom.

2.2. Hypotheses and Models

That's enough philosophy for now. Let's estimate the proportion of the Earth covered in water, as a way of better understanding how models act within statistical analysis. I've probably made you nervous about two-dimensional maps by now. So instead of throwing darts at a map, let's use another sampling strategy. Suppose you have a globe in the room and adopt the following strategy. You will toss the globe up in the air. When you catch it, you will record whether or not the surface under your right index finger is water. Then you toss the globe up in the air again and repeat the procedure.¹⁷ No one believes that even this approach is completely unbiased—the globe is unlikely to be perfectly balanced, after all—but to keep the story simple and productive, we'll assume it is. (We'll confront measurement error later in the book.) This

strategy generates a sequence of surface samples from the globe. The first 10 samples might look like:

W L L W L W W W W L W

where W indicates water and L indicates land. Given this information, what are we to believe about the proportion of the Earth covered by water?

In order to answer the question of what proportion of water to be confident in—to believe in, to speak casually—we have to address what such beliefs would look like. Suppose we name the proportion of water p_W . Every possible belief about the true proportion of water corresponds to a different hypothesis. Each hypothesis corresponds to a different model of the process that generated the data.

2.2.1. Hypotheses. The hypotheses in this context are propositions about the true proportion of water on the Earth's surface. Possible hypotheses range from zero proportion of water to complete coverage. Some of these hypotheses are obviously silly, yes. But they are logically possible. Hypotheses imply certain patterns of observations, and so evidence can be used to distinguish among them.

2.2.2. Models. We cannot do statistics with hypotheses directly, however. Instead, we need models that correspond to the hypotheses. The importance of this realization is that models are narrower than typical hypotheses. This is because a model must include a number of assumptions about sampling and measurement that we tend to ignore when stating hypotheses.

For example, suppose you hypothesize that the proportion of water on the globe is $p_W = 0.71$. To make this into a model, we need a way to relate observations to the hypotheses. This means we have to make some description of how observations arise from this underlying truth, $p_W = 0.71$. Now we have to make some choices.

The standard position, and probably simplest model, in this kind of context is to assume that each sample from the globe has a chance p_W of turning up water. Furthermore, this implies that each toss of the globe is independent of the previous tosses. These assumptions lead to the common binomial sampling model, which we'll spend some quality time with a little later in this chapter. This is the common coin-flipping model of outcomes with only two discrete possibilities. When $p_W = 0.5$, the coin is fair. When p_W takes another value, the coin is biased, but the model remains the same, aside from the change in the underlying value p_W .

For now, the important point is that there are an infinite number of these models, one for each possible value p_W . Furthermore, there are many other assumptions about how the data arise that would change the model but leave the hypothesis unchanged.

2.2.3. Measuring confidence in models. So we have some models, now. What can we do with them? What we'd like to be able to do is say what the true proportion of water is. That is, we'd like to pick out one of these models and hold it up as true.

But clearly the evidence is not sufficient in this case, nor even in most cases, to select a single value of p_W to label *true*, while labeling all of the others *false*. In the sample at hand, for example, 6 out of 10 samples were W, indicating water. Only the most overconfident analyst would then conclude that $6/10 = 0.6$ is the true value of p_W , while all others are demonstrably false. This is because it would be quite easy for some other value, say $p_W = 0.7$, to generate the same observation. Moreover, we know a priori that all models are false, so it makes little sense to talk about true models and false models.

Instead, we require some intermediate measure of confidence in each model, in its usefulness. A sensible measure that turns out to be very productive in practice is the probability that each model is the best approximation—compared only to the other candidates—to the true underlying process. For a model with $p_W = 0.6$, call this probability:

$$\Pr(M_{0.6}|D),$$

where $M_{0.6}$ is a particular model with $p_W = 0.6$, and D is the evidence, the data. Note that the model is definitely not only $p_W = 0.6$, because there are other assumptions in the model. Still, the only assumption we're going to vary among the models in this example is the value of p_W .

Read the expression $\Pr(M_{0.6}|D)$ as *the probability of $M_{0.6}$, conditional on D* . This is intended as the probability that $M_{0.6}$ is the best approximating model, not the probability that $M_{0.6}$ is “true”. In later chapters, the difference this makes will become important (Chapter 5 and up). For now, it'll be alright to proceed with just a mental bookmark, a brain itch to remind you that this business about models being approximations does sometimes matter.

Now, if we have a model with $p_W = 0.6$, then the confidence in this model is measured by $\Pr(M_{0.6}|D)$. There are an infinite number of other models with other values of p_W . Each of them also has a measure of confidence, $\Pr(M_{p_W}|D)$. The entire ensemble of all of these models comprises a probability distribution, because if you add up all of the individual model confidence measures, $\Pr(M_{p_W}|D)$, they will add up to one.

This is just a result of the definition of probability. The value of such a distribution is that it produces a quantified measure of confidence in every model, and so it can be used to compare and produce many calculations that summarize what evidence says about our models. This is certainly not the only useful way to quantify what evidence tells us about the world, but it is a very productive one with a firm mathematical foundation and a long history.

FIGURE 2.1 illustrates what these ensembles of probabilities might look like. These distributions of confidence, $\Pr(M_{p_W}|D)$, are typically known as *posterior probability densities*. What exactly is “posterior” about them will be explained in a bit. The goal of the figure is just to deliver an impression what they might look like, given a single dimension along which models vary. In this case, that single dimension is the parameter p_W , displayed on the horizontal axis of each plot in Figure 2.1. The vertical axis in each plot is our measure of confidence, $\Pr(M_{p_W}|D)$. I have sketched these confidence distributions so that they all have a peak for the model where $p_W = 0.6$, which is the proportion of water in the raw sample on page 32. But even though they all have a peak at $p_W = 0.6$, their shapes are quite different otherwise.

In the top row, the distribution on the left is considerably wider than the one on the right. The left plot reflects greater uncertainty, because more models have higher probabilities. Narrower distributions, like the one on the right, reflect greater confidence in a narrower range of models. Notice also that the distribution on the left is skewed, extending further out to the left than to the right. The distribution on the right is instead almost perfectly symmetrical. This is another typical, although not necessary, feature of narrow confidence distributions of this kind: they tend to be symmetrical. Wide distributions, in contrast, frequently demonstrate skew. This skew reflects greater uncertainty in one direction than the other.¹⁸

Now inspect the bottom row of FIGURE 2.1. This row demonstrates that there is nothing preventing these distributions from having more than one peak. Again, the lefthand plot is much wider than the righthand plot, reflecting greater uncertainty across models. But both distributions have peaks at $p_W = 0.6$ and $p_W = 0.23$. It is a virtue of this approach to quantifying confidence in models that it allows for these sorts of odd distributions. We don’t wish to constrain the possibilities, at the outset. Instead, we wish to turn the crank, compute these probabilities, and see what they look like.

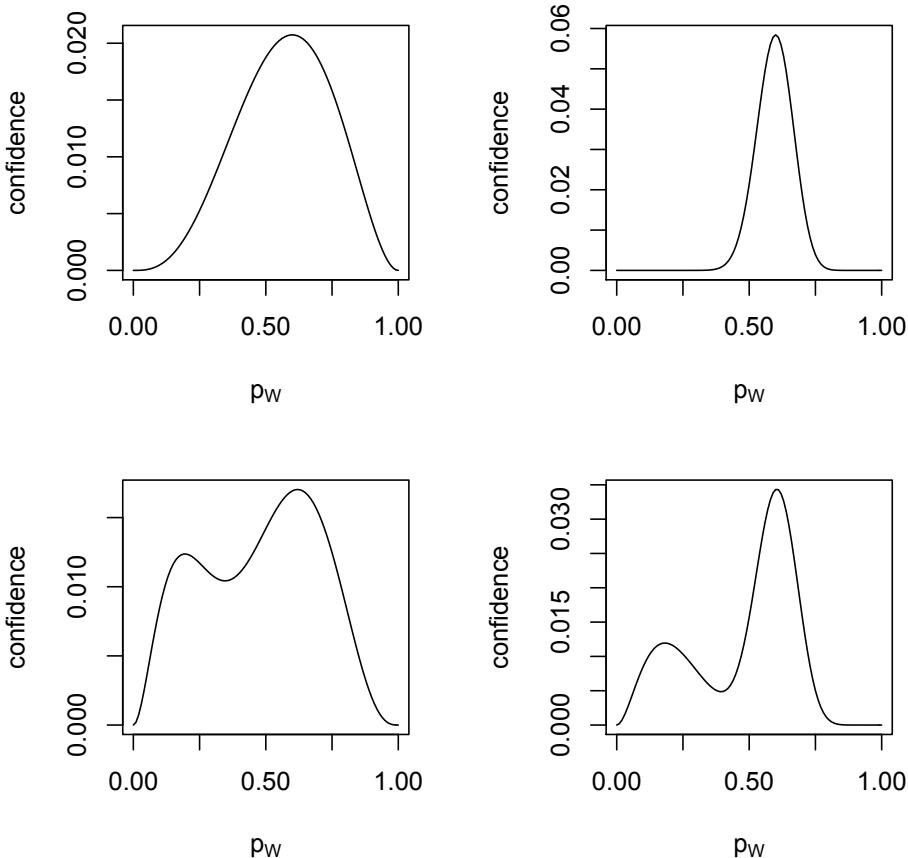


FIGURE 2.1. Example probability distributions $\Pr(M_{p_W} | D)$. These are commonly known as *posterior* probability densities. These distributions may be wide or narrow, reflecting differences in uncertainty across models. They may also be skewed or contain more than one peak.

2.2.4. Computing confidence in models. Okay, so how do you compute these probabilities, $\Pr(M_{p_W} | D)$? Well, since we are talking about a probability, you can just use the axioms of probability theory to define it. In this definition, there will be pieces that tell us what information we require in order to complete the analysis.

Here's the quickest way to derive a formula for $\Pr(M_{p_W}|D)$. You really should pay attention to this derivation, because working through it yourself will help you remember the formula for $\Pr(M_{p_W}|D)$. And if you ever forget the exact formula, you can just find it again, because the derivation really is that simple. Knowing how simple it is to produce this formula will also serve to impress upon you just how basic it is to probability theory. It's not a fancy result at all, despite its importance.

The probability $\Pr(M_{p_W}|D)$ is a *conditional probability*. It says that the probability of M_{p_W} is conditioned on D . The definition of conditional probability is embedded in a common probability rule:

$$\Pr(M_{p_W}, D) = \Pr(M_{p_W}|D) \Pr(D). \quad (2.1)$$

Read $\Pr(M_{p_W}, D)$ as *the probability of both M_{p_W} and D* . The above is true, no matter what M_{p_W} and D reference. So it's just as true that the inverse probability, $\Pr(D|M_{p_W})$, is defined by:

$$\Pr(M_{p_W}, D) = \Pr(D|M_{p_W}) \Pr(M_{p_W}). \quad (2.2)$$

Now since both (2.1) and (2.2) contain $\Pr(M_{p_W}, D)$, it must be true that:

$$\Pr(M_{p_W}|D) \Pr(D) = \Pr(M_{p_W}, D) = \Pr(D|M_{p_W}) \Pr(M_{p_W}). \quad (2.3)$$

Taking out the middle man:

$$\Pr(M_{p_W}|D) \Pr(D) = \Pr(D|M_{p_W}) \Pr(M_{p_W}). \quad (2.4)$$

All that remains is to use a little secondary school algebra and solve Equation 2.4 for $\Pr(M_{p_W}|D)$. Doing this, you arrive at the formula for computing the degree of confidence in model M_{p_W} , conditioned on the data, D :

$$\Pr(M_{p_W}|D) = \frac{\Pr(D|M_{p_W})}{\Pr(D)} \Pr(M_{p_W}). \quad (2.5)$$

Using the same formula for each model corresponding to each value of p_W , you arrive at the posterior distribution.

2.3. Bayes' theorem: A conditioning engine

The result in Equation 2.5 is usually referred to as *Bayes' theorem*.¹⁹ It tells us that in order to compute our measure of confidence in a model M_{p_W} , conditioned on the evidence D , we need to compute:

- (1) The *likelihood*, $\Pr(D|M_{p_W})$, which is the probability of observing D , conditioned on M_{p_W} generating the data.

- (2) The probability of the evidence, $\Pr(D)$, which is the probability of observing the data D , averaged over all models we'd like to consider.
- (3) The *prior* probability, $\Pr(M)$, which is the degree of confidence in M ignoring the evidence.

So re-expressed in this way, Bayes' theorem says that:

$$\text{Posterior} = \frac{\text{Likelihood}}{\text{Evidence}} \times \text{Prior.}$$

Again, the degree of confidence $\Pr(M_{pw}|D)$ is usually known as the *posterior* probability. Bayes' theorem is just a formula for computing this posterior probability, which provides one very useful way to compare models.

It is helpful to think of Bayes' theorem as an engine for conditioning one probability distribution, the prior, on some evidence. The result is a new probability distribution, the posterior. The posterior depends upon the prior, the evidence, and the assumptions embodied in how you calculate likelihoods of evidence. The engine requires the user—you—to provide these inputs, in order to do its work. It needs the prior, of course, and you feed this into it. The engine itself does not tell us what the prior should be, only that it must be a probability distribution. The engine also needs the likelihood of each model, in order to compute the likelihood, $\Pr(D|M)$, and the probability of the evidence, $\Pr(D)$. As you'll see a bit later, if you know how to calculate $\Pr(D|M)$, then you can at least define $\Pr(D)$ and perhaps calculate it too. These likelihoods contain lots of modeling assumptions, such as how sampling works and whether or not samples are independent of one another. This seems like a lot, I know, but there will be lots of examples to work with in this book. So hang on.

Given all of these things, the resulting posterior probability will be logically correct. But if you feed garbage into it, then it will provide only logical garbage out the other end. So like with all mathematics, its internal coherence is not going to save us from our external incoherence. As I argued in Chapter 1, statistical analysis about particular models lives in a small world, while science lives in a large world. We always work back and forth between these worlds, each teaching us about the other. I emphasize this point again and again, because it is tempting to think that the logical small world of probability thinking can somehow encompass all of scientific inference. This is not possible. Bayes' theorem, for example, is as deep and true a logical statement as you will

find in any branch of mathematics. But the theorem itself tells us nothing about what it should be used for, nor how to build assumptions into calculations of likelihoods, nor any number of other considerations.

Here's what you know so far. You want a measure of confidence in each model, p_W , the proportion of water on the Earth. The probability $\Pr(M_{p_W}|D)$ provides such a measure. To compute it, you need to use Bayes' theorem. Bayes' theorem requires two things to make it run: (1) the prior and (2) a way to compute likelihoods for every model under consideration. So now we turn to these two inputs.

2.3.1. Prior probability. The conditioning engine needs an input in order to produce a posterior probability density. It can't run without the prior. So what is this prior?

First, be careful to avoid the verbal trap of thinking that prior probability is somehow temporally antecedent to the posterior.²⁰ The prior is only logically antecedent. It has no necessary connection to philosophical concepts like Kant's *a priori* and *a posteriori*. Of course you can use Bayes' theorem in cases in which the prior does come before the posterior in a sequence of events. But this is not necessary. The posterior is just the prior, conditioned on some specified evidence. We might as well call them the conditioned and unconditioned distributions. But it's too late to change the terminology now. Just resist the urge to let the particular words we use to describe mathematics influence too much your impressions of the mathematics. Calling the "prior" as such demands nothing from us in terms of interpretation.

Where do priors come from? Remember, to define the prior probabilities, you must find a way to assign a probability to each model, before using Bayes' theorem. Since Bayes' theorem gives us no direct advice on how to do this, scientists have developed a number of different strategies. Both proponents and opponents of Bayesian statistics frequently speak of "the Bayesian" approach.²¹ This accidentally misrepresents the diversity of methods and epistemologies. All that every Bayesian statistician appears to agree upon is that posterior probabilities are useful.²²

These days, there are three major families of strategies for selecting a prior probability density, although others do exist.

- (1) Subjective: Sometimes it makes sense to adopt priors that represent individual personal beliefs. Such priors are *subjective* in the sense that two different analysts will likely adopt different priors and therefore may reach different conclusions. This

particular tradition of Bayesian analysis is responsible for scaring away many scientists who hold fast to the normative position that scientific procedures should instead be *objective*, in the sense that personal beliefs do not strongly influence the results. Perhaps as a result, the subjective approach to priors is mostly carried forward by philosophers of science. In the sciences, priors tend to be of the next type, although subjective priors are certainly not absent.²³

- (2) Objective: Instead of subjective priors reflecting personal impressions of confidence, one could instead use previous information or an objective procedure to determine the prior. In principle, when you have information about models, building this information into the prior is one legitimate and objective way to improve inference. Likewise, if ignorance is all you have, then you want an objective procedure for describing that ignorance. Objective Bayesian strategies allow for both. Of course different analysts may propose different priors under objective strategies, but the procedures will still be objective in the sense that they rely upon some argument about relevant information, not one premised on subjective beliefs. As a result, such arguments can be logically validated, replicated, and critiqued. There are different ways to derive objective priors, including the use of previous analyses, information theory, and something called transformation groups. Objective priors of some kind are typical of the majority of applied Bayesian analysis in the sciences.
- (3) Tuning priors: This is properly a subset of objective priors, but it is special enough that it deserves special mention. Sometimes, it is hard to estimate models. Learning about the world is hard. If you were a robot, and you were trying to navigate a room or catch a ball, it would be helpful to have some information, but not too much, built into your inference engine. If this information is useful, it will tune your learning to make it more efficient, to require less data, or to avoid particularly dangerous mistakes. Prior densities allow for us to tune statistical inference in this way.²⁴

These strategies are not necessarily warring tribes, forcing users to swear loyalty to a One True Strategy, forsaking all others. Instead, the same scientist may use all of them, at different times or even in the same analysis. In this book, we will focus on objective priors, mostly *weak* ones designed to have little impact on the posterior. Later in the book, we

will also look at the tuning abilities of weakly informative priors, useful for giving maximum likelihood a helping hand. The reason for this focus is that these two strategies now dominate much of the applied work in Bayesian statistics. And once you appreciate how they work, it will be easier to use strongly informative priors, when appropriate.

Since it won't appear again in the book, I want to be clear that the subjective approach to priors is hardly the boogie man some fear it to be. Suppose for example that we ask four different experts for their subjective prior densities.²⁵ These four experts produce four different priors, suppose. Now one of two things will happen. When we compute the posterior density using each of the four priors, we will either get the nearly same answer with all four or we will get different answers. In the first case, we will have learned that the variation in prior makes little difference. The evidence is overwhelming prior confidence estimates, and so none of the experts should be influenced too much by their subjective priors. In the second case, we learn that there is not enough evidence to overwhelm the subjective priors, and so we must pause to collect more data or reformulate the problem. In neither case are the priors snuck into the analysis, exerting covert influence over the outcome.

Whatever strategy one adopts for specifying a prior distribution, everyone agrees that one must be explicit and forthcoming about it. Many analysts test the sensitivity of their inferences to changes in the specification of the prior. And from this perspective, a problem with analyses that ignore Bayes' theorem is that they necessarily assume some kind of prior, despite the pretense that there is none. This means the prior in many classical statistical procedures is both hidden and has unknown influence on the results. It is better to be explicit about a prior than to be ignorant of the prior you are using.

2.3.2. Likelihood. The next part of the conditioning engine, Bayes' theorem, is the *likelihood*, $\Pr(D|M)$. The role of the likelihood is to link observations to the implications of models. Unlike posterior probability, $\Pr(M|D)$, which is usually hard to calculate, likelihoods are usually quite easy to compute. Or at least, once you have a little training, they are readily available.

2.3.2.1. Likelihood functions. Likelihoods are probabilities of data, assuming a particular model generated the data.²⁶ This is what makes likelihood a workhorse of model-based inference. Given the assumptions of a model, a trained applied mathematician can write down formulas that provide probabilities of different observations, conditional on those assumptions. Until you've done this yourself, it may seem like

a mysterious process. But learning to write likelihood functions from a list of assumptions is much easier than, say, learning a new language.

Like with learning a language, however, it is best to understand likelihood through examples. Let's tackle the proportion of water on the globe problem, as an example. We are assuming:

- (1) Each toss of the globe is independent of all others, in the sense that the outcomes do not influence one another.
- (2) The only possible results of a toss are “water” or “land.”
- (3) The probability each toss results in a “water” rather than a “land” is constant across tosses and equal to p_W .

These assumptions imply a familiar probability density, the binomial. Let n_W be the observed count of W's (“water”) in the sample and n be the total number of tosses of the globe (the size of the sample). These two values summarize the data, at least as far as the assumptions above require. If we were to assume that tosses were not independent, but instead influenced one another in order, then we'd need to know the exact sequence of W's and L's. But sticking with the simple binomial model, the probability of obtaining n_W out of n tosses is given by:

$$\Pr(n_W|n, p_W) = L(p_W|n, n_W) = \frac{n!}{n_W!(n - n_W)!} p_W^{n_W} (1 - p_W)^{n - n_W}.$$

This expression is known as a *likelihood function*. Likelihood functions are often represented by notation like $L(p_W|n, n_W)$, and most people speak of “the likelihood of the model” or “the likelihood of the parameter.” But keep in mind that likelihoods are not probabilities of models or parameters, $\Pr(M|D)$. Instead, they are probabilities of data, conditioned on models, $\Pr(D|M)$. It is conventional to write $L(M|D)$ or, using a fancy “L,” $\mathcal{L}(M|D)$, but this doesn't change the fact about what the probability really references: observations.²⁷

To use the likelihood formula, you have to assume values of the parameters, like p_W , and then plug in the data. For example, using the sample from earlier in the chapter, if we toss the globe 10 times and observe 6 waters and 4 lands, the formula becomes:

$$\Pr(6|10, p_W) = \frac{10!}{6!4!} p_W^6 (1 - p_W)^4.$$

Now suppose that $p_W = 0.7$, then we arrive at:

$$\Pr(6|10, 0.7) = \frac{10!}{6!4!} 0.7^6 (1 - 0.7)^4 \approx 0.2.$$

Read the left side as “the probability of observing 6 W and 4 L, given that the probability of water is 0.7.” The right side is just the binomial

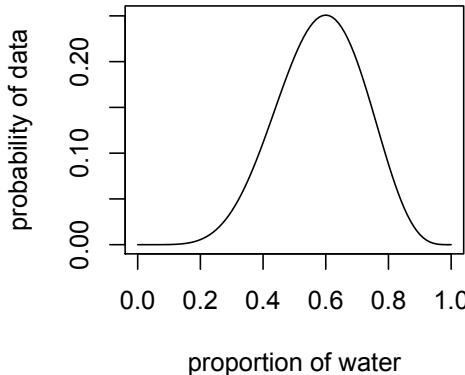


FIGURE 2.2. The probability of observing 6 waters and 4 lands in 10 samples from our globe (vertical axis), for every possible true proportion of water (horizontal axis).

density, with the values plugged in. If you want to evaluate this formula, it is built in to R, so just execute:

R code
2.1 `dbinom(6 , size=10 , prob=0.7)`

[1] 0.2001209

Read that line of code as, “compute the probability of observing 6 W in 10 tosses, where each toss has a probability of 0.7 of being W.”

One of the virtues of a statistical environment like R is that it makes it easy for you to calculate every likelihood over the range of p_W from 0 to 1, holding the data n_W, n constant. This code will plot them all:

R code
2.2 `curve(dbinom(6 , size=10 , prob=x) , from=0 , to=1)`

I reproduce this plot in FIGURE 2.2. The reader really should execute this code in R. Do this now. You don’t have to understand the code yet. But you’ll get a better feel for what we are calculating, if you type the above line of code into R.

What does FIGURE 2.2 tell us? Every possible value of p_W implies a probability of the data, 6 W’s and 4 L’s. The height of the curve at each point is that probability, or likelihood. First, realize that each value of p_W corresponds to a different statistical model. Each model corresponds to a different hypothesis about the true proportion of water covering the planet. Each model can be used as above to help us decide how likely our data are, assuming the model is true. Thus each model produces a likelihood, and these are what are plotted in FIGURE 2.2. If the observed

data are very unlikely, for some value of p_W , then the posterior probability will also be lower for that value of p_W . In this case, as in many cases, there is a unique value of p_W that maximizes the likelihood. This value is $p_W = 0.6$, which is the same as the sample proportion of water, $6/10 = 0.6$. This value of p_W is an example of a *maximum likelihood estimate*. There will be much more to say about such estimates a bit later in this chapter.

Second, these likelihoods do not add up to one, unlike the prior and posterior densities. This is because likelihoods reference the data, which does not change along the horizontal axis in FIGURE 2.2. This means the area under the curve need not and usually will not add up (integrate) to one. This is just like saying that the proportion of women who wear glasses, $\Pr(\text{glasses}|\text{woman})$, and the proportion of men who wear glasses, $\Pr(\text{glasses}|\text{man})$, need not add up to one. Likelihoods vary what the probability is conditioned on. This is unlike the posterior probabilities, which vary what is referenced, what is on the left side of the “pipe” symbol, |.

2.3.2.2. Average likelihood of evidence. The next important thing to do with likelihood is to figure out the probability of the evidence, $\Pr(D)$. This is just the average likelihood, where the averaging is over models. Since a likelihood $\Pr(D|M_{p_W})$ is a probability of the data, if you compute a weighted average of these likelihoods, you get the unconditional probability of the data, $\Pr(D)$. This probability is still conditional on the models, in the sense that it is averaged only over those models you explicitly include in the analysis. So assumptions about the world still intrude here, as they always will. But $\Pr(D)$ is not conditional on a single model, like a likelihood $\Pr(D|M_{p_W})$ is.

So $\Pr(D)$ is a weighted average likelihood. But where do we get the weights of each model, needed to compute this weighted average? From the prior probability density. The probability of the evidence is given by:

$$\Pr(D) = \int \Pr(D|M_{p_W}) \Pr(M_{p_W}) dp_W.$$

All this really says is to multiply each likelihood by its corresponding prior and then add up all of these products. This results in a weighted average likelihood. This sort of probability is often called a *marginal likelihood*, and we'll meet it again in Chapter 5 and explain why at that point.

More important for now is to appreciate the job it does. The role of this probability of evidence inside the conditioning engine, Bayes' theorem, is to normalize the posterior, so that it always sums to one. This ensures that the posterior is a coherent probability density, just like the prior. The relative magnitudes of the different models under consideration will not change, whatever value you assign to $\Pr(D)$. This is because a posterior probability is proportional to the product of the likelihood and the prior:

$$\Pr(M_{p_W}|D) \propto \Pr(D|M_{p_W}) \Pr(M_{p_W}).$$

In many cases, these products are all you actually need, because given these products, you can still compare the relative posterior probabilities of the models, even though you can't say what the precise probability is. If you add up all of these products and divide each product by that sum, you standardize the ensemble of products so that they comprise a probability density, a proper posterior.

We set out here to talk about likelihood, but somehow we ended up again at priors. The prior directly influences the posterior by being the original distribution that is modified inside the conditioning engine, Bayes' theorem. As a consequence, the prior also determines which models have stronger weight in the calculation of $\Pr(D)$.

2.3.3. Running the Conditioning Engine. In order to use Bayes' theorem, to run the conditioning engine and convert a prior to a posterior, you have to have an algorithm for performing the computations. You have four basic choices. In this book, I'll teach you three of them.

- (1) *Analytical.* For some kinds of likelihood functions and prior densities, integral calculus can do the job. But for most scientists, this isn't a reasonable course to pursue. Not only is the analytical approach hard for most natural and social scientists, but it just doesn't work for many important models. So instead we need some kind of numerical approach. Indeed, the modern surge in Bayesian statistics is largely a result of escaping the necessity to use analytical approaches.
- (2) *Grid approximation.* A very capable numerical approach is to divide up the continuous parameter space into a finely-spaced grid of discrete values. Then it is a simple matter of iterating over these values to compute the posterior probability of each. This approach works for any prior and likelihood, not just conjugate pairs. It is an approximation, because it uses a discrete space instead of a continuous one. But the approximation can be very good, provided the density of the discrete values is high

in the region of interest. The problem with grid approximation is that it is very computer intensive. Once you begin considering model families with even a few different dimensions—kinds of parameters—grid approximation becomes impractical.

- (3) *Maximum likelihood and quadratic approximation.* The vast majority of applied statistical analyses published in scientific journals provide a posterior estimate by seeking a maximum likelihood estimate (MLE) and then using a simple approximation of the posterior density, the *standard error*. Together, these two numbers, the MLE and standard error, give an approximation of the posterior that is good provided the sample is large enough and the estimate is far enough from any boundary. Later in this chapter, I'll explain what these conditions mean in more detail.
- (4) *Markov Chain Monte Carlo.* Finally, it is possible to estimate complex posterior densities for high-dimension models using Markov Chain Monte Carlo (MCMC) techniques. These algorithms also provide only approximations, but potentially rather good ones. Much later in the book, I'll explain how some of these algorithms work. They are actually very simple. But like many simple and brilliant ideas, they are also very counter intuitive. MCMC is the modern workhorse of Bayesian model estimation, because there are many important models that cannot be easily estimated any other way.

In the remainder of this chapter, I introduce you to grid approximation and the maximum likelihood approximation. You aren't going to be using grid approximation much, either in this book or in your research. It's usually just too computer intensive to be useful. But grid approximation of the posterior density will serve an important pedagogic purpose, because when you learn grid approximation, you also really come to understand every piece of Bayes' theorem. This serves to further demystify it.

You are going to be using maximum likelihood estimation a lot. It provides an extremely fast and often accurate approximation of the posterior. Furthermore, since scientific journals are full of mostly maximum likelihood estimates and standard errors, being able to provide a Bayesian interpretation of those numbers will allow you to get much more out of the scientific literature.

2.4. Grid Approximation

The posterior probability of a model is merely the standardized product of that model's prior probability and likelihood. So to compute the posterior, all you have to do is:

- (1) For each model, multiply the prior by the likelihood.
- (2) Add up all of the products from (1).
- (3) Divide each product from (1) by the sum from (2).

When the models to consider are actually discretely different parameter values, then the above procedure isn't an approximation at all. Instead, it's the real thing. The trouble arises when parameter spaces are continuous, such that there is essentially an infinite number of models to consider. Infinities are easy to handle in mathematics, but hard to handle in the computer.

So the purpose of a *grid approximation* is the divide up the continuous parameter space into a large number of discrete models. Then the above algorithm will provide an approximate posterior density for the models. If the models are very close to one another, then the approximation can be very accurate.

I'm going to provide a simple block of code for computing a posterior density via grid approximation. I'll dump out the code in one chunk here, but then I'll explain each line of code, one at a time. Here's the working code to compute the posterior density for the proportion of water, p_W :

R code
2.3

```
models <- seq( from=0 , to=1 , length.out=1000 )
prior <- rep( 1 , 1000 )
likelihood <- dbinom( 6 , size=10 , prob=models )
posterior <- likelihood * prior
posterior <- posterior / sum(posterior)
```

The first line of code above divides up the grid. You have to decide how to cut up the parameter space. In the case at hand, the problem of computing the posterior density for the proportion of water p_W , this implies picking how far apart the unique values of p_W will be. The easiest way to make a grid of values of p_W is to use the `seq` function to divide up the interval from $p_W = 0$ to $p_W = 1$ in any number of units. Here is how you divide it up into 1000 units, which makes a grid in which each value of p_W you'll consider is 1/1000th apart from one below it and one above it. The symbol `models` now contains one-thousand parameter values for p_W . That is one-thousand models to compute posterior probabilities

for. The ensemble of all one-thousand will be the approximate posterior density.

The next step is to prepare the prior probabilities of each of these models. This is what the second line of code does. You can use absolutely any prior here. Technically, all you have to ensure is that all of the prior probabilities sum to one. But even that can be ignored in practice, because you are going to standardize the posterior later. It's only the relative values that are going to matter in this calculation. Let's make a boring flat prior for the moment. A flat prior just assigns equal prior probability to every model. This can be done here by merely repeating the same value one-thousand times, and that's what the `rep` function does.

Now for the likelihoods, on the third line of code. You already know how to compute these. Again, assume the observed data are 6 waters and 4 lands. Then the binomial density provides the likelihoods, one for each model in `models`.

Next, Bayes' theorem appears. Multiply each likelihood by its corresponding prior. R makes this trivial, because if you multiply two vectors (lists of numbers) together in R, the computer automatically multiplies the corresponding numbers in the each vector. So you end up with a vector of the same length as `prior` and `likelihood`, but containing the products for all one-thousand models.

The last step is to *standardize* the posterior, which is the job of that marginal likelihood $\Pr(D)$ in the denominator of Bayes' theorem. $\Pr(D)$ is just the sum of all of the products in the symbol `posterior` above. So to standardize and make a proper probability density, all that remains is to divide each element of `posterior` by the sum of all elements. The last line of code does this.

The first row in FIGURE 2.3 shows the prior, likelihood and posterior you just computed. Because the prior density is flat, it has no effect on the likelihood. This fact will be important to us later in this chapter. What is important to grasp for now is that the posterior is the product of both the prior and the likelihood. That's all that Bayes' theorem really says, aside from the additional standardization provided by $\Pr(D)$, to ensure that the posterior sums to one. So to complete a grid approximation, just compute all the products of the priors and likelihoods and then standardize.

The other two rows in FIGURE 2.3 complete the grid approximation, but using different prior densities. On the second row, instead of making every parameter value equally probable, suppose instead that you are willing to bet that no more than 75% of the planet is covered in water.

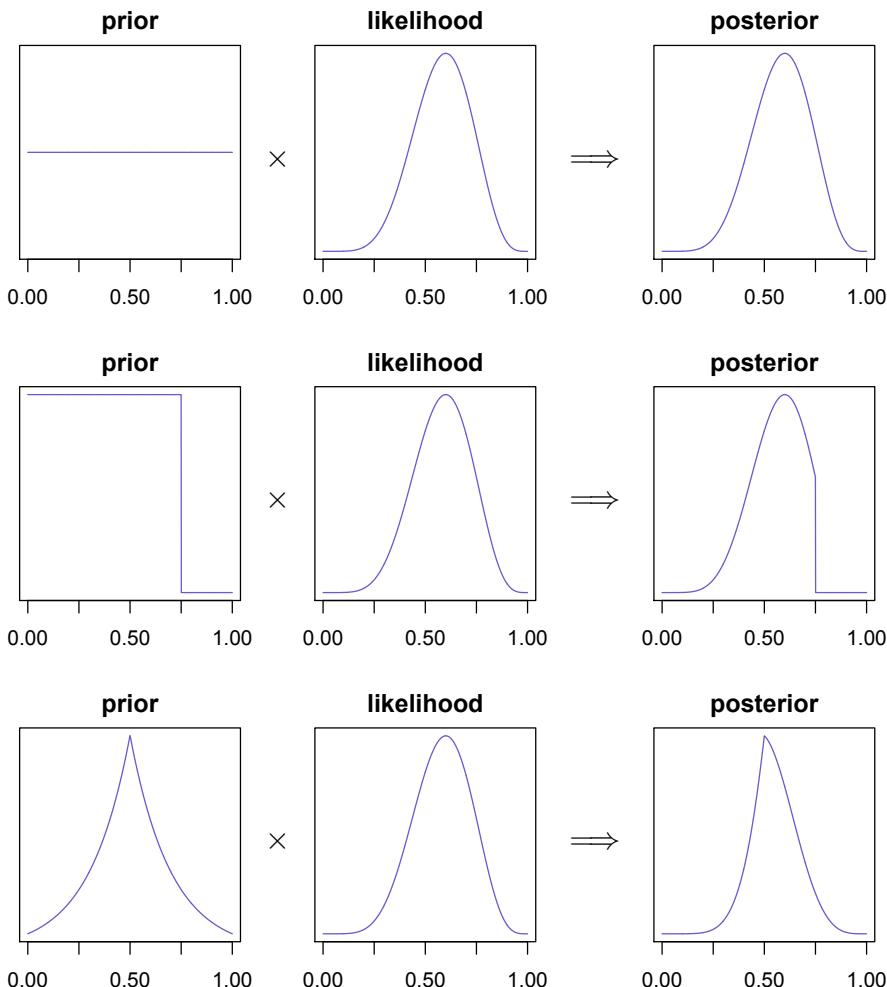


FIGURE 2.3. Posterior densities produced by grid approximation. In each row, a specific prior is combined with the likelihoods of observing 6 waters and 4 lands to produce a posterior density across values of p_W . Top row: A flat prior has no effect on the posterior. Middle row: A truncated prior that assigns probability zero to all models with $p_W > 0.75$. Bottom row: A peaked prior leads to a peaked posterior.

This implies assigning a prior probability of zero (or something very close to zero) to all models with $p_W > 0.75$. Maybe the most transparent way

to construct this prior in the R code is to just modify the previous prior, using a conditional subset:

```
prior <- rep( 1 , 1000 )
prior[ models > 0.75 ] <- 0
```

R code
2.4

All the second line of code does is replace with zero every entry in `prior` that corresponds to an entry in `models` with a value greater than 0.75. The posterior density on the second row of FIGURE 2.3 that results from this prior is essentially lopped off for values of $p_W > 0.75$. This kind of prior is rare in applied statistics, but I have seen it put to good use in archaeology. Sometimes, we know from previous research that an artifact cannot be younger or older than a certain date. In that case, it makes sense to use a prior that observes such a constraint. The prior doesn't have to be all-or-nothing like this one. But if you aren't going to seriously consider some range of models, then you can use the prior to exclude them.

The third row in FIGURE 2.3 shows a prior density peaked at $p_W = 0.5$ and declining rapidly in both directions. This prior is a double exponential (also known as a Laplace prior):

```
prior <- exp( -5*abs(models-0.5) )
```

R code
2.5

Again, multiplying the prior and the same likelihood again produces a different posterior. This time, the resulting posterior density is peaked at $p_W = 0.5$, because the prior was peaked there. There is only modest evidence in this case, so the likelihood did not overwhelm the prior. But if there were more data, even this prior density wouldn't matter much. And that's where we turn next, on the road to understanding the most-used method of estimating posterior densities (although usually unconsciously), maximum likelihood estimation.

2.5. Justifying Maximum Likelihood

Bayes' theorem provides a principled way to relate the evidence to our models. But you probably already know that most of applied statistical inference ignores Bayes' theorem in favor of either (1) P-values or (2) maximizing likelihood. Often these two methods are blended in illogical ways. Neither of these approaches typically mentions Bayes' theorem or posterior probability. How do these “non-Bayesian” approaches relate to the posterior probabilities that provide relative measures of confidence in models?

As you'll see in the next chapter, the use of P -values in the testing of null hypotheses—the Tyranny of Fisher—cannot be easily justified. Still, it can be understood as a kind of Bayesian analysis that prefers a null model. Maximum likelihood estimation has an even stronger connection to the posterior density. Indeed, Gauss and Laplace essentially used maximum likelihood inference as part of a Bayesian analysis, a hundred years before Ronald Fisher.²⁸

Maximum likelihood estimation is the strategy of finding which model M maximizes $\Pr(D|M)$. Maximum likelihood estimates have a number of excellent features.²⁹ It's good features rely upon the fact that the posterior is proportional to the likelihood:

$$\Pr(M|D) \propto \Pr(D|M).$$

But maximum likelihood suffers from the handicap that the posterior is equally proportional to the prior probability, $\Pr(M)$. But under some conditions, it turns out that the likelihood dominates and tells us most of what we need to know about the posterior. These conditions provide Bayesian justifications for comparing models using only likelihoods. Although as you'll see, everyone agrees that we need to know more than just the maximum likelihood estimate itself. Familiar statistics like standard errors are descriptions of the shape of the posterior density in the neighborhood of the maximum likelihood estimate.

What are the conditions that provide some Bayesian cover for likelihood-only inference? There are at least three pragmatic ways to justify using likelihood alone to make inferences about which models to have confidence in.

- (1) You can lean on the fact that, as the amount of data increases, the likelihood will dominate the right side of Bayes' theorem, making the prior unimportant. In such a case, maximizing the likelihood will also maximize the posterior.
- (2) You can adopt the perspective that a fair objective prior is an *uninformative prior* that places equal or nearly-equal probability on every model. In that case, the prior has little to no effect on the posterior, whatever the sample size. Again, maximizing the likelihood will maximize the posterior.
- (3) You can just accept the limitations and risks of ignoring the prior and use likelihood-only inference as a way to summarize what only the evidence at hand says about the posterior.

By adopting any of these positions, you have to give up some of the utility that prior probabilities can provide. But what you gain is a useful way to approximate the most important portions of the posterior distribution,

provided you are comfortable with the assumptions you impose in justifying the approximation. We'll look at each of these justifications in a little detail, before moving on to actually doing maximum likelihood estimation.

2.5.1. Bury the prior in data. The first approach is to observe that the influence of the prior diminishes as the size of the sample increases. Why? Because as the sample size increases, the likelihood function—look again at the curve in FIGURE 2.2—gets narrower, piling up support for the maximum likelihood estimate. Eventually, the likelihoods for models even a small distance from the maximum likelihood estimate become very close to zero. Then, even if the prior probability of a model is large, when it is multiplied by a value close to zero, the posterior probability of the model is very small. Of course, this process works in reverse, too. If the prior probability of a model is very close to zero, it will take a ton of data supporting it, before the posterior probability will be large. However, eventually one can collect enough evidence to overwhelm any prior probability, as long as it isn't exactly zero (0).

FIGURE 2.4 will help you intuit the relationship between priors, likelihoods, and posterior probability. Focus on only the top-left plot for now. The horizontal axis is the set of models, the range of possible proportions of water. The vertical axis is probability. Don't worry about the absolute scale of the probability axis. Instead focus on relative probability. The data in this example is a sample of 10 tosses of the globe, resulting in 6 water and 4 land. Now consider the three curves, representing the prior (dashed), the likelihood (blue), and the posterior (black).

The prior, shown with the dashed curve in FIGURE 2.4, must specify a probability for each model. For the sake of illustration here, I've chosen a prior heavily weighted towards models with small proportions of water. Models with proportions above 0.5 in fact have prior probability of nearly zero. Why use a prior like this? Honestly, I chose it because it is so different from the likelihood. But one might justify such a prior by appealing to other evidence or to mere intuition. We'll address the sources of priors in more depth in later parts of the book.

The likelihood function, shown with the blue curve and measured on the blue righthand axis, states the probability of observing 6 water and 4 land, for each model on the horizontal axis. It peaks at the sample proportion, 0.6, the maximum likelihood estimate. Note however that is quite wide. With only 10 tosses of the globe, likelihood falls only slowly as models increase or decrease the true proportion of water on

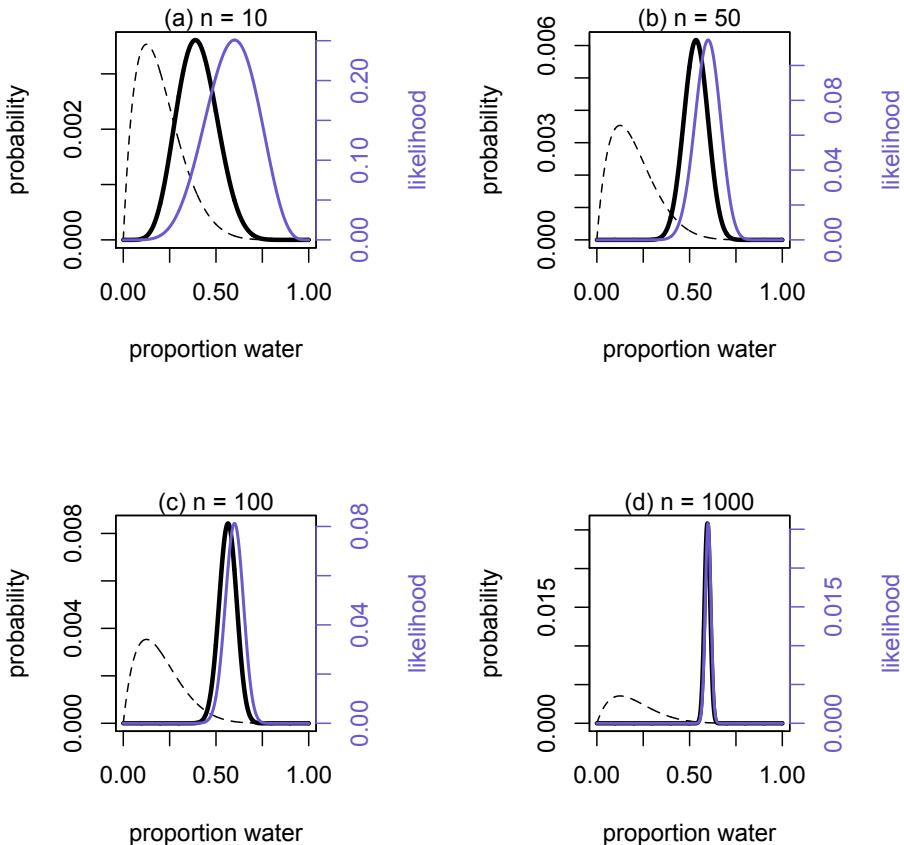


FIGURE 2.4. How to bury your prior in data. In each panel, the three curves represent the prior (dashed), the likelihood (blue), and the posterior (black). Each panel shows the posterior for different amounts of data, n , from 10 to 1000 tosses of the globe. In each case, the proportion of tosses recorded as water remains constant at 0.6. The prior is the same in each case, chosen to be heavily weighted towards small values. In the upper-left, $n = 10$ is a small sample, and the likelihood curve is quite wide. As a result, the posterior ends up being a compromise between the prior and the likelihood, laying about half way between their peaks. In the upper-right, $n = 50$, and the additional evidence narrows the likelihood curve and pulls the posterior much closer to 0.6, the maximum likelihood estimate. In the bottom row, at $n = 100$ and $n = 1000$, there is so much evidence now that the prior is essentially overwhelmed, having negligible influence on the posterior.

the globe. As a consequence, there are many model near $p_W = 0.6$ with high likelihood.

Keep focusing on only the upper-left plot in FIGURE 2.4. The posterior probability, shown by the solid black curve, lies about halfway between the prior and the likelihood, in this case. The weight the prior places on models with low p_W and the weight the likelihood places on models with $p_W \approx 0.6$ results in an intermediate peak for the posterior, near $p_W = 0.4$. Note that the posterior probability at the maximum likelihood estimate, $p_W = 0.6$, is not very high. This is the influence of the prior.

Now consider the other three plots in FIGURE 2.4. In the upper-right, the sample size has increased to 50 tosses of the globe, but the proportion of the sample that came up water has remained the same. Therefore the maximum likelihood estimate is unchanged, still lying at $p_W = 0.6$. Now there is 5 times as much evidence in favor of the maximum likelihood estimate, however. Likelihood declines more rapidly as models move away from $p_W = 0.6$. The likelihood function is narrower than it was when the sample size was only 10. And so the posterior now peaks much closer to $p_W = 0.6$ than before. The same phenomenon is magnified in the final two plots, in the bottom row of FIGURE 2.4. On the bottom-left, the sample is one-hundred tosses of the globe, with 60 coming up water. On the bottom-right, the sample is one-thousand tosses, with 600 water. As sample size increases, the likelihood function peaks more sharply at the maximum likelihood estimate and drags the posterior closer and closer to it. In the bottom-right plot where the sample is one-thousand tosses, the posterior and likelihood are so similar that they are almost exactly the same.

So we see here one way to justify using the likelihood alone to compare models and favor the maximum likelihood estimate. Provided sample size is large enough, and the prior is not strong enough for a given sample size, the prior has little influence on the posterior. In such cases, the likelihood tells us almost everything there is to know about the posterior. One caveat is that, in order to verify that we have enough data to overwhelm the prior, we have to provide a prior and compute the posterior. There is no magic sample size at which, for all models and all kinds of data, the prior suddenly ceases to affect the posterior. Perhaps more important, there are many kinds of models, such as hierarchical regressions, in which the data and parameters are structured in ways that may make it impossible for increasing sample size to swamp the influence of priors. The basic reason is that as sample size approaches infinity, so

too does the number of parameters. We'll encounter examples of such models in later chapters.

You should play around with more examples of how the sample size and the shapes of the prior and the likelihood interact in producing the posterior. In the exercises at the end of this chapter, I provide the code to produce the plots in FIGURE 2.4, along with some instructive experiments to try.

2.5.2. Uninformative prior. The second approach is to use a prior that is uniform or nearly uniform, assigning the same probability to every model. Such priors are often called *weak* or *uniform* or *vague* or simply *uninformative*. Each of these terms can mean slightly different things, so it's easy to become a little confused.³⁰ The intent is usually the same, however: to choose a prior probability density that does not substantially affect the posterior, if it affects it at all. In other words, these priors are designed to let the likelihood function largely determine the posterior.

For example, suppose $\Pr(M)$ is approximately constant across models. Now Bayes' theorem tells us that the posterior is equal to the likelihood multiplied by a scaling constant:

$$\Pr(M|D) = \Pr(D|M) \frac{\Pr(M)}{\Pr(D)} = \Pr(D|M)K.$$

(Since likelihoods across models don't have to sum to unity, unlike the posterior, the scaling constant K just scales the likelihood function so that it sums to one.) Therefore if we find the model that maximizes the likelihood, we have simultaneously maximized the posterior. If we describe the proportions of the likelihood function, then we also describe the proportions of the posterior.

A large proportion of Bayesian statistical analyses actually use this kind of prior. And the defaults of Bayesian modeling software are nearly always largely uninformative. When we get into serious Bayesian estimation in a much later chapter, we'll have to specify these priors, and then you'll see what these defaults are.

Uninformative priors are not always perfectly uninformative, though. A common approach is to use a Gaussian prior, or some other parametric distribution, with a very large variance. In FIGURE 2.5, I illustrate the effect of increasing the variance (think "width") of a prior distribution. In all three panels of Figure 2.5, the data are again 10 tosses of the globe, counting 6 water and 4 land. The likelihood function, shown by the gray curve, therefore is identical in all three plots. However, the priors, shown by the dotted curve, are different in all three plots. These different priors result in different posterior probabilities, shown

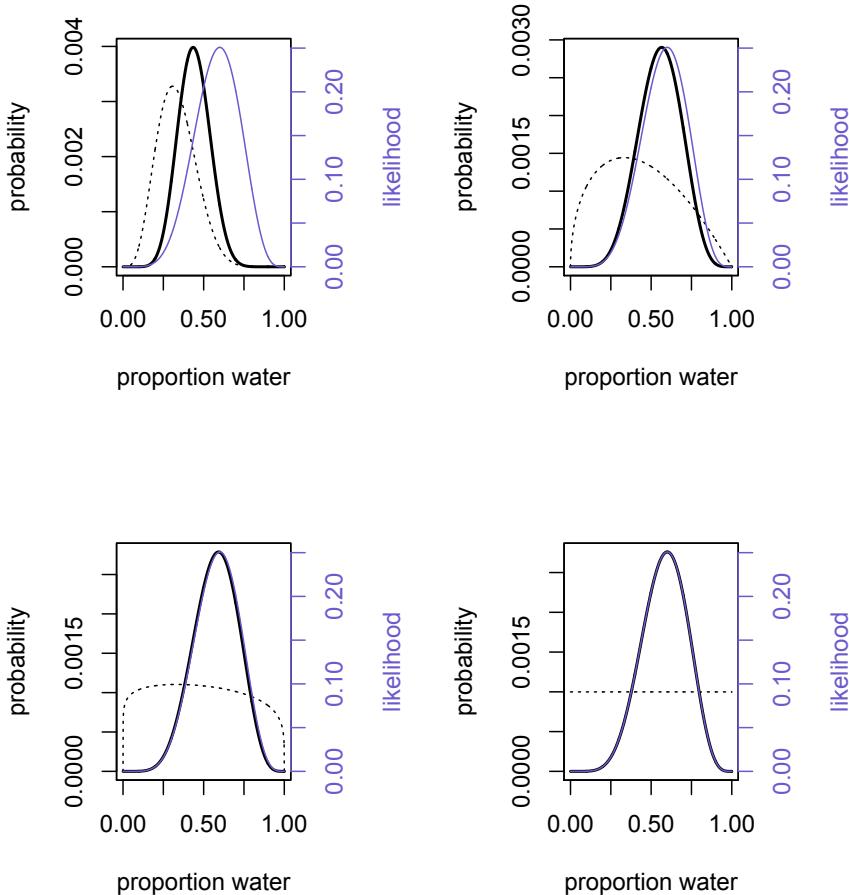


FIGURE 2.5. How to make a prior uninformative. The dotted, blue, and black curves are again the prior, likelihood, and posterior, respectively. Each plot above shows these three curves, for identical data ($n = 10$, 6 water and 4 land), but with different priors. As the variance of the prior increases from top-left to bottom-right, the prior becomes flatter and flatter, and as a consequence the posterior becomes more and more similar to the likelihood, until the posterior and the likelihood are identical in their proportions.

by the solid curves. In the first plot on the left, the prior strongly favors the hypothesis that the proportion of water on the Earth's surface is 0.5. Probability quickly declines on both sides of this hypothesis. As a result, the posterior is tugged close to the prior. In the middle plot, the center of the prior remains at 0.5, but the variance of the prior has increased. Now probability declines much more slowly on both sides of 0.5. As a result, the posterior is more strongly influenced by the likelihood and lies closer to it. Finally, on the right, the variance of the prior is much larger. The prior assigns the same probability to every hypothesis. Now the posterior is identical to the likelihood (at least in its proportions), as the analysis of Bayes' theorem above suggested. A prior with very large, but finite, variance would produce nearly the same effect.

We've really only scratched the surface of this topic. There are situations in which choosing a prior that assigns equal probability to all models seems reasonable, by objective standards. To take a famous example, suppose we know that a ball is hidden under one of three shells. The prior that assigns probability $1/3$ to each shell is not just uniform, it is objective: we have no information that allows us to differentiate the shells, and the prior should reflect this fact. When we meet information theory in Chapter 5, you'll see how to put the claim that equal probabilities are *uninformative* on a formal footing.

But it is easy to come up with other examples in which assigning equal probability to all models seems bizarre. For example, suppose you wish to assign posterior probabilities to models that differ in ways other than just their parameter values. A common problem is that there are many possible variables one might use in a model. Now, should the prior probability of a multiple regression with 27 predictor variables be the same as the prior probability of a multiple regression with 2 predictor variables? I think there is a principled way to address question, and in Chapter 5 you'll see that this way gives the answer, "no."

But in general, there is yet no consensus in statistics on how to answer such questions. But this just makes assumptions about prior probability like assumptions in other parts of science: You have be prepared to adjust your assumptions for each problem and be prepared to defend your decisions. The assumption that each toss of the globe is independent of the others is an assumption, and it too must be defended. So it isn't the case that only prior probability forces difficulties in defending assumptions. The point to appreciate instead is that there are objective arguments that lead to these assumptions. Likewise, there are ways to check these assumptions, but they might call for thinking in the large world.

2.5.3. Focus on the evidence. The third approach to relying exclusively upon the likelihood is to admit that a prior is necessary to compute the posterior precisely, but nevertheless focus on what the current data tells us about the posterior, by focusing on the likelihood. Since the prior does not depend upon the current data, but the likelihood does, we can learn about the how the evidence reflects on models by using the likelihood.

In what sense is this reasonable? Whatever the prior, you can still calculate and describe the change in odds of one model over another, due to the evidence (the likelihood). This holds the value of describing in probability terms what the evidence says. It's not an argument that this is all that is relevant, but it does address important questions about how the evidence changes prior probability. Then, even if different scientists cannot agree upon what is a sensible or useful prior in a circumstance, they can still agree on what the experiment or study would do to *any* prior.

Make no mistake: This justification for using maximum likelihood is contentious. It invites a number of objections. Most readers are likely to be pragmatic types who are motivated more by practical concerns than by epistemological twists. So instead of philosophy, consider how you would feel about the likelihood function alone, if you tossed your globe into the air 10 times and counted water all 10 times. The model that yields the greatest likelihood in this case is obviously $p_W = 1$, a world covered entirely in water. But you would never believe the inference implied by the likelihood function, in this case. You know, before tossing the globe once, that $p_W = 1$ as well as $p_W = 0$ are not credible hypotheses. In later chapters, we'll return exactly to this point and see how to use priors to explicitly incorporate this kind of knowledge.

However it is hardly essential that we always build such information into the prior. It is important to remind ourselves that a late step of any statistical analysis is to check the sanity of the inferences, using information from outside the analysis. In the example of observing 10 waters in 10 tosses of the globe, the quantitative evidence that went into our calculation says we should take seriously the belief that the world is covered entirely by water. But you are no fool. Unless you are reading this book on a boat or in an airplane, you are standing on dry land. For more serious scientific problems, it will not always be straightforward to check such inferences against other information. However, single statistical analyses rarely produce consensus. Our colleagues will pick apart our conclusions, if they are doing their job. Similarly, while I will have little good to say about P -values in Chapter 3, even I agree that

checking the predictions of a model against actual observations has a very important role to play in validating statistical models. But a good way to do this will depend upon the context, the goals, and upon having a good model in the first place, helped along by posterior probability inferences.³¹

Furthermore, statistics is not a substitute for science. This point seems obvious, once you say it, but every time I teach against P -values (Chapter 3), someone objects that P -values must work, because otherwise science would not have produced reliable knowledge in recent decades. My response is always to point out that science worked quite well long before Fisherian P -values became the dominant ritual. I also challenge students to name one famous scientific discovery of the 20th century that hinged upon a P -value. I have yet to hear of one. The mistake here is in identifying statistical procedures with science in total. We should strive to do the best statistical analysis we can, and often that will include adopting a variety models and priors and checking model predictions against data. Much of this is an attempt to simulate the scientific processes of replication and validation of models. But I do not think these formal procedures can replace the informal and unpredictable conversations by which the scientific community reaches and replaces consensus views. Seen in this light, calculations based on naked likelihood are part of this conversation. Often we could do better. But as long as we are wary of their incompleteness, so that we can recognize and diagnose pathological behavior, naked likelihood is a useful Bayesian tool.

2.6. The Bayesian Method of Maximum Likelihood

Provided you are willing to adopt one or more of the positions above, then the comparison of likelihoods is underlain by the logic of the comparison of posterior probabilities. It makes sense to prefer models with larger posterior probabilities, and so if we are betting on the likelihood containing the information we care about, then it makes sense to prefer models with larger likelihoods. I like to call a posterior probability density determined almost entirely by likelihood a *naive posterior*. This posterior is *naive* only in the sense that it is uninfluenced by prior information.³² And that's a term I'll use repeatedly in this book. The naiveté of such a posterior is not a bad thing. It's just honest.

We should keep in mind, however, that we have strategically ignored or tried to ignore prior probability. Later in this book, you'll see that thinking hard about priors can do very useful things for us. Even then, however, the concern will be with acquiring a good estimate of the naive

posterior density. This isn't because I think this is the only useful thing to do, but rather because this approach is very common—even Gibbs Sampling (Chapter 11), an inherently Bayesian approach to model estimation at the core of powerful packages like BUGS and JAGS, is almost always aimed by the user at the naive posterior. So in order to understand the literature and how models fit with such engines relate to typical maximum likelihood or Least Squares (OLS) estimation, it will help to talk about the naive posterior and how maximum likelihood provides an estimate of it. All of these approaches aim at the same sort of estimates, but approaches like Gibbs Sampling are just much much more capable than is maximum likelihood. But Gibbs Sampling is also much much harder to use and consumes much much more computer time. So the goal will not be to eventually abandon maximum likelihood estimation of naive posteriors. Maximum likelihood will remain useful for a very long time, even in a Bayesian society.

In common practice, and as we will employ it for much of the first parts of this book, using maximum likelihood means searching for the unique model (combination of parameter values) that maximizes the likelihood. This model is the *maximum likelihood estimate* (MLE). This turns statistical inference into an optimization problem, for which there are many powerful and efficient methods. The remainder of this section teaches the reader how to do this for simple one-parameter problems like our proportion of water. But the shape of the naive posterior matters as well, and so in the later sections of this chapter, we will see how to use the likelihood to calculate *confidence intervals*, which help us describe the shape of naive posterior. These estimates are essential, because only knowing the maximum likelihood model cannot distinguish between wide and narrow naive posteriors, like those in FIGURE 2.1 (page 35).

So how does maximum likelihood work, in practice? The primary objective of maximum likelihood search is to find the model that maximizes likelihood. To do this, you need to know the formula for the likelihood, or be able to approximate the likelihood. We already know the formula for the likelihood, in the case of our water on the globe example. It is binomial, the “coin flipping” likelihood. The models in this context differ only in a unique value of the adjustable, non-data, part of the model, p_W , the assumed true proportion of the globe that is covered in water. The models have other assumptions, and each could be criticized, but comparison of naive posterior probabilities (likelihoods) is not going to shed much light on them.

Once you have the ability to compute likelihood for all models, there are two ways to go about finding the unique model that maximizes likelihood. First, if the problem at hand is simple enough, you can just optimize the likelihood analytically, as an exercise in elementary calculus. Second, in many of cases, the likelihood formula will be too complex to analyze mathematically, and so we fall back on numerical search—trying a bunch of different models and exploring the space of models, until we find a good candidate for the maximum likelihood.

At this point, some readers will want to skip ahead. If you've had a good education in Fisherian statistics, then you know how maximum likelihood works, at least in the abstract. However, the sections to follow begin to include a substantial amount of R code for actually doing likelihood searches. Importantly, I use these sections to introduce a family of general maximum likelihood commands that the book will use again and again to fit many different sorts of models. These commands are not the easiest way to fit these models, but they do provide a unified and transparent approach that ultimately is much more powerful and illuminating than the menagerie of specialty commands like `lm`, `glm`, `polr`, etc. So I recommend at least skimming through and stopping at the code boxes to execute and meditate on the practical difficulties of actually making your computer do this stuff. The power of an approach is often inversely proportional to how hard it is to learn. So when some approach is easier to learn, you are usually giving up power by using it. After several chapters of fitting models with generalized maximum likelihood commands, you will be grateful I forced you to do it this way, because you will understand the models at a deeper level, and you will be able to fit many models for which there are no built-in convenience commands like `lm`.

2.6.1. Analytical solution. In this case, our maximum likelihood problem is simple enough to be done analytically. For the sake of demystifying the process, here's how you can derive mathematical expressions for some maximum likelihood estimates. The general approach is just to maximize the likelihood, so this is an exercise in basic calculus. If calculus scares you, then skip ahead the next subsection, to where we do this with R code. It is worth knowing the outline of this argument, however, as the method of least-squares estimation commonly used to fit linear regression models (Chapter 4) usually relies upon this kind of analysis for its justification.

We already know the likelihood function, which we aim to maximize:

$$L(p_W|n_W, n) = \frac{n!}{n_W!(n - n_W)!} p^{n_W} (1 - p)^{n - n_W},$$

where p_W is the model, the proportion of the Earth covered in water, n_W is the observed count of water points, and n is the total number of points sampled. Next, we are going to take the natural log of both sides. It is almost always easier to work with probability, if we work in (natural) logarithms. This is especially true when there are derivatives involved. The basic reason for this ease is that multiplication on the probability scale is equivalent to addition on the log-probability scale. Taking the derivative of a sum is easier than taking the derivative of a product, as you might recall from your tedious calculus homework exercises. It is also true that taking the derivative of a logarithm of a function is rather easy: $d \log F(x)/dx = F'(x)/F(x)$. Here's the logarithm of $L(p_W|n_W, n)$:

$$\begin{aligned} \log L(p_W|n_W, n) &= \\ &\log\left(\frac{n!}{n_W!(n - n_W)!}\right) + n_W \log(p_W) + (n - n_W) \log(1 - p_W). \end{aligned}$$

To maximize this as a function of p_W , we take the derivative with respect to p_W :

$$\frac{\partial}{\partial p_W} \log L(p_W|n_W, n) = \frac{n_W}{p_W} - \frac{n - n_W}{1 - p_W} = \frac{n_W - np_W}{p_W(1 - p_W)}.$$

Setting this equal to zero and solving for p_W gives us a formula for the maximum likelihood estimate, let's call it \hat{p}_W :

$$\begin{aligned} \frac{n_W - n\hat{p}_W}{\hat{p}_W(1 - \hat{p}_W)} &= 0, \\ \hat{p}_W &= \frac{n_W}{n}, \end{aligned}$$

which is just the proportion of the sample that is water.

2.6.2. Numerical solution. When it is impossible—or one just doesn't know how—to use the analytical solution, it is still possible to search numerically for the maximum likelihood estimate. The beauty of numerical search is that it can work for any problem, provided you can supply a likelihood for each model. The problem with numerical search is that there is no guarantee that it can find the maximum likelihood estimate. Let's see how to search for and find the maximum likelihood

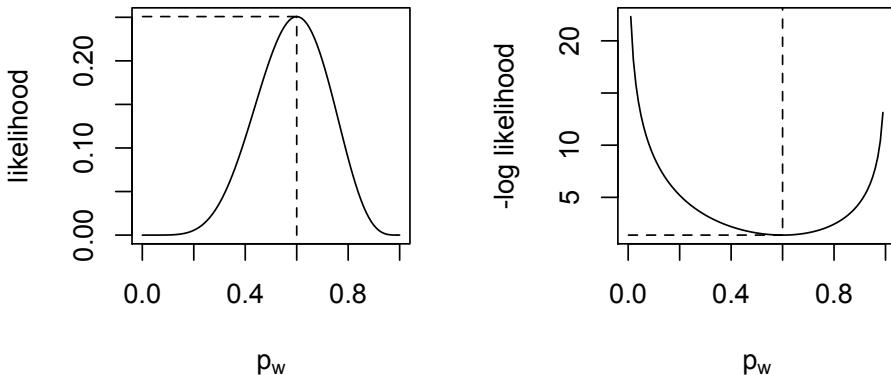


FIGURE 2.6. Minimizing the negative logarithm of the likelihood is equivalent to maximizing the likelihood. On the left, the likelihood function, with a distinct maximum at $p_w = 0.6$. On the right, the same function after taking the natural log and multiplying by -1 . Now there is a distinct minimum, still at $p_w = 0.6$.

estimate, in this case. Then we'll return to the issue of why numerical search cannot provide a guarantee.

2.6.2.1. The importance of log-likelihood. The first lesson about searching numerically for maximum likelihood estimates is that it is almost always more accurate and more convenient to search for the *minimum negative log-likelihood* instead of the maximum likelihood. The negative log-likelihood, $-\log L$, is just the natural logarithm of the likelihood, multiplied by negative one (-1). This change of scale does not change the identity of the model that maximizes the likelihood. If $p_w = 0.6$ maximizes the likelihood, then it will also minimize the negative log-likelihood. The important insight, as shown in Figure 2.6, is that where the likelihood curve has a peak, the negative log-likelihood curve has a nadir. We can go back and forth between likelihood and negative log-likelihood whenever we want, provided we keep in mind whether we want to maximize (likelihood) or minimize (negative log-likelihood). To get back to likelihood, just multiply the negative log-likelihood by -1 and then exponentiate it: $\exp(-(-\log L)) = L$.

So why use log-likelihood rather than raw likelihood? Isn't this just the kind of game mathematicians play on scientists, to make matters more confusing than they should be? Sadly, no. I admit mathematicians do sometimes make matters more confusing than they should or need to be, although rarely is this intentional. The key practical reason for using negative log-likelihood is that computer software handles small decimal values poorly. In even reasonably large sets of data, even the maximum likelihood will be a very small number, close to zero. This is because the likelihood is the joint likelihood of all the data. This means the likelihood is the product of many individual probabilities, each less than one. As such a product gets close to zero, most computer software, including R, will round it to exactly zero. On the $-\log$ -likelihood scale, however, this value becomes a very large positive number. As a result, the computer won't feel inclined to round it. You can verify this for yourself by using the three lines of R code below:

```
p <- runif(1000)
-log( prod( p ) )
sum( -log(p) )
```

R code
2.6

```
[1] Inf
[1] 987.5311
```

This code first generates a list of one-thousand random numbers between zero and one. Think of these as probabilities of individual data points. The second and third lines then perform the same calculation, computing the negative logarithm of the product of the probabilities p , but with the steps in a different order. The second line computes the product of all one-thousand probabilities. This is like a joint likelihood for the whole set of data. Then it takes the log of this single value and finally multiplies it by -1 . The third line computes the same quantity, but on the log scale, by first taking the log of each element of the list p , multiplying each by -1 , and then adding all of these log-probabilities together. Remember, multiplication on the probability scale is addition on the log-probability scale. The direct multiplication route produced the answer `Inf`, which means infinity, ∞ . This can't be correct. The only probability with a $-\log$ of infinity is zero, $-\log(0) = \infty$. But because there are no zeros in the list p , something has gone wrong. The only way for a product to equal zero is if at least one factor is equal to zero. What has happened here is that the product shrank and shrank with each step of multiplication, until R ran out of *precision* and rounded the whole product to zero. The log-likelihood summation route did not end

up rounding, however, because it has been working with large numbers greater than one, and so R has more precision to work with. Going this route gives us the answer `987.5311`, which is certainly large but hardly infinite. You will have gotten a different answer—because the list `p` is random and so different every time you execute the code above—but of a similar magnitude.

You might object that one-thousand data points is a lot—how often do we need to worry about this? Quite often. This example misrepresents how many individual values are needed before rounding occurs, because the list `p` above comprises random values uniformly distributed between zero and one. Likelihoods are typically on the low end of probability, because for most models, there are many possible outcomes. This means that any particular outcome is going to be unlikely, even under the true model. For continuous measurement scales, likelihood of even the central value can be very small indeed. As a result, in actual practice, we need to worry about rounding error long before our data set reaches a size of one-thousand. Now keep in mind that R is better about these numerical issues than other software—popular spreadsheet applications in particular are sometimes reckless when it comes to precision.³³ If you are mindful of always doing probability calculations on a logarithmic scale, you should be safe most of the time, within R. Keep in mind though that any computation that works with values very close to zero can run into problems. Later in the book, you’re going to re-parameterize the normal distribution, to avoid such a problem.

A corollary concern arising from the same facts about how the computer processes decimal values is that so-called *floating point* numbers, inside of a computer, are not necessarily equal, even when they should be. The reason is technical, but interested readers can find good explanations of *floating point arithmetic* and its errors in many places. For our purposes, just keep in mind that logical comparisons like:

R code
2.7

```
0.7/7 == 0.1
```

```
[1] FALSE
```

may not tell us what is plain to see. What has gone wrong here is that we compared floating point values in the wrong way. Whenever you want to know if two or more floating point numbers are equal, use instead:

R code
2.8

```
all.equal( 0.7/7 , 0.1 )
```

```
[1] TRUE
```

Using logarithms makes no difference in this case. Just be careful to always compare decimal valued numbers with a command like `all.equal` and not with the binary operator `==`.

2.6.2.2. The brute force search. Now that you're convinced to work on the log scale, we can get down to actually finding the model that minimizes the negative log-likelihood. There are several approaches to the search. The most obvious, but also least efficient, is to calculate the $-\log\text{-likelihood}$ of each model and then pluck from this list the minimum value. The model that corresponds to the minimum is the maximum likelihood estimate.

Since there are an infinite number of possible models, values of p_W , we can't really evaluate every model. But we can evaluate the range of models at a very fine scale. First, let's generate a list of all the models we will evaluate:

```
models <- seq( from=0, to=1, by=0.001 )
```

R code
2.9

Go ahead and type `models` on the R command prompt, once you've executed the line above, and see that the content of this list is a bunch of values for p_W , from 0 to 1 in increments of 0.001. These are the models we will evaluate. There should be 1001 elements in this list. If you don't want to count them, just use `length(models)` to make R count them for you.

Now to compute the $-\log\text{-likelihood}$ for each model:

```
nLL <- -dbinom( 6 , size=10 , prob=models , log=TRUE )
```

R code
2.10

The list named `nLL` should now contain 1001 $-\log\text{-likelihood}$ values, one for each element of `models`. Notice that instead of using the command `log` as before, I've added the parameter `log=TRUE` to the code. All of the built-in likelihood functions in R, such as `dbinom`, recognize this `log` parameter. If you set it to `TRUE`, R will know to do all of the internal calculations on the log scale and produce a log-likelihood as a result. For understanding, you may now want to plot the $-\log\text{-likelihoods}$ you just computed, against the set of models:

```
plot( nLL ~ models )
```

R code
2.11

I do not reproduce this plot for you here. You really should do it yourself.

Now we need just one more line of code to tell us which model in the list `models` corresponds to the smallest $-\log\text{-likelihood}$ in the list `nLL`.

You can see from the plot you just made that the maximum likelihood model is close to 0.6, and of course you know it lies there, because of the analytical solution. But here's how to pluck this answer from the list of 1001 models:

R code
2.12

```
models[ which.min( nLL ) ]
```

```
[1] 0.6
```

The command `which.min` finds the location (index) of the smallest value in a list. Since `models[73]` just returns the 73rd value in the list `models`, the line above will return whichever value in `models` corresponds to the minimum $-\log$ -likelihood in `nLL`.

2.6.2.3. The hill climbing search. The brute force method we just used works in almost all circumstances, but unless your problem is as simple as this one, it can be very inefficient. The set of models we have considered in this chapter, all the values of p_W from 0 to 1, is in principle infinite in size. But it is only one dimension. As soon as we allow models to vary in other ways, say in the rate of measurement error, then the number of possible models explodes in size. What was a search of 1001 models would now be a search of 1001^2 models. That's a little over one-million likelihoods to compute. If we add a third dimension, that's $1001^3 \approx$ one-billion. It is common for regression models to have a dozen or more dimensions along which they can vary. Brute force isn't going to be practical in most cases. Computers are fast, but they aren't that fast, yet.

There are ways to intelligently search the space of possible models, however, that require computing far fewer likelihoods. I'll provide a sketch of how these methods work, but in most cases, you won't have to program them yourself. R already has a very capable and flexible set of algorithms for searching for minimum (or maximum) values of functions. We're not going to be able to improve upon these built-in commands, except in very special circumstances. Later in the book, when we get serious about Bayesian estimation, we'll leave this kind of search behind entirely.

Here's the basic strategy. Let each model have a proportion of water p_W and an implied $-\log$ -likelihood denoted $-\log L(p_W)$. Pick a starting guess for p_W , say $p_W = 0.5$. Compute the $-\log$ -likelihood of this model, which sticking with our water example is $-\log L(p_W = 0.5) = 1.5844$.

R code
2.13

```
-dbinom( 6 , size=10 , prob=0.5 , log=TRUE )
```

```
[1] 1.584364
```

Now consider another model very close to $p_W = 0.5$, say at $p_W = 0.501$. Compute the $-\log$ -likelihood for this model:

```
-dbinom( 6 , size=10 , prob=0.501 , log=TRUE )
```

R code
2.14

```
[1] 1.580384
```

This $-\log$ -likelihood is smaller than the original one, $-\log L(p_W = 0.501) < -\log L(p_W = 0.5)$. This implies that $p_W = 0.501$ is closer to the maximum likelihood estimate—remember, minimizing the $-\log$ -likelihood maximizes the likelihood. So now adopt $p_W = 0.501$ as the current value. Then consider again another model just above it, say at $p_W = 0.502$. Again compare the $-\log$ -likelihoods. Keep stepping down-hill in this way until you reach a value of p_W for which moving in either direction increases the $-\log$ -likelihood.

The algorithms built into R are actually more efficient than the approach outlined just above, mainly because they know how to make larger strategic jumps around the space of models. But they work in the same basic way: by comparing the $-\log$ -likelihoods of neighboring models, they climb down into valleys and find local minima. Using methods like this, the computer can reliably find maximum likelihood estimates using only a tiny fraction of computations needed in a brute force search.

Here's the simplest way to do this in R, for our water problem. We are going to use a command `optim`. This command tries to find the minimum of an arbitrary function that we provide. The first parameter we pass to this function is the value of p_W to start searching at. Then we give it the $-\log$ -likelihood function. It hill climbs in the space of models and returns the result, here stored in a symbol `pw.mle`.

```
pw.mle <- optim( 0.5 , fn=function(pw) -dbinom(6,10,pw,TRUE) )
```

R code
2.15

Warning message:

```
In optim(0.5, fn = function(pw) -dbinom(6, 10, pw, TRUE)) :
  one-diml optimization by Nelder-Mead is unreliable: use optimize
```

After you execute the line above, you should get a warning message, which I've reproduced just above. R can produce “warnings” all the time, especially when fitting models. Warnings do not mean your calculation failed. Much of the time they are harmless, as in this case. But you should learn what the warning means, so you can be sure it is safe to proceed with your result. In this case, the command `optim` is general to finding minimum values in high dimensional model spaces. There are much better algorithms (like golden section search) for searching one

dimensional model spaces, like the one we have here. R is just alerting you to this fact. If you want to search in a way that avoids this warning, use:

R code
2.16

```
optimize( f=function(pw) -dbinom(6,10,pw,TRUE) , c(0,1) )
```

You'll get the same answer.

Let's inspect the details of `pw.mle`. You could just type the symbol `pw.mle` at the R command prompt and press enter to display its contents. But that will display a rather ugly and long list. A better habit to develop for inspecting the contents of R symbols is to use `str`, which means *show me the structure of*:

R code
2.17

```
str( pw.mle )
```

```
List of 5
$ par      : num 0.6
$ value    : num 1.38
$ counts   : Named int [1:2] 26 NA
  ..- attr(*, "names")= chr [1:2] "function" "gradient"
$ convergence: int 0
$ message   : NULL
```

The object `pw.mle` is a *list*, which is a special kind of data in R. A *list* has named elements, and you can extract each by using the name of the *list* followed by a `$` and then the name of the element you want. For example, let's pull out the maximum likelihood estimate itself:

R code
2.18

```
pw.mle$par
```

```
[1] 0.6
```

Any other named element can be extracted in the same way. The second element of `pw.mle`, called simply `value` in the list, is the $-\log$ -likelihood at the maximum likelihood estimate. We're going to need this value in the next section, when we compute confidence intervals. The other element of `pw.mle` to pay attention to right now is `convergence`. A value of zero (0) here is good. That means the algorithm is confident that it found a minimum. The most likely other value you will see is one (1), which means the algorithm stopped before finding a minimum.

2.6.2.4. Local minima. Hill climbing searches are not a panacea. They can be, and indeed in high dimensional problems almost always are, sensitive to where you start searching. For example, in the example above,

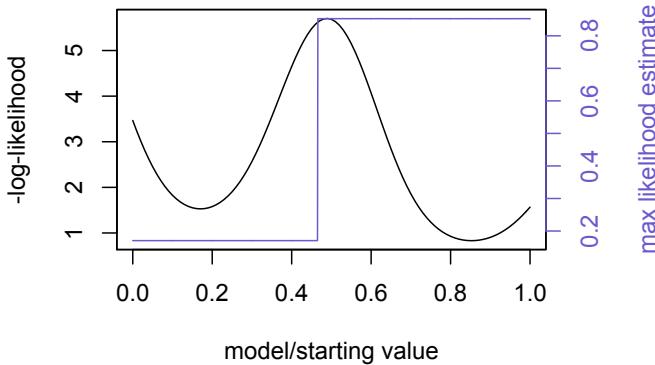


FIGURE 2.7. A $-\log\text{-likelihood}$ function with more than one minimum. The left vertical axis plots the negative log-likelihood across models on the horizontal axis. This particular function has two local minima, complicating the search for a maximum likelihood estimate. Near 0.17, there is a minimum, but the other minimum near 0.85 is lower. It is the global minimum we would like to find. The blue line plots the inferred maximum likelihood estimate (mle), measured on the right vertical axis, across different starting values for search, on the horizontal axis. All starting values below about 0.45 lead to an inferred mle of 0.17, while all starting values above 0.45 lead to an mle of 0.85.

we started searching at $p_W = 0.5$. If you start searching at another value, say $p_W = 0.1$, R will still find the maximum likelihood estimate at $\hat{p}_W = 0.6$, but it will take longer. In more complex problems, there may be more than one local minimum in the $-\log\text{-likelihood}$ function, and hill climbing algorithms can easily get stuck someplace other than the maximum likelihood estimate. FIGURE 2.7 illustrates this problem. If there is more than one valley in the $-\log\text{-likelihood}$ function, as in the figure, then searches that start near a particular local nadir will tend to get stuck in that valley. This makes inferences about the maximum likelihood estimate potentially sensitive to the model we start search with.

One way to cope with this obstacle is to try different and diverse starting models. Another complementary strategy is to use a robust search strategy like *simulated annealing*.³⁴ But there are no guarantees.

2.6.2.5. An easier way. For much of this book, we'll avoid using `optim` directly, only because for more complex models, it can be a chore to set it up correctly and process its output. So we'll work instead with a convenience function, `mle2`, part of the `bbmle` package written by ecologist Ben Bolker.³⁵ This package has a number of flexible tools for fitting and comparing custom statistical models. Provided you are connected to the internet, you can install `bbmle` from within R:

R code
2.19

```
install.packages( "bbmle" , dependencies="Depends" )
```

Like all optional packages for R, make it available for your current session with the `library` command:

R code
2.20

```
library(bbmle)
```

Bolker's book, *Ecological Models and Data in R*,³⁶ is highly recommended. It mainly focuses on models that go beyond standard linear and generalized linear models. So it overlaps quite little with this book in the details and content of examples, although it has a very similar philosophy and also shows the reader a lot of R code.

This library provides a convenient function, `mle2`, to specify and fit maximum likelihood models, and we'll use it a lot in the book. `mle2` actually uses `optim`, so it's not doing anything different, under the hood. But `mle2` is usually easier to use and debug than `optim`. To replicate the result we got from calling `optim` directly, use the code:

R code
2.21

```
pw.mle2 <- mle2( nw ~ dbinom( size=10 , prob=pw ) ,  
                  data=list(nw=6) , start=list(pw=0.5) )
```

Warning messages:

```
1: In dbinom(x, size, prob, log) : NaNs produced  
2: In dbinom(x, size, prob, log) : NaNs produced
```

I'll explain the components of the code in a moment. First, let's talk about those warnings. What are NaNs, and why were they produced, and why does R warn us about them? The reason for warnings in this case is that R tried to calculate likelihoods for a couple of models with values of p_W greater than 1 or less than 0. Since $p_W < 0$, for example,

is not a probability, `dbinom` complains and returns an error named `NaN`, which means *not a number*. Prove this to yourself by executing:

```
dbinom( 6 , size=10 , prob=-1 )
```

R code
2.22

```
[1] NaN
Warning message:
In dbinom(x, size, prob, log) : NaNs produced
```

Any value for `prob` that is less than zero or greater than one will produce the same error. R can nearly always keep searching and find the maximum likelihood estimate, when this happens. The best way to be sure that this harmless condition is what caused your `NaN` warnings is to turn on the search trace, by adding `trace=TRUE` to your call to `mle2`:

```
pw.mle2 <- mle2( nw ~ dbinom( size=10 , prob=pw ) ,
  data=list(nw=6) , start=list(pw=0.5) , trace=TRUE )
```

R code
2.23

You will see a long list of output, the first eight lines of which will be something like:

```
0.5 1.584364
0.501 1.580384
0.499 1.588384
4.500005 NaN
1.300001 NaN
0.6600002 1.461224
0.6610002 1.463922
0.6590002 1.458575
```

Each line of this output is a search step. The first number on each line is a value for p_W . The second number on each line is the $-\log$ -likelihood at that value of p_W . The list begins with 0.5, because that is the starting value we specified. After trying $p_W = 0.501$ and $p_W = 0.499$, the computer tried two values which are not proper probabilities, resulting in `NaN` errors. We didn't tell the computer that the possible models had to constrain p_W to be a value between zero and one, so we should expect errors like this to happen. The computer obediently picked up where it left off, however, and marched onward, eventually locating the maximum likelihood estimate at $p_W = 0.6$. Turning on `trace=TRUE` can help diagnose problems you'll encounter, once you start writing your own likelihood functions, instead of just using the built-in ones like `dbinom`. So keep it in mind.

Now let's look at the code that invoked `mle2` again:

R code
2.24

```
pw.mle2 <- mle2( nw ~ dbinom( size=10 , prob=pw ) ,
  data=list(nw=6) , start=list(pw=0.5) )
```

This code has all the same components that we used to invoke `optim`, but in a form that will turn out to be easier on you. First, `mle2` allows us to specify the statistical model in the form of a *stochastic node*, which is something you don't need to fully understand now, but that will become a fast friend in later chapters. The stochastic node here is everything after the tilde in:

$$\text{nw} \sim \text{dbinom}(\text{size}=10, \text{prob}=pw).$$

Read this as: The data n_W (`nw`) is distributed (\sim) binomially (`dbinom`), with $n = 10$ samples (`size=10`) and probability p_W (`prob=pw`). Second, we have to explicitly tell `mle2` where to find the data `nw`, so we give it a list: `data=list(nw=6)`. In later chapters, this list will reference larger data tables with many variables. Finally, just like `optim`, `mle2` needs to know where to start searching in model space, so we give it a list of starting values: `start=list(pw=0.5)`. When our model space has more than one dimension, you'll see that this list grows in length. Any symbol named in this `start` list must also appear in the stochastic node. Then `mle2` understands that you want it to find the maximum likelihood by searching through values of this symbol. In this case, there is only one, `pw`, which is the probability inside the binomial likelihood.

There will be plenty of examples of code like this, so you'll come to find this stuff familiar, once you use it a few times. It's normal to feel that this is all arbitrary and overwhelming, at first. Learning R's computing language really is like learning a language. Languages are largely arbitrary, and they are vast. At first, a new language is little more than noise. Eventually, you get the rhythm of it. Just be patient with yourself. The good news is that you can do most of what you want with R, with just minimal conversational ability in the language.

So we have our estimate again, this time via `mle2`. When we find a maximum likelihood estimate using `mle2` instead of `optim`, there is a much nicer way to summarize the results:

R code
2.25

```
summary( pw.mle2 )
```

Maximum likelihood estimation

Call:

```
mle2(minuslogl = nw ~ dbinom(size = 10, prob = pw),
```

```

start = list(pw = 0.5), data = list(nw = 6))

Coefficients:
Estimate Std. Error z value   Pr(z)
pw  0.60000   0.15492   3.873 0.0001075 ***
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: 2.766018

```

The summary above provides the output most typically sought, after finding a maximum likelihood estimate: the estimate itself, a measure of the precision of that estimate, and the log-likelihood at the estimate. Let's focus on the coefficients table for now:

```

Coefficients:
Estimate Std. Error z value   Pr(z)
pw  0.60000   0.15492   3.873 0.0001075 ***

```

Each row in this table is a dimension in model space, often called a coefficient or a parameter. There is only one such dimension in this case, the probability p_w . The first value in the row is the estimate, the value of this variable that gives us the model with the highest likelihood. The second value is the *standard error* of the estimate. I explain what this value means and how to compute it yourself, using the likelihood, in the next section. The last two values, a so-called z value and $\text{Pr}(z)$, have to do with a null hypothesis test. They should be ignored, as I'll argue at length in the next chapter.

In fact, if all you want is summary information about the estimates and their precision, better to use an alternative command that comes with the LIBRARY that accompanies this book:

```
precis( pw.mle2 )
```

R code
2.26

	Estimate	Std. Error	2.5%	97.5%
pw	0.6	0.1549193	0.2963637	0.9036363

The `precis` command, after French *précis* (pronounced something like *praysee*), provides a short summary of the coefficients only. I use it throughout the book, as well as in my daily work. It's certainly not all the information you'll need, but at least it doesn't distract you with those significance tests. The estimate and standard error columns are the same as those provided by `summary`. But instead of those pointless *P*-values (see an extended argument against them in Chapter 3), you get instead a 95% quadratic estimate confidence interval. The next major section

of this chapter goes into what these confidence intervals are and how to compute them.

2.7. Confidence: More Than the Maximum

Most readers have probably watched track and field events at the Summer Olympics or similar competitions. In a race like the 100 meter dash, we can almost always identify a single winner, the competitor with the shortest finishing time. Is this winner the fastest runner? In this single race, yes. But what about on average? To make this more interesting, suppose the same runners will compete again in the same race in one month's time. If you are asked to predict the outcome of this future race, how will you do it? Surely you will predict that the same runner will win again. That is the most likely outcome. But how much money would you be willing to wager on that outcome? To answer this question, you would want to know the difference in finishing times between the two runners, on the first race. If the first runner in our imaginary race finished 2 seconds ahead of the second runner, then that runner's win in the future race is almost guaranteed. But if instead the difference in times was only 0.01 seconds, you would understandably be less confident in your prediction that the same runner would win again.

Our models are like these runners. One of them makes better predictions on average, like one of the runners is faster on average. On any particular race (data set), any runner might win (have the lowest negative log-likelihood). The maximum likelihood estimate is the model with the highest posterior probability, provided we are willing to adopt an uninformative prior or one of the other tactics that gets us from naked likelihoods to posterior probability. So far, you've seen how to actually find such an estimate, using a statistical model of your measurements. This helps us estimate which model (runner) is best. The vast majority of applied statistics in scientific journals seek and report these maximum likelihood estimates, peaks of naive posterior densities.

But what's missing is a way to talk about our *relative confidence* in the estimate. Presenting only the maximum likelihood estimate, the peak of the naive posterior, can be very misleading. Posterior probabilities are relative measures of confidence, relative to other explicitly nominated models. When many different models have nearly identical likelihoods, it's like many runners having nearly identical finishing times. In those cases, even though one model (runner) may truly be the best, the differences are so small that it makes no sense to produce confident predictions for the next race.

A common intuition is that knowing the likelihood at the peak provides some measure of absolute confidence. The problem is that raw likelihoods tend to be uninterpretable in this way. Even the model that maximizes the likelihood tends to have a very small likelihood. So like we compare runners, we compare models: By relative finishing times. The entire posterior distribution—so far approximated by the likelihoods alone—gives us a useful way to conduct these comparisons. We want to use the entire posterior distribution to make inferences, instead of plucking out the maximum likelihood and discarding the rest. And as always, just because someone wins the race does not mean that person is a good runner. So there is always a need to check models beyond estimating parameters.

The question for the remainder of this chapter is how we can summarize the posterior distribution to communicate confidence across models. If you've been waiting for confidence intervals to make an appearance, here they come. But you'll also see how to sample models from the naive posterior. This sampling approach will turn out to solve many analytically-challenging problems for you in later chapters. But you won't solve them analytically. Instead you'll use simulations from the posterior to solve them empirically.

Here's the plan. First, you'll see how the shape of the posterior density relates to relative confidence in models. Second, you'll see how to draw samples from the naive posterior. Then you'll see how confidence intervals can be computed from these samples, in an intuitive empirical way. Confidence intervals provide concise summaries of aspects of the shape of the posterior. When questions about models are very focused, such intervals are a natural approach. But remember, you can always show the actual posterior itself, or the distribution of samples from it.

2.7.1. Describing the naive posterior. A thorough approach is just to plot the entire *naive posterior*, which in the example so far is just the standardized likelihood function, $L(p_W|n_W, n)$. We care about the naive posterior which recall is the posterior under the assumption of a weak prior, because the shape of the entire curve provides information about how the evidence reflects on the models. You've already seen likelihood functions for the proportion of water models. Let's look at such curves again and examine how the shape of a curve provides information about confidence in different models. Then you'll see how to use R code to simulate samples from this posterior. These samples will solve a lot of problems for you, and we'll use them throughout every part of this book.

2.7.1.1. Shape and confidence. FIGURE 2.8 plots several different likelihood functions, for three different sample sizes and two different maximum likelihood estimates. The plot in the upper-left shows the likelihood across values of p_W for observed data in which the sample proportion is $p_W/n = 0.6$. Each of the three curves represents a different sample size n , the number of tosses of the globe. The outermost curve is for $n = 10$ tosses. The next is for $n = 20$, and the smallest curve is the likelihood function for $n = 100$ tosses. While the maximum likelihood estimates are all $\hat{p}_W = 0.6$, the shapes of these curves are quite different.

The first thing to notice is that the absolute height of the maximum likelihood estimate declines, as sample size increases. The more data one collects, the more possible combinations of observations are possible. Consider the smallest possible sample, $n = 1$. There are only two possible observations in this case, $n_W = 1$ and $n_W = 0$. If we add a second sample, the number of possible observations increases to 3: $n_W = 2$, $n_W = 1$, and $n_W = 0$. If we add a fourth sample, $n = 4$, there are four possible observations, and so on, such that $n = 100$ could produce 100 different observations. Now, we only observe one particular observation in each case, but the likelihood has to spread itself across all possible observations. In a very large sample, any particular combination of observations is highly unlikely. As a result, likelihoods decline as sample size increases.

This is a principle reason that absolute likelihood, or log-likelihood, is of little use. We must instead compare the relative likelihoods of different models. One way to do that is to inspect the rate at which the likelihood function declines, as we move away from the maximum likelihood estimate. As sample size increases, holding the maximum likelihood estimate constant, the curve is steeper. This may be hard to appreciate in the top-left plot of FIGURE 2.8. So instead, look at the plot in the top-right, which shows the same likelihood functions, but on the $-\log$ -likelihood scale. On this scale, it is much easier to appreciate differences in the rate of decline in likelihood, as we move away from the maximum likelihood estimate. At $n = 10$, the likelihood function declines slowly, as we move away from $p_W = 0.6$. This reflects uncertainty in the region of the maximum likelihood estimate, where many models with p_W near 0.6 have nearly the same likelihood. With a large sample however, likelihood declines more rapidly. At $n = 100$, even models within 0.01 of \hat{p}_W have much lower likelihood than the maximum likelihood estimate.

The bottom row in FIGURE 2.8 shows the same relations, but now for cases in which $\hat{p}_W = 0.1$. Notice now that the likelihood functions are very asymmetric, reflecting greater uncertainty above the maximum

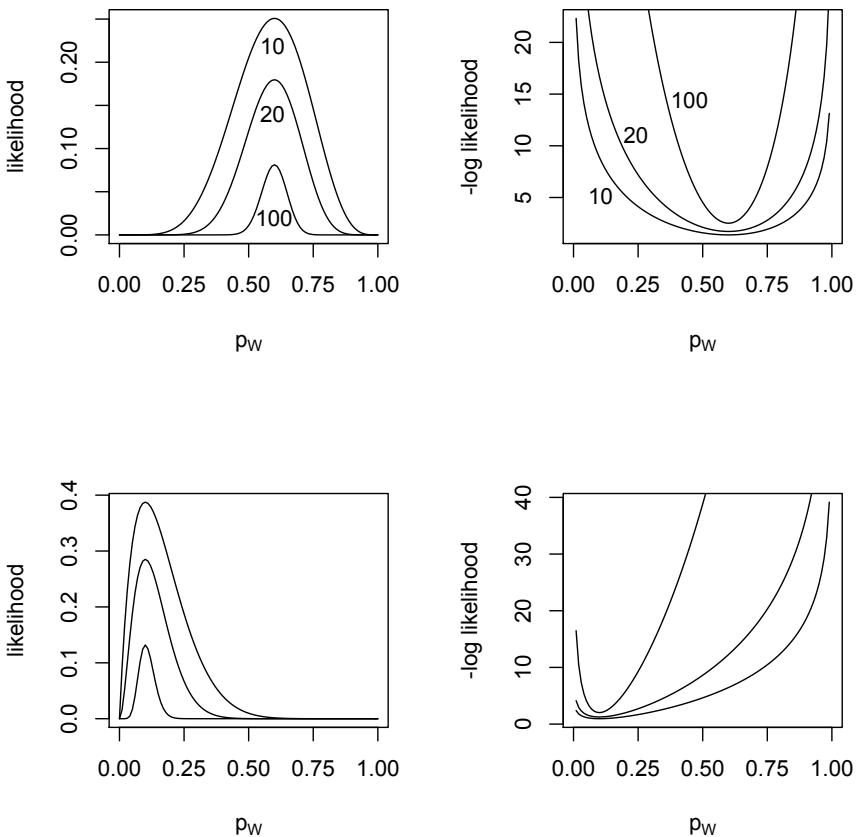


FIGURE 2.8. The likelihood function (unstandardized naive posterior) and confidence in models. In each of the four plots, the three curves represent likelihood functions at three different sample sizes, $n = 10$, $n = 20$ and $n = 100$. The top row shows these functions for cases in which the maximum likelihood estimate is $\hat{p}_W = 0.6$. The bottom shows these functions when the maximum likelihood estimate is instead $\hat{p}_W = 0.1$. The left plot in each row shows the raw likelihood scale, while the right plot shows the same functions on the $-\log$ -likelihood scale, making it much easier to appreciate the different curvatures as a function of sample size.

likelihood estimate than below it. If you think about it for a minute, this had to be true. When the maximum likelihood estimate is near $p_W = 0$, as in this case, then we know that the true value can't be much lower. But there may be considerably more uncertainty about our error in the upwards direction. The naive posterior incorporates this intuition automatically.

As I keep warning, just because likelihoods and naive posterior probabilities must be judged relative to one another, that does not imply that some way of evaluating the absolute fit of a model isn't useful. So far, we're concerned with comparing models in a fixed set of candidates. Another way to say this is that we are concerned with choosing parameter values. This is a problem related to but distinct from deciding if any of the candidate models is any good for some purpose. I keep saying this, because I want to plant this distinction in your mind, where it will incubate and later emerge into practice. We will always need something more than posterior probabilities.

2.7.2. Sampling from the naive posterior. Before you start summarizing the shape of the posterior with intervals, it is helpful to learn an approach for describing the entire posterior that is very flexible and usually much more intuitive than mathematical statements about coverage etc. This approach is to treat the posterior like the probability density it is and *sample* from it. When you sample from the posterior, the samples are models. For now, these models are just parameter values. Once you have samples from the posterior, you can treat them like data, using them to compute various kinds of confidence interval or even to simulate predictions.

2.7.2.1. Simulating samples from the posterior. Here's the general notion. Once you have an estimate of the posterior density, you have a probability distribution of models. This distribution defines a sampling space for generating predictions from the model. Each unique model, say $p_W = 0.2$, predicts a unique pattern of data. You already know the naive posterior in the case of the proportion of water problem. The posterior probability of each value of p_W is simply proportional to the likelihood:

$$\Pr(M_{p_W} | D) \propto \Pr(D|M_{p_W}).$$

In this case, the likelihood function is pretty simple, and if you fluent in the necessary mathematics, it isn't hard to compute all kinds of useful things about this posterior. But most natural and social scientists are far from fluent in the necessary mathematics. Most know what an integral is, but when asked to actually manipulate one, a kind of terror sets in.

An empowering approach that allows for most of the same useful calculations, but which requires less integral calculus, is to sample from the posterior and then perform “empirical” calculations on the samples. As a bonus, when you eventually learn how Markov Chain Monte Carlo (MCMC) estimation works (Chapter 11), you’ll see that it provides nothing other than such samples. So everything you learn here will be immediately applicable to making inferences from MCMC estimations.

Here’s the recipe.

- (1) Estimate the posterior density, as a list of probabilities corresponding to models (parameter values).
- (2) Sample repeatedly from the posterior distribution. You end up with a list of parameter values. If you sample enough, the proportion of each parameter value in this list will converge to the posterior probability.
- (3) Finally, use the samples from the posterior to make calculations.

The main tasks we’ll use these samples for is to calculate confidence intervals and to simulate sampling of new data.

In simple one parameter cases, like the proportion of water problem, all this sampling may seem unnecessary. It’s relatively easy to get exact probability statements for such simple formal distributions. But in later chapters, when there are many parameters in the models, it won’t be so easy. In those cases, the posterior densities of different parameters will be correlated with one another. Then accurately reflecting the joint impact of these correlations on uncertainty is usually well beyond the typical mathematical training of a typical natural or social scientist. However, once you learn the sampling approach, it extends into more complex models quite easily. And again, once you learn this stuff, it prepares you automatically to analyze MCMC output.

2.7.2.2. Sampling proportions of water. We need an example to make more sense of this. In the proportion of water example, we can compute the naive posterior by using the likelihood function. You’ll compute it at intervals of 0.001, using code you’re familiar with by now:

```
models <- seq(from=0,to=1,by=0.001)
post <- dbinom( 6 , size=10 , prob=models )
```

R code
2.27

The symbol `post` now contains likelihoods for all of the models in `models`. These likelihoods are proportional to the naive posterior probabilities. Go ahead and plot the curve these likelihoods imply:

R code
2.28

```
plot( models , post , type="l" )
```

The function isn't really a proper posterior distribution, because it hasn't been standardized so that it sums to one. But that isn't an obstacle for what we're going to do, which is to use the relative proportions of the likelihoods to sample models.³⁷ To draw 10-thousand random samples of models from this un-normalized naive posterior distribution, we can make use of the handy `sample` command, which is specialized for just this kind of task:

R code
2.29

```
samples.pw <- sample( models , size=10000 , prob=post ,
                      replace=TRUE )
```

You just told R to draw 10-thousand values of p_W randomly, in proportion to their posterior probabilities. The command `sample` will automatically ensure that `post` is normalized, so that's why you didn't have to do it yourself. Values of p_W may appear more than once in the resulting list `sim.models`, and the `replace=TRUE` parameter in the code ensures this.

FIGURE 2.9 demonstrates the relationship between these samples and the posterior probabilities themselves. In the lefthand plot, I've plotted each sample in `samples.pw`, with sample number on the horizontal axis and the sampled value itself on the vertical axis. The points scatter all over, because there is considerable uncertainty as the value of p_W . But they do cluster around $p_W = 0.6$, the maximum likelihood estimate. Each value of p_W appears in these samples with proportion approximately equal to its posterior probability. In the righthand plot, I show the density resulting from the point samples, in blue. It is jagged, because such a random sampling approach is not exact at fine scales. The black curve that overlays this density is the actual naive posterior, computed in `post`. You can see that the samples provide a quite good approximation of posterior probability. If you want an even better approximation, you can increase the number of samples, by changing the `size` parameter in the call to `sample`.

What to do with this collection of samples, `samples.pw`, now that we have it? There are two tasks that you'll perform over and over again in this book. The first is to use the posterior samples to empirically compute confidence intervals. By using this pseudo-empirical approach, you'll be able to estimate confidence intervals for all manner of fancy models with any number of parameters. The second is to simulate data (water and land) sampling from it, and thereby generate model-based predictions

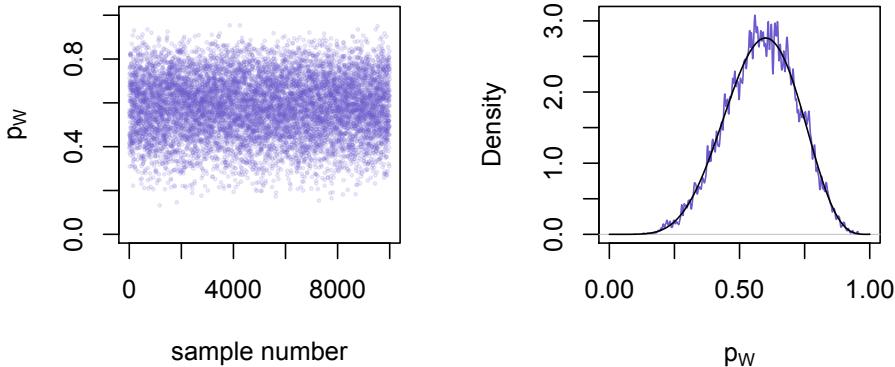


FIGURE 2.9. Samples from the naive posterior of p_W . On the left, the individual samples are ordered along the horizontal axis. Density of the points is greatest near $p_W = 0.6$, the maximum likelihood estimate. On the right, plotting the density (similar to the histogram) of these samples in blue shows an excellent approximation to the actual naive posterior, shown in black.

that incorporate all of the uncertainty contained in the posterior density. These predictions allow one way to evaluate the absolute adequacy of a model for some defined purpose.

2.7.2.3. Sampling from complex posteriors. Before moving on to show you how use these samples, it's worth cautioning that, once you have models with more than one dimension (parameter), there will be some wrinkles to work out in getting your samples from the posterior. It can be computationally expensive and tricky to sample from complex posterior densities. You'll see exactly why in Chapter 4. And at that point, you'll meet some ways to address the problem. But however you get the samples, they are treated the same way once you have them.

2.7.3. Confidence intervals. What FIGURE 2.8 (page 77) illustrates is that the shape of the naive posterior gives us information about relative confidence in models. When the region near the maximum likelihood is relatively flat, it is hard to confidently claim that the best value of p_W must be the maximum likelihood estimate. For many problems, plotting the naive posterior is a good way to communicate to your audience the

uncertainty in your estimates. This is particularly the case when the naive posterior has a large flat region or is highly asymmetric.

Still, the purely visual presentation has its limitations. These pictures take up a lot of space. In a complex model with a dozen parameters, you probably don't and shouldn't use up an entire page displaying their densities. Often, the posterior for a particular parameter can be very narrow, and then all you need to do is reassure your audience of this. In these cases, much of the relevant information about the shape of the posterior can be communicated with *intervals*. There are two major types of intervals of interest: (1) Intervals with predefined boundaries for which we wish to know how much probability mass lies within, and (2) intervals with predefined mass for which we wish to know the upper and lower bounds.

2.7.3.1. Predefined intervals of variable mass. Often you want to know that total probability mass between some predefined parameter values. These predefined limits may come from some practical interest. For example, Box and Tiao³⁸ analyze data on the breaking strength (in grams) of 20 samples of yarn—exciting stuff, I know. Don't laugh too much, though. Yarn is serious business. Manufacturing desires, for the sake of the example, that the breaking strength be no less than 48 grams but also no more than 51 grams. If you can calculate the total probability that lies between these predefined limits, you can say whether or not these data suggest that the yarn is up to specification. Once you have samples from the posterior, this is a simple matter of counting up all of the samples that lie between 48 and 51 grams. Then divide this count by the total number of samples, and you'll have the total probability mass within the interval.

In basic science, rarely do manufacturing limits predefine our intervals. But it is routine and useful to be interested in how much probability mass lies above or below a parameter value of zero, for example. Indeed, much gnashing of teeth over regression coefficients is concerned with this question. When the parameter is a probability, like our p_W , then one might instead be interested in the mass on either side of 0.5. The right boundaries for such an interval will always depend upon the context.

For example, suppose in the proportion of water problem you wish to know how much probability mass lies between $p_W = 0$ and $p_W = 0.5$. This is the probability that the best value of p_W for prediction is less than one-half. If you wish to solve this problem analytically, you could use integral calculus or look up the value in a special table. But once you

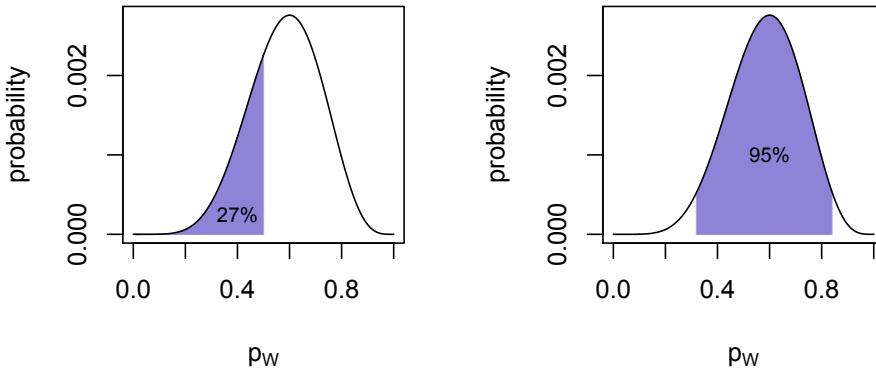


FIGURE 2.10. Two kinds of confidence interval. On the left, an interval with predefined boundaries, in this case seeking all of the probability mass between $p_W = 0$ and $p_W = 0.5$. On the right, an interval with predefined probability mass but variable boundaries.

have samples from the naive posterior, all you have to do is count. This line of code gives you the number of samples within the interval:

```
length( samples.pw[ samples.pw <= 0.5 ] )
```

R code
2.30

```
[1] 2728
```

Read the code inside the `length` command as *give me all of the elements of `samples.pw` that are less-than-or-equal-to 0.5*. And since there are 10-thousand samples in total (use `length(samples.pw)` if you doubt it), the probability that the best value of p_W is less than or equal to 0.5 is $2728/10000 \approx 0.27$. If you want to explicitly define the lower bound of the interval too, you need code like this:

```
length( samples.pw[ samples.pw >= 0 & samples.pw <= 0.5 ] )
```

R code
2.31

You can verify that this gives the same answer, 2728. I plot the region defined by this interval in FIGURE 2.10. One thing you could say from such a calculation is that the odds that more of the Earth is covered with water than with land are $(1 - 0.27)/0.27 \approx 2.7$. (Or rather, the odds that the best approximation assumes more water than land is about 2.7.)

2.7.3.2. Variable intervals of predefined mass. More common in science journals is the second type of interval, one with predefined probability mass. These intervals are useful for communicating the shape of the posterior, when a full picture of it is either cumbersome or unnecessary. The goal is to define the lower and upper boundaries of an interval of parameter values that includes some predefined probability of all models. The most common predefined probability is 0.95, corresponding to a 95% confidence interval. There are actually a few different ways to define such an interval, although the differences rarely matter in any applied context with an even moderately large sample size. In the context of small samples, you are probably better off inspecting the entire posterior, if you can get it.

A useful and common way to define such an interval is to ask for the narrowest interval that contains the predefined mass. This will be the *highest posterior density interval* (HPDI).³⁹ Here's a heuristic way to find such an interval. Suppose you have a predefined probability mass \mathcal{P} . This could be 0.95 or 0.5 or any other probability. Then imagine submerging the entire posterior density in water. Now gradually let out the water, letting the peak of the posterior emerge like an island from the sea. As the peak emerges, add up all of the exposed probability mass. If the exposed mass is less than \mathcal{P} , then keep lowering the water. As soon as the the exposed probability is equal to \mathcal{P} , stop letting out the water. Find the points where the posterior curve meets the water line on the left side and on the right side. These will be the lower and upper bounds of the highest posterior density interval.⁴⁰ This kind of interval is also sometimes called the *highest density interval*, HDI.

It would be a chore to write the code to do that yourself. If you are using the LIBRARY of code that accompanies this book, you can use HPDI to compute the interval from a sequence of samples from the naive posterior. For example, to compute the 95% highest posterior density interval for p_W :

R code
2.32

```
HPDI( samples.pw , prob=0.95 )
```

```
lower upper
0.319 0.840
```

This command is actually just a convenient interface to the **HPDinterval** function in the **coda** package. So install **coda**, which will also be useful to you later on, if you decide to work with MCMC estimation. FIGURE 2.10 displays in the righthand plot the confidence interval corresponding to this 95% HPDI.

With a symmetrical posterior distribution, like the perfect bell curve of a normal distribution, the HPDI will have equal-sized tails. But with an asymmetric posterior like in FIGURE 2.10, this is not necessarily true. If you instead want an interval that ensures that there is equal area in each of the tails outside of the interval, you will want to use a simple percentile confidence interval. There are some key advantages to these simpler intervals.⁴¹ These are also extremely easy to calculate, once you have samples from the posterior. For a 95% confidence interval that assigns 2.5% of the probability to each tail, you can use the very flexible `quantile` command:

```
quantile( samples.pw , probs=c( 0.025 , 0.975 ) )
```

R code
2.33

```
2.5%    97.5%
0.305000 0.834025
```

Or for convenience you can use the `PCI` function in the LIBRARY of code that accompanies this book. It produces the same answer:

```
PCI( samples.pw , prob=0.95 )
```

R code
2.34

```
2.5%    97.5%
0.305000 0.834025
```

Call such a confidence interval a *percentile confidence interval*.

In practice, the differences between such simple percentile intervals and highest posterior density intervals are minor. The HPDI will be narrower, so it usually represents a better statement about the shape of the posterior, and I'll use it throughout the book. But if different kinds of intervals result in very different bounds, then you should probably not use an interval at all to summarize the posterior, but instead report the entire posterior. Remember, it is the posterior that theory tells us is relevant, in all cases. No strong theory tells us that any particular interval calculation is relevant.

For much of the book you'll also be using large-sample approximations of the posterior that render the HPDI and percentile intervals the same. And that's where we turn next. A curious and useful thing happens to the naive posterior, as sample size increases, leading to the most common method of calculating confidence intervals.

2.7.4. Quadratic approximation. When calculating the full naive posterior using likelihoods is impractical—and it will be in later chapters—an approximation that works well for large samples is *quadratic estimation*. The quadratic estimate is usually what is presented as a standard error in

a table of coefficients, like the 0.15492 value in `mle2`'s output from the previous section. This number is an estimate of the standard deviation of the naive posterior, derived by assuming that the posterior is a perfect normal distribution.

Why is this sensible and useful? If the naive posterior is normal, then it can be completely described by its first and second derivatives. This is because normal distributions have no higher-order derivatives than those. This is lucky, because it means we really only need the second derivative. Why? We want to approximate the posterior near the maximum likelihood estimate. At that point, the first derivative must be zero, otherwise it wouldn't be a maximum. So the first derivative is zero, leaving only the second to estimate. Once we have the second derivative, we can compute anything we like about the naive posterior, provided we are happy to assume it is shaped like a normal distribution.

This is not the most obvious approach, and in my experience, most students never received a solid education in what standard errors mean and what assumptions they are premised on. I am sorry to admit that I have sent more than one student crashing in flames on a PhD qualifying exam, because he or she couldn't answer a couple of basic questions about what the "se" values in standard regression output mean. So let's work through the proportion of water case and replicate the 0.15492 from `mle2`'s coefficients table. Of course we don't have to use the quadratic estimate at all in this case—we can easily compute the actual naive posterior, as we have done several times already. But it helps to explain this approximation for a case in which the exact naive posterior and the quadratic estimate can be compared, because the approximation is not always a good one. And you really should know what those "se" things are, anyway. They are information about the shape of the posterior.

2.7.4.1. Finding the curvature of the likelihood function. In the case of simple binomial models, like the one that has consumed us in this chapter, it turns out to be easy to derive the quadratic estimate analytically. All we need is the second derivative of the log-likelihood. That will tell us the quadratic curvature of the likelihood function (which recall the unstandardized naive posterior). We calculated the first derivative of the log-likelihood before to be:

$$\frac{\partial}{\partial p_W} \log L(p_W | n_W, n) \equiv \frac{\partial \log L}{\partial p_W} = \frac{n_W - np_W}{p_W(1 - p_W)}.$$

Just differentiate again, with respect to p_W , to get the second derivative:

$$\frac{\partial^2 \log L}{\partial p_W^2} = \frac{\partial}{\partial p_W} \left(\frac{\partial \log L}{\partial p_W} \right) = \frac{n_W - n}{(1 - p_W)^2} - \frac{n_W}{p_W^2}. \quad (2.6)$$

Now since we want to know the curvature in the region of the maximum likelihood estimate, and for the current data, substitute in $p_W \rightarrow \hat{p}_W = 0.6$ and $n_W \rightarrow 6$ and $n \rightarrow 10$. This evaluates to:

$$\frac{\partial^2 \log L}{\partial p_W^2} \Big|_{p_W \rightarrow \hat{p}_W} = \frac{6 - 10}{(1 - 0.6)^2} - \frac{6}{0.6^2} \approx -41.667.$$

This value is the acceleration of log-likelihood, as we move away from the maximum likelihood estimate. It is negative, because the log-likelihood gets smaller at a faster rate, as we move away from \hat{p}_W .

2.7.4.2. From curvature to confidence interval. Now what do we do with this number, -41.667 ? What we need at this point is an argument that gets us from the number above—the curvature of the likelihood function in the region of the maximum likelihood—to probability. We want to be able to say what range of models corresponds to 95% (or any other percent) of the posterior probability. Here's how we do it.

First, realize that when we summarize the log-likelihood function with its second derivative, as we did above, we are implicitly approximating that the log-likelihood is *parabolic*. A parabola, recall from secondary school, is just a quadratic function like $y = -x^2$. It turns out that if a function is parabolic on the log scale, it is Gaussian on the probability scale. So by assuming that the log-likelihood is parabolic, we are simultaneously assuming that the likelihood is normal. Most people were never taught this fact, so take a moment to plot a simple parabola on both regular and exponentiated scales:

```
par(mfrow=c(1, 2))
curve( -x^2 , from=-3, to=3 , main="a parabola" )
curve( exp(-x^2) , from=-3, to=3 , main="exp parabola" )
```

R code
2.35

I'm not reproducing those graphs in this book, because I want the reader to really engage with the code. Once you execute the three lines above, you'll see that a Gaussian function, the familiar bell curve, is just an exponentiated parabola, $\exp(-x^2)$. The rest of the junk in the normal probability formula (see next paragraph) is just to standardize, scale and translate the bell curve. The characteristic bell shape comes from the underlying parabola.

So if we approximate the posterior probability with a normal distribution, how do we relate the -41.667 above to the normal? To see, let's analytically take the second derivative of the normal likelihood function. The result should tell us how to relate the parameters of the normal distribution to the numeric curvature estimate above. The probability density for a normal is:

$$L(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

where x is an observed value, μ is the mean, and σ^2 is the variance. We're treating this as the likelihood of the observed mean x , when the true mean is μ and the variance of the likelihood function is σ^2 . (If you don't recognize this formula, or don't understand where it comes from, that's okay. You'll get a crash course in this important density in a later chapter.) The log-likelihood is then:

$$\log L = -\frac{(x-\mu)^2}{2\sigma^2} - \frac{1}{2} \log(2\pi) - \log(\sigma).$$

Now to compute the curvature. The derivative with respect to x is:

$$\frac{\partial \log L}{\partial x} = -\frac{x-\mu}{\sigma^2}.$$

And finally the second derivative, yielding the curvature, is:

$$\frac{\partial^2 \log L}{\partial x^2} = -\frac{1}{\sigma^2}.$$

Solving for σ^2 , the variance of the normal probability density, we get:

$$\sigma^2 = -\frac{1}{\partial^2 \log L / \partial x^2}.$$

In other words, the variance of the posterior is approximately—if the posterior is approximately normal—the reciprocal of the second derivative of the negative log-likelihood. So if we can estimate the second derivative of the negative log-likelihood, all we have to do is divide one by it to get an estimate of the variance of the posterior distribution.

Back to our proportion of water estimate, $\hat{p}_W = 0.6$. We computed the curvature of the log-likelihood at the maximum likelihood estimate to be about -41.667 . So it follows then that our estimate of the variance of posterior distribution for \hat{p}_W is:

$$-\left(\frac{1}{-41.667}\right) \approx 0.024.$$

Most functions in R use the standard deviation of a normal, σ , instead of the variance, σ^2 . The standard deviation is just the square root of the variance, $\sigma = \sqrt{\sigma^2}$. The standard deviation here is therefore about 0.1549—that’s the standard error from the coefficients table produced by `summary(pw.mle2)`. So if we want to find the values of \hat{p}_W that enclose 95% of the probabilities of models, then we just ask R:

```
qnorm( c(0.025, 0.975) , mean=0.6 , sd=sqrt(1/41.667) )
```

R code
2.36

```
[1] 0.2963649 0.9036351
```

This is functionally identical to just adding and subtracting the z -score for a 95% interval, 1.96, to the maximum likelihood estimate, which is an approach you might already know. That is, the upper and lower bounds are given by: $\hat{p}_W \pm 1.96 \times \sqrt{1/41.667}$. If you prefer, here’s some compact code for that:

```
0.6 + c(-1,+1)*1.96*sqrt(1/41.667)
```

R code
2.37

```
[1] 0.2963593 0.9036407
```

Note the trick of using `c(-1,+1)` to simultaneously subtract and add the deviation to \hat{p}_W .

2.7.4.3. The quadratic approximation gets better as sample size increases. One thing to note in the above argument is that it focuses on uncertainty in μ while ignoring uncertainty in σ . In general, that is not a good idea. All parameters are uncertain to some extent, so the logic of probability compels us to try to account for the joint uncertainty. Many readers will already know that when one is uncertain about both μ and σ , one should be concerned with something called a t distribution, rather than a normal distribution. But the quadratic approximation is a good approximation exactly when the joint uncertainty between μ and σ is of little importance: when the sample size is large. Likewise, as sample size increases, the t distribution rather quickly converges to a normal distribution.

I plot our normal approximation against the actual likelihood, in FIGURE 2.11. On each row, I plot both the likelihood (left) and $-\log$ -likelihood (right) for our proportion of water model, at different sample sizes. Pay attention to only the top row first, where $n = 10$ tosses total, like the worked example. The orange dashed curve in both plots is our normal approximation, with variance $1/41.667$ and mean 0.6. The solid curve is the actual binomial likelihood. The orange vertical dashed

lines are the confidence limits we just derived. The solid vertical lines are the confidence limits from the naive posterior itself, no approximations. When these vertical lines do not overlap, it is due to error in the quadratic approximation. Notice that the upper bound of this confidence interval overshoots the answer we got by explicitly mapping out the true likelihood function, while the lower bound gets very close to it. What has happened here is that the normal approximation to the binomial likelihood does not care about the edges at $p_W = 0$ and $p_W = 1$. As a result, some of the probability mass for the normal approximation actually overshoots $p_W > 1$. This leads the upper tail to be thicker than it should, and so the quadratic estimate leads us to believe the upper 95% confidence bound is wider than it really should be, but not by much. However, if the maximum likelihood estimate were, say, $\hat{p}_W = 0.9$ instead of 0.6, then the error of the quadratic approximation would be much larger. You'll prove that to yourself in the exercises at the end of this chapter.

The right-hand plot on the top row of FIGURE 2.11 shows the same two functions, but now on $-\log\text{-likelihood}$ scale. It is much easier to see deviations in the tail of the distributions, on this scale. The approximation (red) and actual likelihood (solid) are quite close until extreme values. It should be much easier to see here what we proved earlier: on the log-likelihood scale, the normal distribution is parabolic.

The bottom row in FIGURE 2.11 repeats the same two plots, by now for a sample size of $n = 50$, 10-times as many tosses of the globe. The approximation is much closer to the actual likelihood function, now. This turns out to be a general property of the quadratic approximation: it is increasingly accurate as sample size increases. The 95% confidence interval is also much narrower now, because more of the posterior is bunched up around $\hat{p}_W = 0.6$. While the red curve and solid curve do differ quite a lot in the tails still—this is still quite obvious on the log-likelihood scale—since the confidence interval is narrow, these deviations matter much less. Don't be fooled by these examples, however, into thinking you only need $n = 50$ to get a perfect quadratic estimate. Depending upon the details of the data, the models, and the method of approximating the standard error, it might take many more samples or even many fewer to get a good fit between the approximation and the actual likelihood.

How much data is needed to get a good approximation will also depend upon how much of the full posterior is of interest. If you are only interested in rather narrow confidence intervals, for example, then even

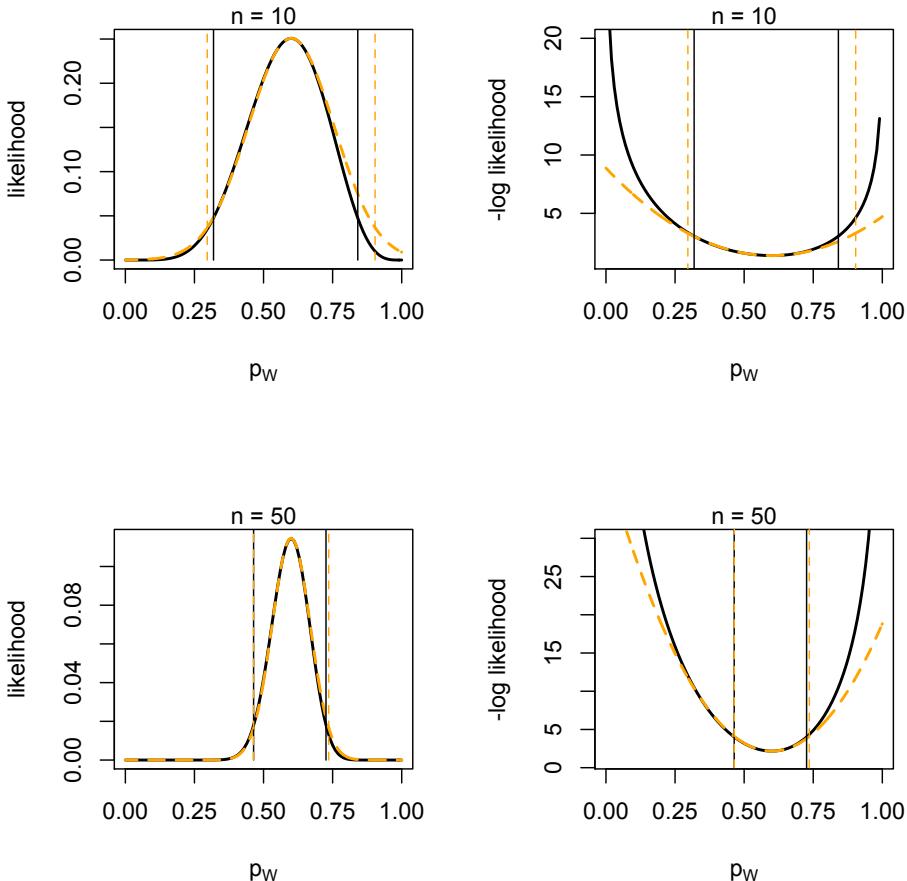


FIGURE 2.11. The accuracy of the quadratic approximation. The orange dashed curve in each plot represents the normal approximation to the likelihood. The solid curve in each plot is the actual likelihood. The orange dashed vertical lines show the 95% confidence interval, computed from the quadratic estimate. The solid vertical lines show the same interval, computed directly from the true likelihood. Plots on the left side show these functions on the raw likelihood scale. Plots on the right show the same functions on the negative log-likelihood scale. Top row: At $n = 10$ tosses of the globe, the upper tail of the approximation deviates visibly from the true likelihood. Bottom row: At $n = 50$ tosses, the normal approximation and actual likelihood are much more similar, especially in the 95% confidence region.

rather modest samples will provide a good estimate of the actual curvature. This is because the region near the peak of likelihood function converges to a normal much faster than do the tails. Look again at the top row plots in FIGURE 2.11. While the actual likelihood function and the quadratic approximation deviate quite a lot in the tails, near the peak, they are nearly identical, even at this very modest sample size. This is a quite common result.

But when the maximum likelihood estimate, the peak of the naive posterior, is near a boundary like $p_W = 0$ or $p_W = 1$, then even a quite large sample will still result in a considerably skewed likelihood function. Even rather narrow confidence intervals can be very non-normal in this case. So another fact to consider in questioning the accuracy of the quadratic approximation is where the peak is located. Always be cautious of *boundary estimates* of all sorts. This is a serious problem that we will return to later in the book, when we begin to seriously consider the estimation of variance components, such as those in hierarchical models.

2.7.4.4. The quadratic approximation recipe. Most of the time, R's functions handle all of the steps in computing a quadratic approximation for you, with minimal supervision. But you really want to understand what is going on, because otherwise you won't understand what the standard errors and such really mean. So let's summarize the process of computing a quadratic approximation. This will just be a summary of the steps you've already seen above. Then I'll provide some brief code that lets you compute it, using either `optim` or `mle2`. Assuming you already have a maximum likelihood estimate in hand, there are three steps.

- (1) The first step is to estimate the second derivative of the likelihood function. If you can do this analytically, that is obviously best. If that's not an option, then it is possible to estimate this second derivative numerically, just like one can estimate a first derivative numerically. How? The intuition is just compute the first derivative at two points, take the difference, and divide the difference by the distance between the points. As the distance becomes very small, you get an accurate estimate of the derivative. Luckily, R will usually do a good job of approximating the second derivative for you. But if it fails, it's nice to know it isn't magic, and that you can do it yourself.
- (2) Once you have an estimate of the second derivative, convert it to a standard error. If the second derivative of the log-likelihood,

evaluated at the maximum likelihood estimate is called H , then the standard error is $\sigma = \sqrt{-1/H}$.

- (3) Finally, compute approximate confidence intervals. If all you want is a standard 95% interval, then just multiply the standard error by 1.96 (the “ z -score” for a 95% interval) and add/subtract this product to/from the maximum likelihood estimate, to get upper/lower bounds. To get z -scores for other interval widths, just use `-qnorm((1-0.95)/2)`, replacing `0.95` with the width you desire. For example, to get the z -score for an 80% interval, use `-qnorm((1-0.80)/2)`.

There’s been a lot of fuss about this quadratic estimation, but we’re almost done. The final thing is to provide a way to extract the second derivatives—the estimates of curvature—from maximum likelihood results in R. First, I’ll show you how to get `optim` to calculate these derivatives. Second, I’ll show you how to extract the standard error from `mle2`’s coefficient table. Finally, I’ll provide a short piece of code to compute a numerical approximation, worked for our proportion-of-water case study.

2.7.4.5. Getting the curvature from `optim`. While we’ll use `mle2` for most maximum likelihood fits in the book, it’s good to know what is going on under the hood. That means knowing what `optim` does. You already saw how to get a maximum likelihood estimate, using `optim`. To get `optim` to compute the second derivative of the negative log-likelihood, you just have to add `hessian=TRUE` when you invoke it. For example:

```
pw.mle <- optim( 0.5 , fn=function(pw) -dbinom(6,10,pw,TRUE)
                  , hessian=TRUE )
```

R code
2.38

This adds one more element, called `hessian`, to the result list `pw.mle`. You can extract it with:

```
pw.mle$hessian
```

R code
2.39

```
[,1]
[1,] 41.66707
```

This is our old friend, the second derivative of the negative log-likelihood, evaluated at the maximum likelihood estimate. To compute the standard error from this value, just use:

R code
2.40

```
sqrt( 1/pw.mle$hessian )
```

```
[ ,1]
[1, ] 0.1549186
```

The funny `[,1]` and `[1,]` surrounding the result just mean that `hessian` is a *matrix*. We don't need to worry about this right now, but when there is more than one model dimension (parameter), then `hessian` contains as many rows and columns as there are dimensions, because we need curvature estimates for each dimension. We'll work through such a case later in the book. At that time, I'll also explain why this thing is called `hessian`.

2.7.4.6. Getting the curvature from `mle2`. You've already seen that `mle2` will compute the standard error for you, in its `summary` table. You can extract these standard errors directly from the table with:

R code
2.41

```
summary(pw.mle2)$coef[,2]
```

```
[1] 0.1549193
```

The symbol `pw.mle2` is the symbol you stored the maximum likelihood estimate in before. The code above just generates the summary (`summary`), selects the coefficients table from it (`@coef`), and then extracts only the second column of that table (`[,2]`).

What if you want the raw second derivative? Well, you could compute it from the standard error, knowing that $-\partial^2 \log L / \partial p_W^2 = 1/\sigma^2$. It turns out that `mle2` doesn't store a `hessian` with the second derivatives, like `optim` does. Instead, it stores a matrix of the reciprocals of the second derivatives. This makes the diagonal of this matrix variances, σ^2 . You can extract this matrix using:

R code
2.42

```
vcov( pw.mle2 )
```

Later, when we have multi-dimensional models, we'll work with this matrix, a so-called *variance-covariance matrix*, to summarize the posterior distribution. So it's good to know how to access it. Most of the model fitting commands in R support extracting this variance-covariance matrix with the command `vcov`. However, `vcov` does not work with `optim` directly. Don't worry: When we need this matrix, you'll see exactly how to use it. And since you just suffered through all that junk about second derivatives, in principle you could compute this matrix yourself.

2.7.4.7. *Estimating the curvature numerically.* In most cases, what `optim` and `mle2` are doing is using a numerical technique to approximate the second derivatives. They aren't doing analytical calculus. Still, sometimes neither `optim` nor `mle2` can manage to compute estimates of the second derivatives. That doesn't mean all is lost, however. Sometimes you can do the estimate yourself, with a similar numerical approach. Or maybe you just want to know how `optim` is doing it. Here's some code to estimate the second derivative of the log-likelihood numerically, for our proportion of water problem:

```
nw <- 6
pw <- 0.6
n <- 10
delta <- 0.0001
pw2 <- pw + delta
pw3 <- pw2 + delta^2
nll2 <- -dbinom(nw,n,pw2,TRUE)
nll3 <- -dbinom(nw,n,pw3,TRUE)
d1 <- ( nll3 - nll2 )/delta^2
d2 <- (d1 - 0)/delta
sqrt(1/d2)
```

R code
2.43

```
[1] 0.154909
```

This is an estimate of the standard error, and it is quite close to the estimates that we've already seen.

For those interested in how this code works, the second derivative is just the rate of change in the first derivative. All this code is doing is estimating the second derivative by computing the first derivative at two points. As the distance between these two points shrinks, the difference between the first derivatives divided by the distance approaches the second derivative. We already know the first derivative at the maximum likelihood estimate, zero, so all we actually need to do is estimate the first derivative at one other point, called `pw2` above. The first derivative at this point is the difference in $-\log\text{-likelihood}$ between this point and yet a third point, very close to `pw2`, called `pw3`. Then we divide the derivative at this point by the distance between the maximum likelihood and that point to get an estimate of the second derivative. For this to be a good estimate, the distance between the two points used to estimate the first derivative needs to be much smaller than the distance between the maximum likelihood estimate and the point chosen to estimate the first derivative at. That's why the distance between `pw2` and `pw3` above

is `delta`² instead of `delta`. Since `delta` is a very small number less than one, squaring it makes it a very very small number.

In most cases, you can trust R packages like the library `numDeriv` to do a better job of this than you could, with less of your own effort. The function `hessian` in that package is usually all you need. But for some problems, specifying the input to such functions is not always easier than just doing it yourself. Of course it's also nice to know how the magic is done.

At this point, I want to add a note about the distant horizon, Markov Chain Monte Carlo (MCMC) estimation. Later in the book, you'll see how to get samples from the posterior density, naturally including all of the variances and covariances among parameters, but without worrying about any of this quadratic approximation stuff. That will be fantastic, and it's one reason why MCMC is so powerful. The cost, however, is that MCMC takes a lot more computer time than maximum likelihood and quadratic approximation does. And in many cases, the two approaches yield functionally identical inferences. There are many important models, however, for which we can't compute the necessary likelihoods or for which we can't trust the quadratic approximation. In those cases, MCMC saves your bacon. This business of applied statistical computing is all about tradeoffs. You want to know the costs and compromises of each method, so you'll know when to put down one method and pick up another.

2.7.5. Predictive checks. To evaluate the range predictions implied by the naive posterior, we generate a sample of data. Each value of p_W implies its own distribution of predictions, and we want to average over these unique distributions, using the posterior probabilities. This sounds pretty daunting, when spoken of mathematically. But again, once we have “empirical” samples from the posterior, it’s just like summarizing data. First, we use each sampled posterior to simulate sampling. This collection will automatically be averaged appropriately over the posterior. Then we summarize the collection of samples.

There is a random binomial command in R expressly for the purpose of simulating samples from a binomial process. In fact, all of the built-in likelihood functions in R, like `dbinom`, have corresponding random functions that accept parameter values and produce simulated samples of data. Here’s how to use `rbinom`, the random function corresponding to `dbinom`:

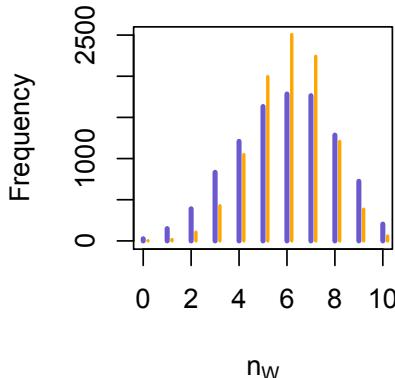


FIGURE 2.12. Simulated data, using both the entire posterior density (blue) and just the maximum likelihood estimate (orange). Model predictions based only on the maximum likelihood can substantially underestimate the uncertainty.

```
models <- seq(from=0,to=1,by=0.01)
post <- dbinom( 6 , size=10 , prob=models )
sim.models <- sample( models , size=10000 , replace=TRUE , prob=post )
sim.data <- rbinom( n=10000 , size=10 , prob=sim.models )
```

R code
2.44

What should you do with these “fake” data now? That depends upon your question, but a typical use might be to visualize the pattern of predictions and generate *prediction intervals*. Prediction intervals are like confidence intervals, but they are ranges of data rather than ranges of models containing a specified proportion. Let’s visualize the simulation data we just computed, before turning to computing prediction intervals in this case. In later chapters, you’ll see that these concepts have huge value in interpreting the meaning of model fits.

To plot the simulated samples, you could use the standard `hist` command in R, but for discrete data of this sort, better to use the `simplehist` command that is part of the code that accompanies this book:

```
simplehist( sim.data )
```

R code
2.45

You can make a similar plot for simulated data produced from only the maximum likelihood estimate:

```
sim.data.mle <- rbinom( n=10000 , size=10 , prob=0.6 )
simplehist( sim.data.mle )
```

R code
2.46

In Figure 2.12, I show both of these plots, stacked together for ease of comparison. The blue bars show the frequencies of different counts of water, n_W , generated by averaging over the uncertainty in the entire naive posterior density, the profile likelihood. These are the values in `sim.data`. The orange bars are the frequencies of counts simulated from only the maximum likelihood estimate. These are the values in `sim.data.mle`. You can see right away that the blue distribution is much flatter than the orange. This is a result of honestly incorporating the uncertainty as the true value of p_W into our predictions. Using only the maximum likelihood model, $p_W = 0.6$, produces a narrower distribution of predictions, bunched up around $n_W = 6$, but this greater confidence is just an illusion of ignoring the uncertainty in the entire posterior. It makes little sense to discard the additional information embodied in the entire posterior distribution, just because some single parameter value has higher probability than all of the others.

You can use the lists of simulated data to produce *prediction intervals*, which are ranges of observations that contain some proportion of all simulated observations. Here's how to compute 95% prediction intervals for the naive posterior simulations and the maximum likelihood simulations:

R code
2.47

```
quantile( sim.data , probs=c(0.025,0.975) )
quantile( sim.data.mle , probs=c(0.025,0.975) )
```

2.5%	97.5%
2	9

2.5%	97.5%
3	9

You could use `HPDI` just as well, in place of `quantile`. As you might expect after looking at Figure 2.12, the prediction interval for the simulations from the full posterior is wider, in this case including $n_W = 2$, whereas the interval for the maximum likelihood model alone ranges only from $n_W = 3$ to $n_W = 9$. In larger samples, there would be more possible observations, and you would be able to see finer differences even at the upper bound of these intervals.

In fact, that's a good exercise for the reader to try: modify the preceding code to simulate data from samples of $n = 50$. What you should discover, when you do this, is that the differences between the posterior simulations and maximum likelihood simulations only grows, and the

prediction intervals will be more different as well. One strength of using the simulation approach is that it becomes possible to ask what the model will predict in a context other than the one we collected the data in. As you'll see in the next chapter, this luxury is not afforded by other approaches to evaluating model predictions, such as the use of P -values.

The distributions in FIGURE 2.12 do provide the first example though of the sort of absolute model check that a P -value can also provide. P -values, as I'll argue at length in Chapter 3, are terrible ways to choose parameter values. But as a way of evaluating whether or not a model provides an adequate fit at all, to some reference data, they are on the right track. The simulated samples in FIGURE 2.12 demonstrate that the maximum likelihood model, as well as the general samples from total naive posterior, fit rather well. You can see this, because the actual observation, $n_W = 6$, is located very centrally in the simulated data. These models have an easy time predicting this observation. Now, in more complicated data problems, this kind of predictive check of model sensibility isn't going to be as transparent. But plotting the posterior's implied predictions is a good first way to always check the sensibility of your models. I will show you how to write code to do this, for each of the major model types in this book.

But in general, there is no single right way to check a model's adequacy. The intuition that one wants to evaluate how well the model predicts key observations remains true in most situations. However, what one defines as a key observation will change, depending upon context, purpose, and intended implementation of the model. For example, suppose we build a fancy model meant to predict hurricane wind speeds. After fitting the model, we evaluate its accuracy within a 7-day window. That is, we feed the model data about historical hurricanes and then look at how well the model's predictions match what actually happened to those hurricanes 7 days later. How good a job does the model do predicting wind speed, 7 days out? If we look only at the average, or median, prediction, it might do quite well. Perhaps the model's predictions are off by less than 1%, on average. But with hurricanes, it isn't the average that worries us, because there is asymmetrical risk from wind speed. As wind speed increases, expected damage rises approximately exponentially. So we should be more concerned with accuracy on the high end of predictions. How good a job does the model do predicting extreme wind events? These are the events that we should focus on, because errors from predicting high winds but actually observing low winds cost society much less, in terms of blood and treasure.

This hurricane example is not typical of many problems one might use statistical models for. But it is meant to drive home the point that the same kinds of questions are needed every time one tries to evaluate the adequacy of a model. Neither the mean nor median nor 95th percentile will always be the right prediction to focus on. There is no getting around thinking, in this business.

2.8. Confidence Intervals Are Not Significance Tests

This chapter has been about one elementary logic of science, probability. We saw how Bayes' theorem logically underlays the common reliance on maximum likelihood. That is, maximum likelihood estimates derive their desirable properties from Bayes' theorem. We also saw how the likelihood of every model (parameter value) is useful in describing inferences from data. It makes little sense to adopt the maximum likelihood estimate and throw away the rest. This is why confidence intervals and other measures of the shape of the posterior are common in both Bayesian and non-Bayesian traditions.

In all of this work, uncertainty is center stage. In order to justify maximum likelihood and to describe the shape of the likelihood function, we had to make approximations. The maximum likelihood estimate is approximately proportional to the posterior, provided evidence overwhelms the prior. Likelihoods are approximate, because our models are incomplete, compared to the complexity of reality. Good models approximate reality. And confidence intervals are further approximations, because the vast majority of published intervals are based upon normal approximations to non-normal distributions.

Despite the omnipresent uncertainty that arises from approximation, statistical models can be developed into highly useful tools. But what does not make much sense in such a context is a dogmatic reliance upon intervals or thresholds for decisions about models. It is common to see scientists reject a hypothesis, because it is not included in a 95% confidence interval. When the hypothesis lies far outside the interval, rejection can be reasonable. But even when the hypothesis lies just outside the interval, many scientists treat the hypothesis the same way. This kind of procedure is often called *significance testing*. This is a bad way to use confidence intervals. It violates the logic of probability and forgets the uncertainty inherent in producing an interval.

Consider for example a 95% confidence interval for p_W of 0.365 – 0.499. This interval does not include $p_W = 0.5$, but just barely. Should we treat $p_W = 0.5$ the same as $p_W = 0.9$ then, rejecting both equally? Of course not. But that is what is often done in practice, when scientists use

confidence intervals to reject models that lie just outside some arbitrary threshold of confidence and likewise accept a model that lies just within it. When we do such a thing, changing any of a dozen details of the model or how it is fit to data or how confidence intervals are estimated will move the interval and suddenly lead to a different conclusion. This cannot be a good way to make inferences from data, if this approach is so sensitive to the exact details of computation.

So what are we supposed to do instead? There are many alternatives, but none are as standard as significance testing, and so all of them will be frowned upon by some reviewers and editors, inflexible and nervous and conformist about statistics. My preference lately is to calculate the sum probability of models on the other side of some interesting inflection point in the model space. For our proportion of water example, this might be $p_W = 0.5$, focusing on the question of whether or not the world is mostly covered in water ($p_W > 0.5$) or not ($p_W < 0.5$). In many cases, the natural inflection point in model space will be where some parameter is equal to zero. Andrew Gelman calls the probability that the parameter lies on the other side of the central estimate the probability of a Type S error, “S” for sign. Such a probability is an estimate of the chances we are mistaking the direction of an effect. I’ll compute some of these Type S error probabilities in later chapters, in order to demonstrate their potential value.

In general though, I don’t want to endorse any one metric of confidence. Instead, consider what you need to help in your own science. Confidence intervals certainly do provide valuable advice for making decisions about hypotheses. But the bridge between probability and decision is a long one, made from considerations of costs and benefits and risk. [Need to explain this Columbus map example in full.] To return to the tale of Cristoforo Colombo, if his map had allowed for a confidence interval on the circumference of the Earth, then how should he have used it to advise his voyage? Should he have planned for a voyage as long as the upper bound of a 95% interval? Why 95% instead of 50% or even 99.9%? After all, we’re talking about the lives of himself and his crew. These are not easy questions to answer, and whatever the answers, they will not be general to all problems of making decisions from inferences about probability.

My own concern in this arena is the possibility that unimagined models are important. For example, Colombo did not consider a map with two very large continents intervening between Europe and Asia. Under that model, planning for the journey should be quite different,

indeed. In other words, everything we've done so far in computing maximum likelihood and confidence intervals are examples of *small world* problems, in which every possible model is known and a posterior probability can be assigned to each. But we don't live in small worlds. We live in *large worlds*, in which important possibilities are generally unimagined.

Using confidence intervals to conduct significance tests is a bad procedure. But the usual estimate used in such tests is not a confidence interval, but rather a *P*-value. Like confidence intervals, the *P*-values used in standard significance testing are based upon approximations and live in small worlds. Unlike confidence intervals, however, *P*-values are not easily understood as approximations of posterior probability. In the next chapter, we continue with this line of reasoning by considering in detail the Tyranny of Fisher, the dogmatic reliance upon null hypothesis significance testing for making decisions about the values of parameters.

3 The Tyranny of Fisher

Suppose you have a number of dresses in your closet. You want to know which one fits you best. There are two basic strategies. You could try each dress on, one at a time, until you find one that seems to fit good enough for you. That is, you try to evaluate each dress relative to some standard of fit. Perhaps you want the hem line to reach no lower than 1 inch below the knee. The other approach would be to try each dress or suit on and select the one that fits best. This is a relative standard, finding the best dress by comparing them.

In statistical inference, these two approaches also exist, but the dresses are models and the fit is predictive accuracy, or some other criterion. Sometimes we compare models to an independent standard. Call this the *absolute evaluation*. Is the model good enough? Can it predict a crucial observation? Other times we compare models to one another. Call this the *relative evaluation*. Which model predicts best? Each approach has its strengths and weaknesses. The relative evaluation can more reliably find the best model in your closet, but even that model might fit badly. The absolute evaluation is good for telling us if we need to go buy a new dress or come up with another model. But it will be less good at finding the best fitting model, when several of them might be good enough. In that case, the order you evaluate the models in will determine which you choose. It might also be hard to determine the absolute standard to judge a model by. When is a model good enough? In contrast, it's a simpler matter to judge fits relative to one another, as you saw in the previous chapter, even though such comparisons tell us little about the absolute evaluations of models.

There is nothing stopping us from using both evaluation methods, of course. We could use relative evaluation to find the best dress in the closet, and then compare that dress to some absolute standard of fit. Indeed, this is the course that many statisticians practice and recommend, substituting dresses for models, of course. And often we don't want to choose any single model, but rather just summarize what the evidence

says about all of them. But much of applied statistics does little relative evaluation. Instead, many natural and social scientists have been trained to decide whether or not to accept or reject a statistical model based only upon a specific absolute evaluation, the P -value, of a single model that usually does not even correspond to a research hypothesis, but rather a *null hypothesis*.

A P -value is a kind of predictive check of a null model, a preferred dress from the closet. The standard of prediction for P -values is almost always that the probability of the observations or more extreme observations be greater than 0.05. If so, then it fits well enough, and usually no other model is checked. Indeed, often no other model is ever fit to the data. There may be models lurking in the closet that make much more accurate predictions than the null model does, but if one uses only this sort of absolute evaluation, one will never know it.

Moreover, as I argued in the first chapter, we should carefully distinguish *inference* and *decision*. The posterior density provides inferential power that can be used to make many different cost-benefit decisions. Much of the time in science, we are not trying to decide in the moment which model is correct. Rather we wish to infer what the evidence says about all of the models. The decision about which model to adopt is a community project. The P -value approach attempts to leap right past the inferential step to the decision step. Of course it must quantify how evidence reflects on a model, but because it uses only one model, it is prone to some odd behavior. And because it forces a decision, it encourages us to believe that we must make a ruling right now, instead of honestly reporting the posterior density to the community of scientists.

In this chapter, we consider in detail the Tyranny of Fisher, the dogmatic reliance upon null hypothesis significance testing for making decisions about models. By *null hypothesis significance testing*, I mean the procedure of computing a P -value for a point hypothesis of no effect or difference and then rejecting this null hypothesis only when $P < 0.05$. By *dogmatic reliance*, I mean the social context in which studies cannot get published and dissertations cannot get signed unless it is shown that $P < 0.05$.

In this chapter, I argue and show the reader how to prove that P -values are neither what most scientists believe them to be nor do they perform well at the jobs put to them. Absolute evaluations have a role to play, but not in this form and never alone. If you are like most scientists, the actual properties of a P -value will surprise you. Odds are that your elementary education in statistics actively misinformed you about them. Exploring and deconstructing the Tyranny of Fisher is partly a

problem in the sociology of science. Prominent statisticians have been trying for decades to correct misunderstandings and misuse of *P*-values, without success. Meanwhile, statistics departments continue to teach a procedure that they largely do not respect but that science departments require their majors to know.

I'm not going to have much new to say about the sociology of the problem, however. Instead, I will direct our attention to practical difficulties with *P*-values. The goal is to demonstrate, and teach the reader the tools needed to prove for themselves, that using *P*-values and typical null hypotheses provide unreliable help in finding good models. None of these arguments are new. But in my experience students and colleagues need to be confronted with them in an uncompromising way, before they will engage with more powerful approaches. At the end of the chapter, I recommend further reading to explore the history of these ideas.

The perspective I present here does reference Bayesian concepts, when they provide clarity. But it's important to realize that one does not have to accept Bayesian norms to accept most of these criticisms of *P*-values. Non-Bayesian statisticians have been just as fiercely critical of null hypothesis significance testing.

3.1. Defining the *P*-value

Before getting into misunderstandings and drawbacks of *P*-values, it is necessary to precisely define them and compare them to the measures in the previous chapter. Let's continue with the proportion of water on the globe example, just for sake of continuity and comparability.

In order to compute a *P*-value, we have to define a *null hypothesis*, which we have not done so far. In principle, the null hypothesis can be anything. But the reader is probably accustomed to the null hypothesis being equivalent to a model of no effect or no difference between groups in the data. The customary role of such a null model is to capture a hypothesis in which some causal effect is absent. So there is a natural attraction to such hypotheses, for many. But the logic of null hypothesis testing survives, no matter what value of a parameter we choose to be the null.

Which null model should we imagine for the proportion of water example? We could focus on the hypothesis that $p_W = 0.5$, but that seems bizarre. Why is half water and half land a hypothesis of no effect or no difference? So to make the example a little more interesting, suppose instead of sampling from our globe now, that we have launched a probe to an exoplanet with unknown proportion of surface water. When the

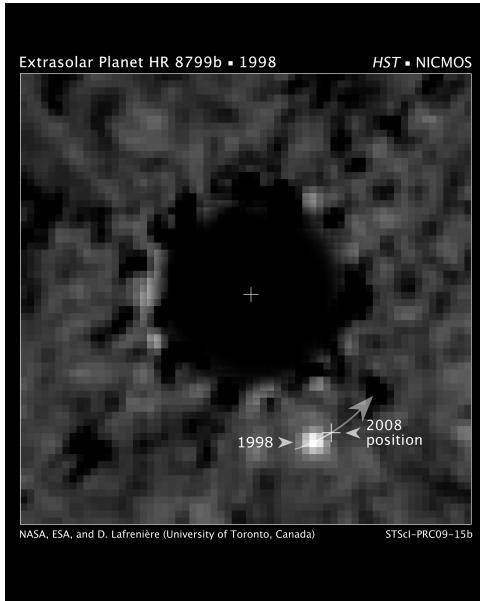


FIGURE 3.1. Our mysterious exoplanet. Image source: <http://hubblesite.org/newscenter/archive/releases/2009/15/>

probe arrives, it places 10 landing probes into essentially random decaying orbits. When each probe lands, it records only whether or not it landed in water and sends that message back to Earth. Since we can't see the exoplanet clearly at this distance, we will assume that the count n_W of the number of probes that said "water" is all the information we have to estimate the proportion of the exoplanet covered in water, p_W .

I don't happen to think that a null hypothesis makes much sense here, but that's because I don't think null hypotheses make much sense in hardly any circumstance. Many students I've taught find the proportion of water covering the Earth as a natural null, however, so let's adopt $p_W = 0.7$ as the null. Now that we have a null hypothesis, we define the P -value as:

P -value: The likelihood of the observed data or any unobserved more extreme data, assuming that the null hypothesis is true.

How do we define "more extreme?" It is defined relative to the null hypothesis, H_0 , and the scale of measurement we adopt. So if what we observe, D , is smaller than what is expected under the null hypothesis, then "more extreme" means all those likelihoods for values of D even smaller. This is a little confusing, so there will be plotted examples shortly.

In the case of our exoplanet problem, suppose half of the probes reported "water" and half reported "land." This means we observe $n_W = 5$, and the P -value in this case becomes:

$$P \equiv \Pr(n_W \leq 5 | p_W = 0.7).$$

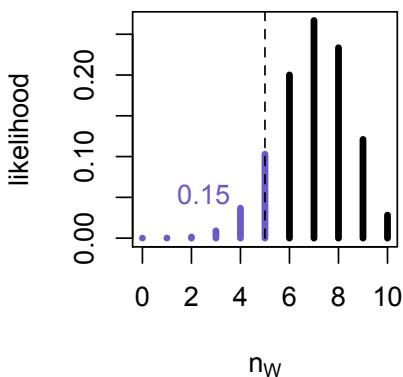


FIGURE 3.2. Computing the P -value when 5 of 10 probes report “water.” Each bar is the likelihood of a possible observed count of “water.” The vertical dashed line shows the location of the actual observation, $n_W = 5$. The blue bars sum to the P -value, $P = 0.15$.

This expression is actually a sum of likelihoods—probabilities of data given a model—and so expands to:

$$P = \sum_{n_W=0}^5 \Pr(n_W | p_W = 0.7).$$

You can easily calculate this in R with code like:

```
p <- sum( dbinom( 0:5 , size=10 , prob=0.7 ) )
```

R code
3.1

This line of code just adds up all the likelihoods for those observations less-than-or-equal-to 5, holding the probability constant at 0.7. If you are not familiar with syntax like `0:5` for making lists, just enter that bit of code alone on the command prompt in R to see that it is just a shortcut for `seq(from=0,to=5,by=1)`.

Figure 3.2 illustrates this sum. On the horizontal axis are the possible observations, from zero reports of “water” to ten reports of “water.” The vertical axis is likelihood, and the height of each bar in the figure is the likelihood of each possible observation, under the model that $p_W = 0.7$. So unlike all of the figures you saw in Chapter 2, here the model is constant and the data are what we are varying. The actual observation is shown by the location of the vertical dashed line at $n_W = 5$. The blue bars are those values of n_W equal to or more extreme than the actual data we observed. When we sum up the heights of these blue bars, we get the value shown in blue in the figure, 0.15. This is the P -value in this case, $P = 0.15$.

What do we do with this number? Following the widespread ritual of null hypothesis significance testing (NHST), we adopt $P \leq 0.05$ as the criterion for rejecting the null hypothesis. Since $P = 0.15 > 0.05$, we conclude that there is not sufficient evidence to reject the null hypothesis and probably even claim that there is no statistical evidence that the proportion of water on the exoplanet in question is different from 0.7. The result is *statistically insignificant*. If the result had instead been that $P \leq 0.05$, we would conclude—still following the ritual—that the result is *statistically significant* and that the proportion of water on the exoplanet is not 0.7.

Let's review the procedure. To conduct a null hypothesis significance test, one must:

- (1) Define a null hypothesis, usually a statistical model equivalent to zero effect or no difference. Specify no other models.
- (2) Compute the likelihood of observing the actual data or any observation more extreme than the actual data, assuming that the null hypothesis is true. Call this sum of likelihoods P .
- (3) If $P \leq 0.05$, conclude that the null hypothesis is false. It has been rejected. The observations are *statistically significant*. If instead $P > 0.05$, conclude that we cannot reject the null hypothesis. The observations are *statistically insignificant*.

There are some elaborations of this ritual, such as power analysis and corrections for multiple comparisons. But this core above remains intact in almost all cases.

3.2. Hopeful Illusions

Judging from surveys of graduate students and professors and even professors who teach statistics, scientists commonly hold to a number of hopeful illusions about the meaning of P -values. Before we dig into these illusions and dispel them, I must stress again, as I did in Chapter 1, that the intention here is not to scold. A majority of scientists have been taught these illusions, either actively in poor instruction or passively in reading the scientific literature. Many textbooks contain incorrect definitions of P -values, so it becomes unfair to blame students and faculty who are not experts in statistical inference. And even when scientists know that these illusions are fallacies, they may still behave as if they were true. So if you find yourself guilty of endorsing any of them, you are in very good company. The important thing is to find the courage and mastery of the material to challenge those who would coerce you into participating in these illusions.

Figure 3.3 shows the results of two surveys, conducted in the years 1986 and 2000 (Oakes 1986, Haller and Krauss 2002). These surveys were of academic psychologists, which is particularly shocking, since psychologists perhaps rely upon P -values more than any other field. This figure shows the percent of different categories of people endorsing different absolutely false statements about the meaning of a significant result of $P = 0.01$ in a simple experimental context. The top group of bars shows the percent in each category endorsing at least one false statement. The other groups of bars correspond to individual false statements. These statements are abbreviated in the figure. They are in full, in order from top to bottom:

- (1) You have a reliable experimental finding in the sense that if, hypothetically, the experiment were repeated a great number of times, you would obtain a significant result on 99% of occasions.
- (2) You know, if you decide to reject the null hypothesis, the probability that you are making the wrong decision.
- (3) You can deduce the probability of the experimental hypothesis being true.
- (4) You have absolutely proved your experimental hypothesis (that there is a difference between the population means).
- (5) You have found the probability of the null hypothesis being true.
- (6) You have absolutely disproved the null hypothesis.

What all of these statements have in common is that they are false and they are hopeful. All of them overestimate rather than underestimate what P -values can tell us (Gigerenzer 2004). In my experience, the general atmosphere of understanding is no better in biology nor anthropology nor economics and has not improved since.

Before exploring the computational details of P -values and the consequences of their use, it might be helpful for many readers to see explanations of why each of these statements is false. Haller and Krauss (2002) provide such explanations, as well as a number of valuable heuristics for teaching about and interpreting such tests. I will provide slightly different explanations of these fallacies here, but readers who can admit to themselves that they believed any one of these would do well to consult Haller and Krauss (2002) and Gigerenzer (2004) and follow citations therein.

- (1) *You have a reliable experimental finding in the sense that if, hypothetically, the experiment were repeated a great number of times, you would obtain*

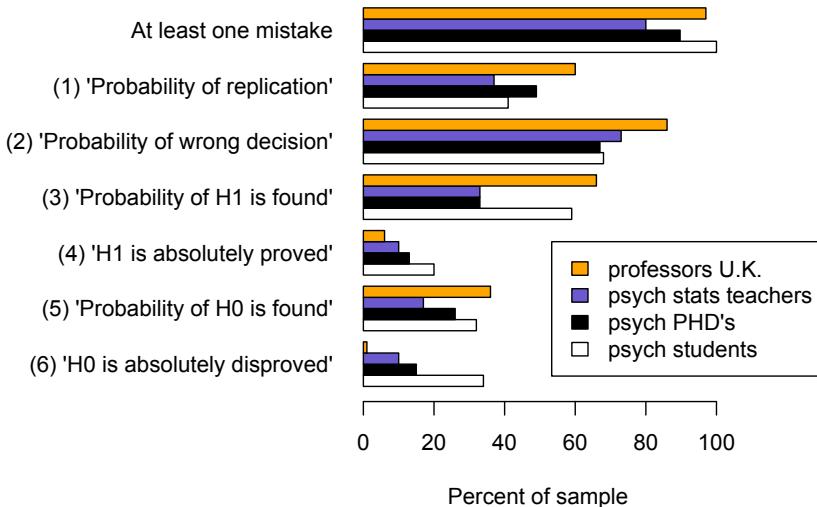


FIGURE 3.3. Two surveys of false beliefs about the meaning of P -values (Oakes 1986, Haller and Krauss 2002). See full text of these false statements in the main text.

a significant result on 99% of occasions. This statement is false foremost because any conclusions derived from a P -value are only valid under the assumption that the null hypothesis is true. If the null hypothesis is not true, then whatever the probability of replicating the results, it isn't provided by the P -value, which only applies to the null hypothesis. There is a deeper confusion usually lurking here, one of conflating Fisherian P -values with the kind of Type I error rates, α , advocated by Neyman and Pearson. These are not equivalent and usually not even compatible approaches to hypothesis testing.

(2) *You know, if you decide to reject the null hypothesis, the probability that you are making the wrong decision.* Again, this is foremost false because any information we derive from the P -value is only valid when the null hypothesis is true. This statement implies that the P -value tells us the probability that the null hypothesis is true, which is instead what we must assume, in order to compute the P -value. As we learned in the previous chapter, $\Pr(D|M)$ is not the same as $\Pr(M|D)$. A P -value is more similar to a likelihood than it is to a posterior probability, and only posterior probability informs us about probabilities of models.

(3) *You can deduce the probability of the experimental hypothesis being true.* No alternative hypothesis factors in the calculation of a P -value, and so P contains no information about the truth value of any hypothesis other than the null. Even then, as I keep saying, P is not the probability that the null is true, because we must assume it is true to compute P .

(4) *You have absolutely proved your experimental hypothesis (that there is a difference between the population means).* There are a couple of key illusions here. The first is that probability can prove anything. Probabilities only corroborate hypotheses, and to pretend otherwise is to indulge in a kind of Tyranny of Popper (see Chapter 1). The second is again that we must assume the null hypothesis is true in order to compute P . As a result, P cannot tell us whether or not either it or any other hypothesis is true.

(5) *You have found the probability of the null hypothesis being true.* Yet again, since we must assume the null is true in order to compute P , it does not tell us if the null is true or not. $\Pr(D|M)$ is not the same as $\Pr(M|D)$.

(6) *You have absolutely disproved the null hypothesis.* Since probabilities cannot provide logical proof in the manner this illusion imagines, it is false. This is a common example of the Tyranny of Popper, manifested through the Tyranny of Fisher.

It is certainly true, however, that a P -value provides some information about the null hypothesis. What is it? Perhaps the easiest way to explain P is to compare it to the naive posterior we explored in the last chapter.

3.3. Comparing P to the Naive Posterior

A puzzling thing about the null hypothesis testing procedure is that no model other than the null model is ever made to predict the observations. It is a method of absolute evaluation. This is quite a change from the previous chapter, in which we used relative evaluation of models. In Chapter 2, we saw that the logic of probability gives us advice on which model (value of a parameter) the evidence supports, when we compute the posterior probability density. Under a uniform prior, the posterior is proportional to the likelihood. I called the resulting posterior a *naive* posterior, one informed only by the likelihood function. The likelihood function is computed (at least in the case of a single parameter family of models) by varying the model and recomputing the likelihood of the observed data. The data were constant, but the model was unknown, so we

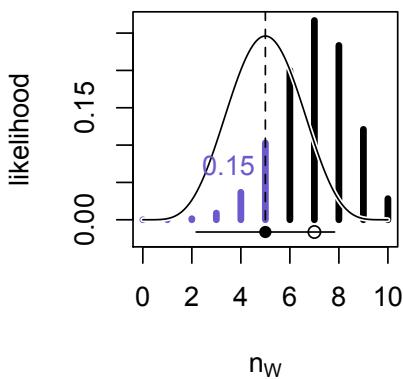


FIGURE 3.4. Computing both the P -value when 5 of 10 probes report “water” (blue bars) as well as the likelihood function for $n_W = 5$ (solid curve). The line underneath is the 95% confidence interval, with the maximum likelihood estimate (filled circle) and null model (open circle).

varied the model to see how models performed relative to one another in predicting the fixed data. Since the P -value is computed holding the model constant, at the null model, and instead varying the data, it’s somewhat the opposite of using the naive posterior.

In FIGURE 3.4, I reproduce the plot from before, but now superimposing the likelihood function and the 95% confidence interval around the maximum likelihood estimate. The sum heights of the blue bars are again the P -value in this instance. The thin curve is the likelihood function, treating the horizontal axis as the expected samples from a family of models with different values of p_W . The horizontal line under the curve represents the 95% confidence interval. The solid circle on this line is the maximum likelihood estimate, $\hat{p}_W = 0.5$ in this case, and the open circle is the null model, $p_W = 0.7$. Notice that the fact that $P > 0.05$ does not also imply that the maximum likelihood estimate is $\hat{p}_W = 0.7$, the null model. Instead, we see that the null model, the open circle underneath, is included in the far right end of the 95% confidence interval, while the maximum likelihood estimate lies at the sample proportion.

When a test of significance fails, it is very common to act as if the best estimate of the parameter is the null model (usually zero, here 0.7). Are we really supposed to conclude here that there is not sufficient evidence to reject the null hypothesis? There are a couple of puzzles that arise immediate from this question. Let’s consider each before turning to clearing up confusion about the meaning of P .

3.3.1. The maximum likelihood estimate always has higher likelihood.

First, the null hypothesis has much lower likelihood than does

the maximum likelihood estimate, $p_W = 0.5$. You can compare the likelihoods of the two models quite easily:

```
lik.mle <- dbinom( 5 , 10 , 0.5 )
lik.H0 <- dbinom( 5 , 10 , 0.7 )
lik.mle/lik.H0
```

R code
3.2

[1] 2.391132

This number is the likelihood ratio of the maximum likelihood to the null model. The maximum likelihood estimate is more than twice as likely as the null, but our NHST procedure still has us failing to reject the null in favor of the maximum likelihood estimate. So obviously, whatever justification we might offer for favoring the null, it isn't simple likelihood alone. We might attempt to patch this up by comparing the likelihood of the null model to the total likelihood of all other models, or maybe just all the models less than the null. But this will just make things worse for the null, because most of the likelihood is bunched up around the maximum likelihood estimate. Let's try two other approaches, both of which will demonstrate concerns with NHST, but also teach some ways to compare approaches to statistical inference.

3.3.1.1. A prior belief in the null model. Another idea is to adopt a prior probability density that favors the null model. One can always find a prior that will lead to the null having higher posterior probability than the maximum likelihood estimate. The simplest way is just use a uniform prior with a spike in probability at the null. If that spike is tall enough, posterior probability will still favor the null model over the maximum likelihood model. In this way, NHST amounts to having an unstated prior preference for the null hypothesis.

Here's what such an approach would look like. I don't happen to think this is a credible way to arrive at the procedures of NHST, but going through this exercise now allows me to accomplish two goals. First, I get to introduce you to computing posterior probabilities using a simple technique called *grid approximation*. Second, this exercise will illustrate how strongly NHST tends to favor the null hypothesis.

First, we need to construct a prior probability distribution over possible values of p_W . So let's make a list of values of p_W in 0.001 increments. By dividing up a continuous variable like p_W in this way, we are doing something often called *grid approximation*, which approximates a continuous distribution with a finite number of discrete values. This is a handy technique for getting used to computing posterior probabilities,

especially for students who have never had a course in integral calculus. To make a list of models, we use our friend the `seq` command:

R code
3.3

```
models <- seq(from=0, to=1, by=0.001)
```

Now we want to start construction a prior probability distribution that is flat everyplace but at the null model. This will do it:

R code
3.4

```
prior <- rep( 1 , length(models) )
h0idx <- min( which( models >= 0.7 ) )
prior[h0idx] <- 10
prior <- prior/sum(prior)
```

The first line just makes a list of 1's of the same length as our list of models. The second and third lines then find the location (index) of the null model and place a 10 at that location in the prior. Finally, we normalize the prior so that it sums to one, because it is a probability density. This gives us a list of probabilities corresponding to each model in `models`. The probability of the null in this list is 10-times greater than that of any other model.

Now recall that Bayes' theorem is:

$$\text{Posterior} = \frac{\text{Likelihood}}{\text{Probability of data}} \times \text{Prior.}$$

In our current problem, this becomes:

$$\Pr(p_W|n_W, n) = \frac{\Pr(n_W, n|p_W)}{\Pr(n_W, n)} \Pr(p_W).$$

We've already constructed $\Pr(p_W)$, by assigning a probability to each unique value of p_W in `models`. Now we need the likelihoods and the probability of the data. After working through Chapter 2, computing the likelihoods is familiar to you:

R code
3.5

```
likelihood <- dbinom( 5 , size=10 , prob=models )
```

To compute the normalization constant, $\Pr(n_W, n)$, the unconditional probability of the observed data, it helps to realize that this probability is an average across models (values of p_W) of the likelihood of n_W, n . So if we multiply each likelihood by each prior probability, we get the average we are after. In code form:

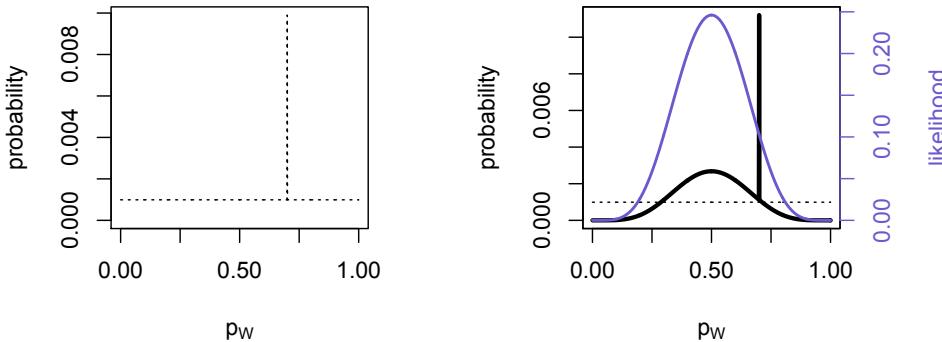


FIGURE 3.5. Prior probability and null hypotheses. On the left, an arbitrary prior probability density that strongly favors the model at $p_W = 0.7$, the null hypothesis. On the right, the posterior (black) still is maximized at $p_W = 0.7$, because the prior so strongly favored the null. With enough sampling, the prior can eventually be overwhelmed. In this way, prior probability of this kind behaves much like null hypothesis significance testing, in that it requires some threshold of evidence before a pre-existing preference for the null hypothesis can be dislodged.

```
PrD <- sum( likelihood * prior )
```

R code
3.6

Finally, we combine all three pieces to compute the list of posterior probabilities of each value of p_W :

```
posterior <- likelihood * prior / PrD
```

R code
3.7

In FIGURE 3.5, I plot the prior we created and the posterior we computed from it. On the left, the plot with the dotted lines shows the prior probability of each value of p_W being very small and equal, but with a spike in probability at $p_W = 0.7$. On the right, the posterior probability in black is distinctly different from the likelihood curve in blue, because of the prior probability. The spike in posterior probability at $p_W = 0.7$ means that the null model still has larger posterior probability than any

other model, even the maximum likelihood estimate. If you play around with the code, you can show that, depending upon how strongly the prior favors the null, with enough sampling the maximum likelihood estimate will eventually surpass the null in posterior probability. But it might take a lot of sampling.

With enough data, any prior can be overwhelmed, as you saw in Chapter 2. So a prior probability like the one we created here will behave similarly to a null hypothesis test. However, this exercise is heuristic. Null hypothesis tests are not derived via any consideration of prior probabilities, except perhaps by rejecting their use. The point to take away here is that NHST acts as if we had a pre-existing preference for the null hypothesis.

3.3.1.2. Is NHST accurate? All that philosophizing is nice, but most of us are more interested in how a method performs. Does NHST work or doesn't it? To decide that, we need a comparison to some other method, as showing that NHST works better than guessing would hardly be surprising. But setting up a contest between NHST and some other approach is simple enough. Let's consider NHST as the choice between the null hypothesis, when $P > 0.05$, and the maximum likelihood estimate, when $P < 0.05$. In practice, that's what most people do, although as I've argued already in summarizing the posterior, it usually doesn't make much sense to discard the posterior in favor of a single model. But directly comparing inference under NHST and adopting maximum likelihood will make this lesson easier to understand.

Suppose we compute the accuracy of NHST and maximum likelihood estimation, across different true values of the parameter p_W , by simulating samples. Define accuracy as the distance of model (value of p_W) we adopt from the true value. Since we simulated the samples, we know the true value in each case, even though our NHST procedure acts as if it doesn't know. Here's the outline of how we might do this.

- (1) Imagine the true value of p_W to be anything between zero and one.
- (2) Under the true value of p_W from (1), generate a large number of simulated samples from 10 probes.
- (3) For each sample from (2), calculate the P -value for the null hypothesis that $p_W = 0.7$ (or any other value).
- (4) If $p < 0.05$, adopt the maximum likelihood estimate as the NHST estimate of p_W . Otherwise, adopt the null model as the NHST estimate of p_W .

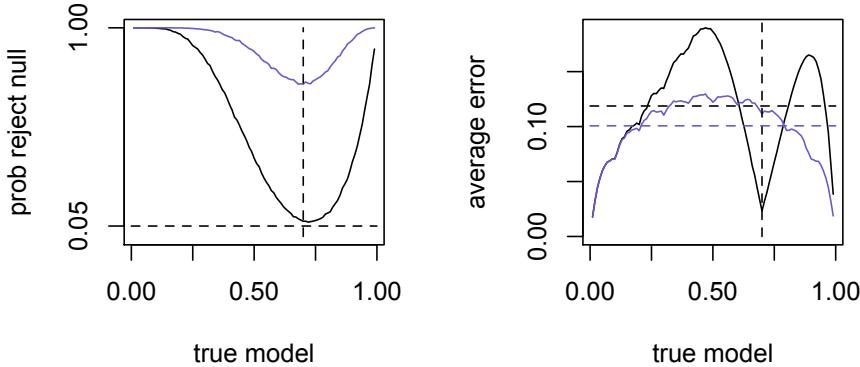


FIGURE 3.6. Behavior and accuracy of null hypothesis significance testing (NHST) compared to the maximum likelihood estimate. In both plots, the black curve represents NHST and the blue curve the maximum likelihood estimate. The vertical dashed line shows the location of the null model, at $p_W = 0.7$. For each possible value of p_W on the horizontal axes, 10-thousand simulated samples of 10 probes were used to compute decisions to accept/reject the null model that $p_W = 0.7$ (NHST) and to compute the maximum likelihood estimate. On the left, NHST adopts the null model much more often than maximum likelihood does. This behavior explains why in the right-hand plot NHST is less accurate, unless the true model is very close to the null model.

- (5) Compute the distance between the true value of p_W from (1) and the value we adopted in (4).
- (6) Go back to (1) until there are no more unique values of p_W to consider as true values.

In FIGURE 3.6, I show the results of doing this in R. The code is a little more involved than usual, so I include it at the very end of this section, rather than right here. The black curves in both plots represent decisions taken under NHST. The blue curves in both plots represent instead the decision of always adopting the maximum likelihood estimate. The horizontal axis in both plots are different true values of p_W used in simulations. The vertical dashed lines mark the location of

the null hypothesis, $p_W = 0.7$. At 0.001 increments, I simulated 10-thousand samples of $n = 10$ at each imagined true value of p_W . For each simulated sample, I calculated the P -value and the maximum likelihood estimate, $\hat{p}_W = n_W/n$.

In the lefthand plot, I show the proportion of simulations in which the null hypothesis was rejected. The horizontal dashed line is the significance threshold of 0.05. When the null is true, NHST is supposed to falsely reject the null model this often. Notice that NHST, in black, accepts the null much more often than adopting the maximum likelihood estimate. And NHST doesn't even reject the true null 5% of the time, as it advertises. It does get pretty close, though, when the true model is near $p_W = 0.7$. Adopting the maximum likelihood estimate, in blue, always "rejects" the null hypothesis, unless the null and the maximum likelihood estimate are the same. In a small sample like this, that is possible, but rare. It happens only when the null is very close to the true value. When the true value of p_W is very far from the null model, NHST usually rejects the null. But in a wide region around the null, NHST accepts the null over the maximum likelihood estimate.

What kind of accuracy does NHST achieve by acting this way? In the righthand plot, I show the average accuracy of each approach. The error on the vertical axis is the distance from the adopted model, whether null or maximum likelihood, to the true value on the horizontal axis. Notice that when the true value of p_W is very close to the null model, then NHST is more accurate than using the maximum likelihood estimate. A little further out, however, and NHST is less accurate than the maximum likelihood estimate. Very far out from the null model, both methods adopt the maximum likelihood estimate and are equally accurate. If we are willing to entertain the notion that all values of p_W on the horizontal are equally probable, in our state of ignorance of the exoplanet, then the average error for NHST in this example is 0.1188 (horizontal dashed line) and that of the maximum likelihood estimate 0.1007 (blue horizontal dashed line). If we make the sample size larger, this relationship does not change. NHST will still always perform worse, averaged across possible true models. And in case you think some other procedure like a likelihood ratio test will do better, likelihood ratio tests actually do worse. They do worse because they are based on normal approximations to binomial probability, in this case.

The comparison between MLE and NHST here is actually handicapping the Bayesian approach. NHST would look even worse, if we used the entire posterior, rather than just the MLE. The MLE averages worse accuracy than the average of the posterior, whenever they differ.

They will differ when the posterior is not symmetrical, as is common for binomial probabilities like these.

The only way to make NHST more accurate on average than naive maximum likelihood is if the null hypothesis, or values near it, are truly more probable in real samples. Then NHST's greater accuracy near the null model would make up for its terrible performance further away. So we see again that NHST is much like adopting some kind of prior probability that favors the null hypothesis.

How reasonable such a preference for the null appears to you will depend upon your view of how science should work. It will also probably depend upon whether or not the null is your pet hypothesis or not. What I think most of us can agree upon, however, is that one should either be explicit about the form and magnitude of this preference—make it into a real prior probability distribution—or rather leave it out of the analysis and include it instead in the discussion of how current results reflect on models. In its usual form, the prior preference NHST endows upon the null model is unconscious and of unmeasured strength. It is leading many scientists to think that logical analysis has accepted the null hypothesis, when in fact some vague preference for the null has accepted it. Checking model predictions is fine, but why check the predictions of the null? Why not check instead the MLE?

Finally, here's the code to compute the values used in FIGURE 3.6. First, we make a convenient function that computes the P -values for the binomial sampling model. This code is functionally equivalent in this context to the built-in R command `binom.test`, but much simpler.

```
calc.p <- function(n=10, nw=1, pw.h0=0.7) {
  if ( nw/n <= pw.h0 ) { p <- sum( dbinom(0:nw, n, pw.h0) ) }
  else { p <- sum( dbinom( n:nw , n , pw.h0 ) ) }
  p
}
```

R code
3.8

Now we need a function that simulates sampling under some true value of p_W and simulates rejecting the null hypothesis. This function returns four different values: (1) the proportion of simulations in which the null was accepted, using NHST; (2) the proportion of simulations in which the maximum likelihood estimate (MLE) and the null coincide; (3) the average absolute distance between the accepted model and the true model, under NHST; (4) the average absolute distance between the maximum likelihood estimate and the true value.

R code
3.9

```
sim.false.accept <- function(n=10,pw.true=0.1,pw.h0=0.7,
  R=9999,alpha=0.05) {
  nw <- rbinom( R , size=n , prob=pw.true )
  p <- sapply( nw , function(z) calc.p(nw=z,n=n,pw.h0=pw.h0) )
  accept.null.nhst <- ifelse( p > alpha , 1 , 0 )
  accept.null.mle <- ifelse( nw/n==pw.h0 , 1 , 0 )
  avg.error.nhst <- mean( abs( accept.null.nhst*(pw.true-pw.h0)
    + (1-accept.null.nhst)*(pw.true-nw/n) ) )
  avg.error.mle <- mean( abs( pw.true - nw/n ) )
  c( mean(accept.null.nhst) , mean(accept.null.mle) ,
    avg.error.nhst , avg.error.mle )
}
```

We'll use this function to repeat such simulations across all possible true values of p_w . The valuable trick here is using `sapply` (Simplified Apply), which is a command that allows us to pass each value in a list to a function, storing the individual results in a new list. For example, to square each element of a list of numbers `x`, you could just enter `x^2` in R. But suppose you are working with a function that doesn't recognize lists. In that case, you want to use `sapply` or a similar command. You could do it like this:

R code
3.10

```
x <- 1:10
x.squares <- sapply( x , function(z) z^2 )
```

That `z` inside of `sapply` is just an arbitrary name. You could call it `donut` if you want to. It just takes on values from the list of numbers being fed to it, one at a time. Here's the code that applies our `models` to `sim.false.accept`:

R code
3.11

```
models <- seq(from=0.01,to=0.99,by=0.01)
sims <- sapply( models , function(z)
  sim.false.accept( pw.true=z , n=10 , pw.h0=0.7 ) )
```

The use of `sapply` above takes each value in `models`, hands it to `sim.false.accept`, and then stores the four-part result in the matrix `sims`. So we end up here with a matrix of results, in which each column is an individual call to `sim.false.accept` and each row is one of the elements returned by `sim.false.accept`. Now you can use standard plotting commands like `plot(1-sims[,] ~ models)` to replicate the figure. The logic of

sapply can be awkward at first. But there will be other examples. So be patient with yourself and this trick will make sense eventually.

3.3.2. Why predict what did not happen? A second puzzle is that it's not clear why the likelihoods of events that didn't happen should matter. Every blue bar to the left of $n_W = 5$ is the likelihood of an event that we have not observed. This isn't just about judging the fit of a dress by an absolute standard—the standard of fit here imagines ways that the dress could fit worse than it actually does and uses those imaginary flaws to decide how well the dress does fit. Statisticians have long worried about this issue, especially Bayesian statisticians. Here is what Harold Jeffreys had to say about P -values in 1939, in his book *Theory of Probability* (page 315):

What the use of P implies, therefore, is that a hypothesis that may be true may be rejected because it has not predicted observable results that have not occurred. This seems a remarkable procedure. On the face of it the fact that such results have not occurred might more reasonably be taken as evidence for the law, not against it.

This would be like convicting an innocent man because we have not found his prints on the murder weapon. To return to the dress metaphor, this is like deciding a dress does not fit well enough, because it could fit worse.

In practice, however, the usual trouble with including these unobserved events in P is that they tend to overstate the evidence for the null hypothesis. So we might say more often that the use of P implies that a hypothesis that may be wrong may be accepted because it predicts observable results that have not occurred. This would be like acquitting a guilty man because you found his prints on the murder weapon. In this section, we'll examine a consequence of the fact that P is defined as a sum across imagined data rather than across models: The magnitude of P depends upon how sampling works, even when the parameter of interest doesn't affect sampling in any way. The major problem arises when sample size is not fixed but instead varies randomly or contingently upon what events occur.

Suppose our probe to the exoplanet carries 20 surface probes, each capable of landing in a different random location on the exoplanet. However, there is a chance that any surface probe burns up on entry into the exoplanet atmosphere, and as a result, only 10 of them ever report back to the mother probe. Suppose still that the observed data are

5 “water” in 10 reports, for a sample proportion of 0.5. If we could replicate this “experiment,” the sample size would not always be $n = 10$, but would instead vary randomly according to the probability of a probe’s being destroyed on entry. That is, sometimes all 20 would report back, sometimes only 3 would report back. This means we now have to consider variable sample size.

While comparing the maximum likelihood estimate and the null hypothesis based upon their respective likelihoods remains unaffected by the potential for variable sample sizes, NHST is dramatically altered by it. In practice, computer software always assumes constant sample size, but as I’ll argue at the end of this section, only in the most disciplined experimental sciences is such an assumption reasonable.

Let d be the probability any surface probe burns up on entry. Then the likelihood of observing n_W reports of “water” from n surface probes that actually manage to land, out of 20 probes that were launched at the exoplanet, is:

$$\Pr(n_W, n|p_W, d) = \Pr(n|d, 20) \Pr(n_W|p_W, n). \quad (3.1)$$

This is just to say that the likelihood is the chance of both n probes landing successfully and receiving n_W reports of “water.” $\Pr(n|d, 20)$ is the probability of achieving a sample size of n , conditional on d and 20 attempts. $\Pr(n_W|p_W, n)$ is just the same old binomial likelihood we’ve been working with since Chapter 2. So what exactly is the distribution $\Pr(n|d, 20)$? This too is just binomial, with 20 attempts and n successes, each success having probability $1-d$, the chance a surface probe is not destroyed during entry. We’re assuming that the number of probes that land successfully does not influence where they land, and so the probability on the left is just the product of the probabilities on the right. Note that the general conclusion we are about to arrive at does not depend upon this assumption of independence between landing location and landing successfully. The assumption of independence is just to make the example easier to work through.

Suppose the chance of a probe being destroyed is about 2%, $d = 0.02$. Now let’s produce some code to make R compute P -values for our landing probes data. First, we need the likelihood function above, and just multiplying two calls to `dbinom` will do the trick. To compute the likelihood of observing 5 “water” when 10 of 20 surface probes survive, using the null hypothesis that $p_W = 0.7$:

R code
3.12

```
dbinom(10,size=20,prob=1-0.02) * dbinom(5,size=10,prob=0.7)
```

```
[1] 0.0840926
```

The first call to `dbinom` computes the probability of 10 out of 20 surface probes surviving. The second call computes the probability of getting 5 reports of “water” from 10 probes, with a 0.7 probability of “water” from each. We’re going to want to use this code again and again, to sum up likelihoods of all the unobserved events that contribute to the *P*-value, so let’s package it up into an R *function*. A function is just a bundle of R commands that we give a name, so we can use it again and again. Entering the following into R will define the function `dprobe`:

```
dprobe <- function( nw , n , nmax , d , pw ) {
  prn <- dbinom( n , size=nmax , prob=1-d )
  prnw <- dbinom( nw , size=n , prob=pw )
  prn * prnw
}
```

R code
3.13

This function computes the likelihood of observing `nw` reports of “water” from `n` probes, when `nmax` probes were launched, each with probability `d` of being destroyed on entry, and finally where each surviving probe has a chance `pw` of landing in water. Once you have defined the function in R, it works just like every other R command. For example, to compute the likelihood of the observed data under the null hypothesis:

```
dprobe( nw=5 , n=10 , nmax=20 , d=0.02 , pw=0.7 )
```

R code
3.14

```
[1] 1.590949e-13
```

Readers who didn’t grow up coding like I did may reasonably wonder what that `e-13` means. That is how computers typically format scientific notation. The `e-13` just means $\times 10^{-13}$.

The *P*-value for the observation $n_w = 5, n = 10$ is the sum of all likelihoods for all possible events with sample proportion more extreme than the observed. So that means we want to add up the likelihoods of all possible combinations of n_w and n where $n_w/n \leq 0.5$. There are some slick compact ways to compute such a thing, but for the sake of clarity, I’ll just use some loops. Here’s a function to compute the *P*-value:

```
pval.probe <- function( d ) {
  p <- 0
  for ( n in 1:20 ) {
    for ( nw in 0:n ) {
      if ( nw/n <= 0.5 )
```

R code
3.15

```

    p <- p + dprobe(nw=nw, n=n, nmax=20, d=d, pw=0.7)
} # nw
} # n
p
}

```

This function loops over all possible values of n , the number of surviving surface probes, from 1 to 20. We don't consider cases in which zero probes survive, because in that case, the sample proportion n_w/n is undefined—no information means no estimate. For each value of n , the function then loops over each value of n_w from zero to n . These are all the possible counts of reports of “water.” For each of these, it adds the likelihood under the null hypothesis, $p_W = 0.7$, to the growing P -value, stored in the symbol p .

Now we want to examine how changes in d affect the P -value. So we make a list of values of d and compute P at each:

R code
3.16

```

d.list <- seq(from=0,to=1,by=0.01)
p.list <- sapply( d.list , function(z) pval.probe(d=z) )

```

FIGURE 3.7 shows these P -values as a function of d . The solid curve is the P -value and the horizontal dashed line is just the conventional significance level of $P = 0.05$ for reference. Notice that p increases drastically as d increases, up to very high probabilities of destruction, at which point we begin to get no information from the probes. The blue line shows the likelihood ratio of the maximum likelihood estimate, $\hat{p}_W = n_w/n$, to the null model, $p_W = 0.7$. You can calculate this ratio as a function of d in a similar way:

R code
3.17

```

Lr.list <- sapply( d.list , function(z)
  dprobe(5,n=10,nmax=nmax,d=z,pw=5/10) /
  dprobe(5,n=10,nmax=nmax,d=z,pw=0.7) )

```

This ratio is insensitive to d . It has the same value, regardless of how probable it is for a probe to be destroyed during entry. Why is this?

We can use the analytical likelihood function for this model, Expression 3.1 (page 122), to compute the likelihood ratio between the maximum likelihood estimate and the null model:

$$\frac{\Pr(n_w, n | \hat{p}_W, d)}{\Pr(n_w, n | 0.7, d)} = \frac{\Pr(n|d, 20) \Pr(n_w | \hat{p}_W, n)}{\Pr(n|d, 20) \Pr(n_w | 0.7, n)} = \frac{\Pr(n_w | \hat{p}_W, n)}{\Pr(n_w | 0.7, n)}.$$

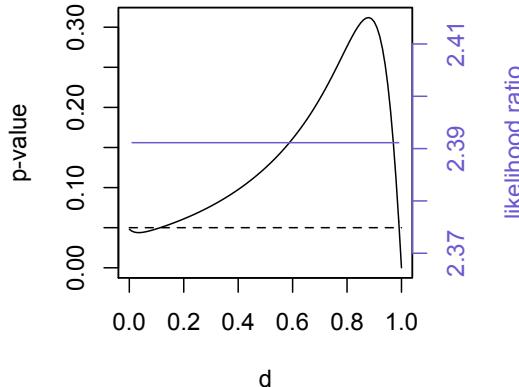


FIGURE 3.7. Comparing P and the likelihood ratio when sample size varies. The solid curve is the P -value of observing $n_W = 5, n = 10$, as a function of d . The dashed line is conventional significance, $P = 0.05$. The blue line is the likelihood ratio of the maximum likelihood estimate, $\hat{p}_W = n_W/n = 0.5$, to the null model, $p_W = 0.7$. While the P -value is dramatically influenced by how the data are sampled, the likelihood ratio is invariant to it.

The probability of destruction d has no influence on the likelihood ratio, because whatever the likelihood $\Pr(n|d, 20)$, it is the same on both the top and bottom, and so it just divides out.

What is going on here? The reason that the P -value is so dramatically altered by the way the sample size is determined is because P is a sum of events that have not been observed. As a result, the likelihood of these unobserved events can have big effects on the magnitude of P . The more likely it is for a probe to be destroyed during entry, the more ways there are to get a sample proportion equal-to-or-more-extreme than the observed proportion. Most of the time, the probabilities of these multiplying unobserved events contribute to P and make it harder and harder to reject the null model. But notice that initial increases in d actually reduce the P -value. Either way, the dependency of P on our assumptions about sampling is a logical consequence of how the P -value is defined. In contrast, the likelihood ratio is undisturbed by variation in sampling. The likelihood ratio is unaffected by the nature of sampling, because the

parameter p_W has no role in determining sample size. Likelihood ratios vary models and compare their ability to predict the sample we actually got. NHST, however, does not vary the model, p_W , but instead imagines new data. This partly bases the decision about which value of p_W to favor upon probabilities that p_W does not influence, $\Pr(n|d, 20)$.

One way to try and rescue P -values here is to note that it would make sense then to define the P -value as the probability of data-or-more-extreme data only when sample size is imagined to be constant across replications. In practice, this is what everyone (everyone's computer) implicitly assumes anyway. If we go this route, however, there are other problems. Sample size is rarely fixed in the real world, and usually we don't even know how it is determined. Many social psychologists, for example, just run experiments for a fixed duration and take however many participants they can get in that time frame. When we analyze data collected by someone else for another purpose, it's not clear what we should believe about sample size across replications. And while everyone knows it's forbidden, sometimes scientists collect data only until they achieve statistical significance. In all of these cases, if you want to actually interpret P -values according to their definition, as a probability of data-or-more-extreme-data, then imagining that sample size is fixed means that the probability you compute will be wrong. It will not have the long-run properties you want it to have. In the wicked case of collecting data until statistical significance, then $P = 1$, by definition, after correctly accounting for the sampling model.

In summary, anytime sample size can vary, then computing a P -value requires taking account of the distribution of sample sizes. Just comparing models (parameter values) using likelihoods (naive posterior probability) does not suffer from this problem, however. Almost no one ever worries about this in practice, but the issue is always there.

My own belief, however, is that this logical snag with P -values isn't the most important reason to reject their use. It certainly is hard to defend the use of events that did not happen when deciding upon the value of a parameter. But even if we could justify always computing them under constant sample size assumptions, it wouldn't make them perform better than maximum likelihood, unless the null hypothesis really were more probable or somehow less costly to adopt than another model. This turns our attention to other issues.

3.4. The need to develop the alternative model

My principle concern with the use of *P*-values, and indeed any other method of absolute evaluation used in isolation, is that they discourage scientists from developing research models. When all one has to do in order to support a research hypothesis is to reject the null model of “randomness,” then there is little incentive to develop the research hypothesis into a mature predictive model. It is possible that null models do not make substantially different predictions than the implied alternatives. But this fact may never be discovered, unless we actually force both (or all) models to predict the observations. As a consequence, scientists sometimes accept the null hypothesis for no reason other than it was the only model they fit to the data. They conclude the alternative is worse, but they never fit the alternative to the data, so the claim is counterproductive and actively retards the progress of knowledge.

It’s hard to force the proportion of water problem into an example of this problem. The possible models that we’ve learned so far are too simple to exhibit the difficulties. Later in the book, we’ll examine in detail an infamous case of this problem, the use of “neutral” models in evolutionary biology. There’s nothing wrong with neutral models, as long as they have company. Rather, the sort of data commonly used to test them is not very capable of differentiating them from non-neutral models. Many scientists never realized this, because they did not actually specify any selection model. They just fit the neutral model and it was good enough by an arbitrary standard, so they rejected selection as a candidate force. The plague of naive neutral modeling spread rapidly, to the point that a few papers actually conducted the same exercise of the distribution of baby names in North America, concluding that the choice of baby names is random, despite decades of work in sociology to the contrary. But since no non-neutral model of name choice was fit to the data, the possibility that the data could not detect non-neutrality, even if it is there, never came up.

As usual, I have a hard time holding the researchers responsible in these cases. They’ve been the victims of bad scientific norms. They were taught by prestigious PhD programs to test theories by constructing only models in which the forces of interest are not present. The norms are the villains here. In defiance of those norms, we must insist that research hypotheses be specified and explicitly compared to any null or neutral models. In doing so, any scientist will immediately realize when the data cannot distinguish the models. They will also realize that there are many ways to make a non-neutral model, and so rejecting the neutral model does little to tell us about nature.⁴²

3.5. How to Interpret P -Values, If You Must

Sadly, journals are full of P -values and will be for some years to come. Norms are changing, such that it is harder to find papers that do not report at least estimates and standard errors, with P -values tacked on the end. In those cases, you can just ignore the P 's and use the estimates and standard errors to construct the approximate naive posterior. But it is still common enough to see publications in highly-ranked journals that report only estimates and P -values or the statistics P -values are derived from, like t -values. There is still a lot of valuable science in these papers, if we can cut through the vagueness of the significance tests. Here is some advice on extracting as much information as you can from these papers.

3.5.1. Confusing effect size and sample size. One difficulty in interpreting P -values is that they are an unclear mix of information about *effect size*—how far the maximum likelihood estimate is from the null model—and *precision*—how confident we can be in the maximum likelihood estimate. It's not possible to know from a P -value alone whether we have accepted/rejected the null model because the effect is small/large or rather because the precision is low/high. The distinction is crucial. Let's consider some examples.

Suppose for the sake of illustration that there are actually four different exoplanets and we have sent probes to each of them. The results of null hypothesis tests for each are displayed in the table below. In each case, the null model is $p_W = 0.7$.

Exoplanet	P -value
Fomalhaut b	0.15
HR 8799c	0.048*
83 LeonisBb	0.39
51 Pegasi b	0.044*

I display an asterisk next to each significant result, as is customary for the results of NHST. Now what can you conclude from this information? Which exoplanets are likely to have water coverage similar to the Earth? It is true that HR 8799c and 51 Pegasi b are significantly different from the Earth, but without knowledge of how many probes reported back in each case, it's impossible to know whether this is due to estimates very different from $n_W/n = 0.7$ or rather few surface probes surviving entry and managing to report back a reading. For example, 51 Pegasi b could be statistically significant either because the sample proportion of water reported back by the surface probes was very small, relative to the null model, or rather because the sample proportion was close to

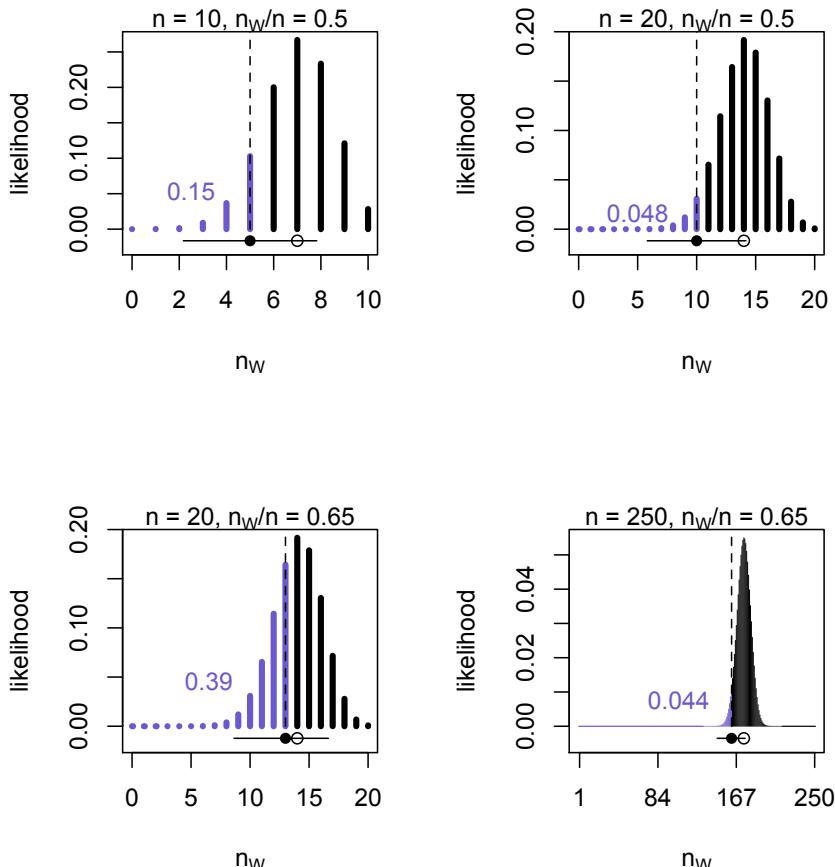


FIGURE 3.8. The ambiguity of P -values with respect to effect size and precision. See explanation in main text.

0.7 but many probes reported back, producing a precise estimate. The P -value alone cannot tell us which of these two things happened. This point is widely appreciated. What is less well appreciated is that, once we have the information necessarily to resolve ambiguity in the P -value, the P -value provides no additional information of value.

It is easier to appreciate how little the table above actually tells us if we fill in the examples with sample size and precision information. In Figure 3.8, I show all four null hypothesis tests, resulting from our

probe reconnaissance of each exoplanet. Each plot shows possible observations along the horizontal axis and likelihoods under the null hypothesis on the vertical axis. The blue bars are included in summing the P -value, which is shown in blue. The observation in each case is shown by the location of the vertical dashed line. The filled circle under each set of likelihoods is the maximum likelihood estimate, and the open circle is the null model. The horizontal line under the likelihoods is the 95% confidence interval. Now we can distinguish large effects from precise small effects. In the upper-left, 10 surface probes managed to report back, and half of them said “water.” This result is statistically insignificant, even though the sample proportion is identical to the plot in the upper-right, in which 20 surface probes survived. The results in the upper-right are statistically significant, with exactly the same effect size, because they are more precise. In the lower-left, again 20 surface probes survive to report back, but this time 13 of them, a proportion 0.65, reported “water.” This is a very small effect size, due to how close 0.65 is to the null model. As a result, this result is statistically insignificant. But in the lower-right, the exact same effect size from a stunning 250 successful probes (perhaps this mission was launched later than the others and so had a larger payload of surface probes to deploy) leads to a statistically significant result, even though this exoplanet we must conclude has a coverage of water very similar to the Earth.

Again, the point of these examples is that small P -values alone can result from either estimates that differ greatly from the null model or from the good precision of small estimates. To resolve this ambiguity, we need to know the estimate and the sample size, with which we could describe the precision by a confidence interval (as in Figure 3.8) or just by plotting a likelihood function. Let’s augment our previous table with maximum likelihood estimates and 95% confidence intervals, to see how much more information this gives us.

Exoplanet	P -value	\hat{p}_W	95% confidence interval
Fomalhaut b	0.15	0.50	0.22–0.78
HR 8799c	0.048*	0.50	0.29–0.71
83 LeonisBb	0.39	0.65	0.43–0.83
51 Pegasi b	0.044*	0.65	0.59–0.71

Now we can more easily see that the evidence supports stronger inferences about the proportions of water on HR 8799c and 51 Pegasi b than for the other two exoplanets. Notice also that in every case, the 95% confidence interval includes the null model, $p_W = 0.7$. This should drive home the point that inferences from the naive posterior are not

the same as inferences from the P -value. In special cases, statistical significance is attained exactly when the null model falls outside the 95% confidence interval. But in general, this is not true. And since we already saw that one can move the P -value around greatly just by changing the sampling model, there may be very little correspondence at all, once the P -value is correctly computed. At the end of Chapter 2, I already argued that conducting dogmatic tests of statistical significance with confidence intervals abuses the nature of those intervals. Here I emphasize also that tests using confidence intervals give different answers than tests using P -values. We have now not only an arbitrary threshold at 5%, but now also more than one arbitrary way to compute that 5%, and these two methods disagree.

So let's ask again: What do we learn from the P -value? First, if we know sample size, we can with some difficulty infer effect size from P . The P -value is strongly correlated with sample size, because for any given effect size, P declines as sample size increases. So it is possible to compute, for a given P -value and sample size, the value of the maximum likelihood estimate. But it would be much easier to just report effect size and precision. And once you know the effect size and precision, all that knowing P adds is information about how strong your prior preference for the null hypothesis is.

Second, if we know effect size, then with some difficulty we can estimate precision from P . For a given estimate, it is possible to compute the sample size at which the estimate will become statistically significant. Unless the estimate is exactly the same as the null model, with enough data, the effect will always become significant. For very small effect sizes, it will take a huge amount of data. For very large effects, it will take only a few samples. Again, it would be much easier on everyone—scientist and readers alike—to just report the estimate and its precision to begin with.

While I still encounter papers that report nothing more than P -values and some vague measure of effect size like an F value, most people are aware that P -values alone are not sufficient for interpreting statistical models. But many people still base decisions about research hypotheses upon the value of P . They are using P to choose parameter values, to choose among models. And since computing P doesn't actually compare models, the procedure is an absolute evaluation where we need a relative one. Certainly one could invent other criteria that lead us to prefer the null model, but those criteria will not be how well the model predicts the data, compared to the other models.

So again we see that appeals to P -values are like appeals to prior belief in the null model. When $P > 0.05$, the evidence at hand is not strong enough to overcome the prior. When $P < 0.05$, it is. In neither case is the prior belief specified with any precision, so it's difficult to follow the chain of inference that leads to a concern with P .

I'm pretty down on classical P -values, as the reader can see. But I did say at the beginning of this chapter that I reserve an important role for checking individual models against data in an absolute evaluation. It's just that P -values, because they actively reject or ignore relative evaluation, and because they are sums of events that we have not observed, are not the form of absolute evaluation that we seek. Or at least, there is no reason to think that the same threshold of predictive quality, and the same measure of it, is the right standard to adopt for all models, for all types of data, in all fields, whatever the purpose of our statistics. Indulge me for a couple more difficulties that arise from reliance on P -values, to support that point.

3.5.2. Statistical significance is not itself significant. Here's a common scenario. A researcher has two variables that each might help predict an outcome of interest. A regression (coming in detail in the next chapter) is fit that includes both of the variables. The researcher's software produces a P -value to correspond to each variable, and only one of them is significant. At this point, it is very common to conclude that only the variable with a significant outcome helps predict the outcome variable.

The logical fallacy in this kind of reasoning is the assumption that the difference in significance is itself significant.⁴³ It might be, but it might also not be. This is not a criticism that relies upon rejecting NHST, as your author does. Instead, it is an inconsistency in how scientists interpret tests of significance. If we adopt a uniform criterion of significance and compute P -values as is standard, then it does not necessarily follow that a difference in statistical significance implies that the difference between two estimates would itself be significant.

This is confusing, without examples. Unfortunately, since we haven't introduced a way to include more than one parameter (model dimension) in our models yet, we're not ready to analyze examples of this issue. So let's use an abstract example that goes straight to estimates and the standard errors of those estimates (Figure 3.9). The curves in this figure are naive posteriors, so they show the maximum likelihood estimates and uncertainty for three different parameters. To take a common example from the scientific literature, suppose these estimates refer to rates of recovery from different drugs, labeled "1" and "2." The effect of

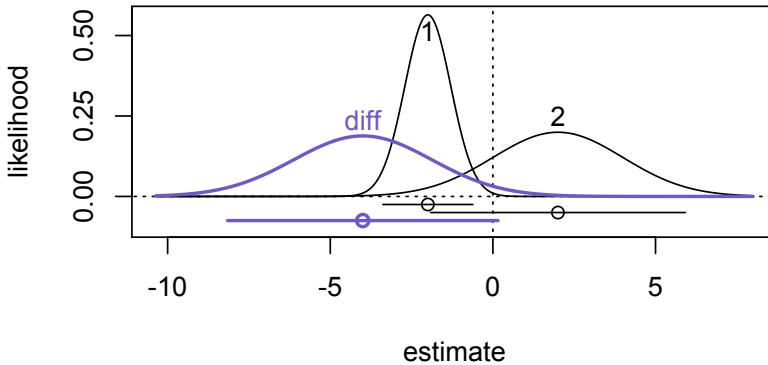


FIGURE 3.9. A difference in statistical significance is not always statistically significant. In this figure, the naive posteriors for three different estimates are shown, as well as corresponding 95% confidence intervals. The estimates labeled “1” and “2” differ in their statistical significance: “1” is significantly different from zero, while “2” is not. The naive posterior of their difference, labeled “diff” and shown in blue, is however not significantly different from zero, meaning that by the same standard of statistical significance, one cannot conclude that “1” and “2” are not the same.

the first drug is negative and significantly different from zero, if we use the confidence interval to conduct a significance test—you know I am not endorsing this procedure, but it is very common. The effect of the second drug is positive, but more variable, and so this drug’s effect is not significantly different from zero. In such a situation, it is routine to see scholars concluding that the drug with the significant result is less effective than the one with the insignificant result. This does not follow. You can see this clearly, if you now inspect the blue curve, which is the naive posterior of the difference between the two black curves. The maximum likelihood difference is indeed negative, but the variance of this density is very large, and so the difference is not significantly different from zero, by the NHST based upon the confidence interval. The same logic applies however you compute the P -value, but using confidence intervals in this case makes the lesson more transparent.

The key reason that differences in significance are not always significant is that such null hypothesis tests do not compute P -values for the model that the two drugs—or two planets, or whatever—are the same. To do that, we'd need to structure the statistical model differently, so that it includes a parameter (a model dimension) corresponding to the *difference* between the two drugs. In the next chapter, we'll see how to do that, but even then, it won't be very useful to test the null hypothesis of zero difference, for all the reasons we've already encountered in this chapter. But it will help you understand why concluding that two estimates are significantly different from one another, just because one or both is significantly different from the null model, is illogical.

If you doubt that scientists commit this fallacy, start looking for it. Nieuwenhuis et al (2011) surveyed the neuroscience literature and found that about half of the papers published contained this fallacy. Whenever you see a claim that some variable has an effect and some other does not, because the first is significant but the other is not, then you are seeing an example of this fallacy. Now, it might be true that the difference between the effect sizes of the two variables is indeed significant, but it does not follow from testing if each effect is different from zero (or any other null model). In a later chapter, you will see how the use of *interactions* can address this problem. But even then, it won't make sense to focus on significance. Instead, the interest is in estimating the difference between two categories in the data. Estimating the response in each category is different than estimating the difference.

A closely related fallacy is to use the overlap of confidence intervals of different estimates to conclude whether they are significantly different from one another. Figure 3.10 illustrates this scenario. Again we see the estimated effects of two drugs, labeled “1” and “2,” and their difference, in blue. I have drawn this example such that neither drug is significantly different from zero, and their confidence intervals overlap considerably. Nevertheless, the naive posterior of their difference is indeed significantly different from zero, by the same standard. Never use overlap or lack of overlap in confidence intervals to make inferences about the posterior density of a difference. Just estimate the difference, to be sure.

I encourage the reader to experiment with examples of this kind. You can use the `show.naive.posterior` function from the code LIBRARY that accompanies this book to make these plots yourself. The code to produce Figure 3.10 is:

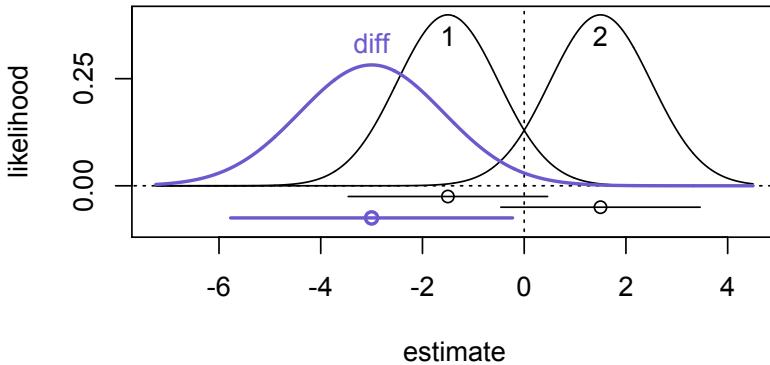


FIGURE 3.10. A difference between estimates may be significantly different, even when the confidence intervals of the estimates overlap.

```
estimates <- c( -1.5 , 1.5 )
variances <- c( 1 , 1 )
ests <- c( estimates , estimates[1]-estimates[2] )
ses <- c( sqrt(variances) , sqrt( sum(variances) ) )
show.naive.posterior( ests , ses , label=c(1,2,"diff") ,
    lwidths=c(1,1,2) , cols=c("black","black","slateblue") )
```

R code
3.18

Just change the values in the `estimates` and `variances` symbols to change the location and precision of “1” and “2.” The difference will be computed by the third and fourth lines. The last line plots the estimates.

3.6. Defenses of *P*-values

Published defenses and rebuttals thereof.

That weird BBS article. That paper in PRSB. The econ defense. Is there an example in psychology, other than the BBS article?

What about Mayo? Not really a defense of NHST, but seems like it much of the time. Best paper is probably the regression assumptions paper: Mayo and Spanos 2004. Agree that we need more than posterior probability to evaluate model assumptions and absolute performance. This segues to next section.

3.7. Checking assumptions

Brief comments on other ways to provide absolute evaluations, aimed at critiquing model structure. Basic message is that no universal method exists, because models are meant for different purposes.

Jaynes on testing assumptions with significance tests (Jaynes 1985 page 351):

It would be very nice to have a formal apparatus that gives us some optimal way of recognizing unusual phenomena and inventing new classes of hypotheses that are most likely to contain the true one; but this remains an art for the creative human mind. In trying to practice this art, the Bayesian has the advantage because his formal apparatus already developed gives him a clearer picture of what to expect; and therefore a sharper perception for recognizing the unexpected.

Segue from there to model building and estimation in next chapter.

4 Building Models

This chapter introduces the most common approach to building multivariate statistical models, the linear model approach. First, the normal or Gaussian distribution is built up as a attractive model of error, because of its commonality in nature and its informational constraints. The parameters of the normal distribution are then made into simple *linear models* that allow for complex multivariate patterns to be inferred from evidence. Common problems in fitting and interpreting linear models provide a foundation for fitting and interpreting non-linear models later in the book. After this chapter, the reader will be able to fit and interpret simple linear regressions. In the next chapter, these skills will be leveraged in the comparison of structurally different linear models. Then in the next chapter after that, we turn to modeling interactions, situations in which the effect of a variable depends upon the values of other variables.

4.1. A Normal World

Suppose you and a thousand of your closest friends line up on the half-way line of a soccer field (football pitch), facing the side of the field. Each of you has a coin in your hand. At the sound of the whistle, you begin flipping the coins. Each time a coin comes up heads, that person moves one step left. Each time a coin comes up tails, that person moves one step right. Each of you flips the coin 16 times and then stands still. Now we measure the distance of each person from the half-way line. Can you predict what proportion of the one-thousand people who are standing on the half-way line? How about the proportion 5 yards left of the line?

Using the binomial distribution, as you used it in previous chapters, you could. But only if each person always took a step of the same size. To make this puzzle more interesting, assume instead that each step is different from all the others, a random distance between zero and one yard. Thus a coin is flipped, a distance between zero and one yard is

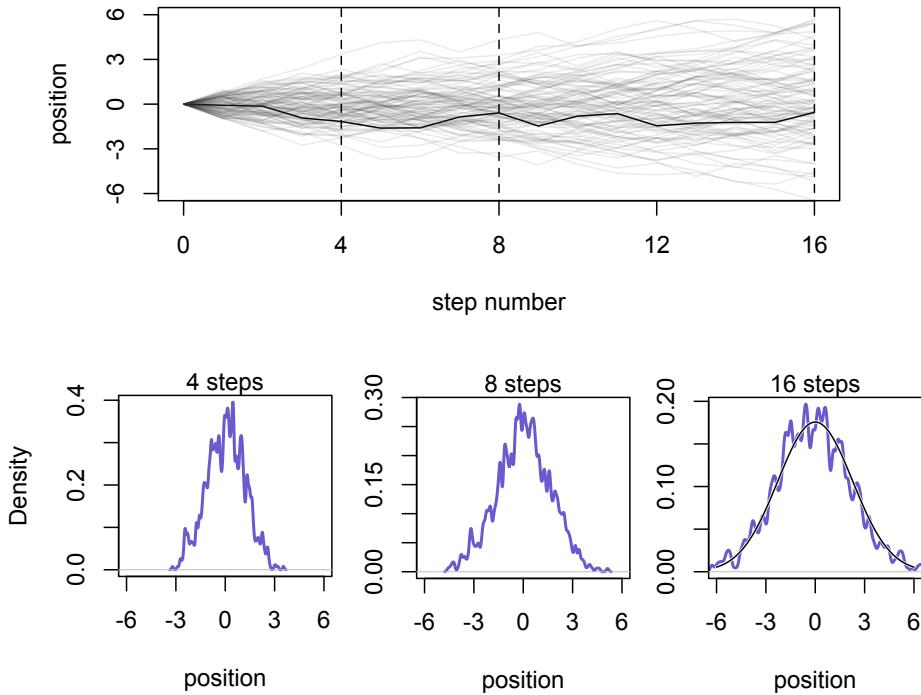


FIGURE 4.1. Random walks on the soccer field converge to a normal distribution. The more steps are taken, the closer the match between the real empirical distribution of positions and the ideal normal distribution, superimposed in the last plot in the bottom panel.

taken in the indicated direction, and the process repeats. Do you know where people come to stand?

It's hard to say where any individual person will end up, but you can say with great confidence what the collection of positions will be. The distances will be distributed in approximately normal, or Gaussian, fashion. Let's prove it, by simulating this experiment in R. What we want to do is generate, for each person, a list of 16 random numbers between -1 and 1 . These are the individual steps. Then we add these steps together to get the position after 16 steps. Then we need to replicate this procedure one-thousand times. This is the sort of task that would be

harrowing in a point-and-click interface, but it is made trivial by the command line. Here's a single line to do the whole thing:

```
pos <- replicate( 1000 , sum( runif(16,-1,1) ) )
```

R code
4.1

You can plot the distribution of final positions in a number of different ways, including `hist(pos)` and `plot(density(pos))`. In Figure 4.1, I show the result of these random walks and how their distribution evolves as the number of steps increases. The top panel plots one-hundred different, independent random walks, with one highlighted in dark. The vertical dashes indicate the locations corresponding to the distribution plots underneath, measured after 4, 8 and 16 steps. Although the distribution of positions starts off seemingly idiosyncratic, after 16 steps, it has already taken on a familiar outline. The familiar “bell” curve of the normal distribution is emerging from the randomness.

Go ahead and experiment with even larger numbers of steps to verify for yourself that the distribution of positions is stabilizing on the normal. You can get convenient comparisons to a normal by using `dens(pos, norm.comp=TRUE)`. The `dens` function is part of the code that accompanies this book. You may also want to play around with the boundaries of the random step sizes. You can square the step sizes and transform them in any number of arbitrary ways, without changing the result: normality emerges. Where does it come from?

We can be sure, before anyone in this experiment takes a single step, that the distribution of positions will stabilize eventually (in this case, very quickly) on the normal distribution. The reason we can be so sure is that any process in nature that adds together random values from the same distribution converges to a normal. It doesn't matter what shape the underlying distribution—the distribution of step sizes, in analogy with the soccer field example—possesses. It could be uniform, like in our example above, or it could be anything else. Depending upon the underlying distribution, this convergence might be slow, but it will be inevitable. Often, as in this example, convergence is rapid. This result is often cited as the *central limit theorem*. It is an amazing truth about how deviations aggregate. I am consistently amazed that this mathematical truth applies to the real world, but centuries of experiment confirm that it does. Here are a few more examples.

Suppose the growth rate of an organism is influenced by a dozen loci, each with several alleles that code for more growth. Suppose also that all of these loci interact with one another, such that each increase growth by a percentage. This means that their effects multiply, rather than add.

For example, we can sample a random growth rate for this example with this line of code:

R code
4.2

```
prod( 1 + runif(12,0,0.1) )
```

This code just samples 12 random numbers between 1.0 and 1.1, each representing a proportional increase in growth. Thus 1.0 means no additional growth and 1.1 means a 10% increase. The product of all twelve is computed and returned as output.

Now what distribution do you think these random products will take? Let's generate 10-thousand of them and see:

R code
4.3

```
growth <- replicate( 10000 , prod( 1 + runif(12,0,0.1) ) )
dens( growth , norm.comp=TRUE )
```

The reader should execute this code in R and see that the distribution is approximately normal again. I said normal distributions arise from summing random deviations, which is true. But the effect at each locus was multiplied by the effects at all the others, not added. So what's going on here?

We again get convergence towards a normal distribution, because the effect at each locus is quite small. Multiplying small numbers is approximately the same as addition. For example, if there are two loci with alleles increasing growth by 10% each, the product is:

$$1.1 \times 1.1 = 1.21.$$

We could also approximate this product by just adding the increases, and be off by only 0.01:

$$1.1 \times 1.1 = (1 + 0.1)(1 + 0.1) = 1 + 0.2 + 0.01 \approx 1.2.$$

The smaller the effect of each locus, the better this additive approximation will be. In this way, small effects that multiply together are approximately additive, and so they also tend to stabilize on normal distributions. Verify this for yourself by comparing:

R code
4.4

```
big <- replicate( 10000 , prod( 1 + runif(12,0,0.5) ) )
small <- replicate( 10000 , prod( 1 + runif(12,0,0.01) ) )
```

The interacting growth deviations, as long as they are sufficiently small, converge to a normal distribution. In this way, the range of causal forces that tend towards normal distributions extends well beyond purely additive interactions.

But wait, there's more. The reader might also know about log-normal distributions. Large deviates that are multiplied together do not produce normal distributions, but they do tend to produce normal distributions on the log scale. For example:

```
log.big <- replicate( 10000 , log(prod(1 + runif(12,0,0.5))))
```

R code
4.5

Yet another normal distribution. We get the normal distribution back, because adding logs is equivalent to multiplying the original numbers. So even multiplicative interactions of large deviations can produce normal distributions, once we measure the outcomes on the log scale. Since measurement scales are arbitrary, there's nothing suspicious about this transformation. After all, it's natural to measure sound and earthquakes and even information (Chapter 5) on a log scale.

The world is full of normal distributions, at least approximately. As a mathematical idealization, we're never going to experience a perfect normal distribution. But it is a widespread pattern, appearing again and again at different scales and in different domains. Measurement errors, variations in growth, and the positions of molecules all tend towards Gaussian distributions. There are many other patterns in nature, so make no mistake in assuming that the normal pattern is universal. In later chapters, we'll see how other useful and common patterns, like the exponential and the gamma and poisson, arise from natural processes. The normal is a member of a family of fundamental natural distributions known as the *exponential family*. All of the members of this family are important for working science, because they populate our world. As a result, we will build statistical models around these distributions.

The normal distribution, as a very common emergent distribution in nature, is a good focus for model building. We're going to spend the rest of this chapter using the Gaussian distribution as a skeleton for our hypotheses, building up models of measurements as aggregations of normal distributions. But the natural occurrence of the normal distribution is only one reason to build models around it. Another route to justifying the normal as our choice of skeleton, and a route that will help us appreciate later why it is often a poor choice, is that it represents a particular state of ignorance. When all we know or are willing to say about a distribution of measures—measures are continuous values on the real number line—is their mean and variance, then the normal distribution arises naturally as the least *surprising* distribution. That is to say that the

normal distribution is the most natural expression of our state of ignorance, because it is the unique distribution for any given mean and finite variance that maximizes *information entropy*.

Indeed, the very reason that normal distributions arise so often in nature is because addition discards all information other than mean (location) and variance. It isn't that other aspects of the distribution don't exist—the underlying distributions can have any kind of skew or kurtosis or higher moment that we wish. But once values produced by such a distribution are added together a sufficient number of times, all the information about those higher moments is bled away, leaving only information about mean and variance. As such, if we adopt the normal as a description of the pattern of a set of measures, this will typically be a conservative description that errs on the side of honest ignorance.

If you haven't studied information theory before, then this talk of "information" and "entropy" is likely puzzling. That's fine and normal (pun intended). We'll dwell a little on information theory in the next chapter, when it'll be more useful. For now, let's take the causal and informational justifications of the normal distribution as reasons to start building models of measures around it. Throughout all of this modeling, keep in mind that using a model is not equivalent to swearing an oath to it. There is no contract that forces us to believe the assumptions of our models, as all models are always false. They just need to be useful.

4.2. From Parameters to Linear Models

This book adopts a unified language for describing and coding statistical models. For each list of values we'd like to predict, we define a distribution that the individual values arise from. Then we relate the shape of that distribution—it's precise location and variance and other aspects of its shape, if it has them—to other variables, those that we wish to use to predict the original variable. This surely isn't the only way to describe statistical modeling, but it is a widespread and productive language. Once a scientist learns this language, it becomes easy to communicate the assumptions of our models. We no longer have to remember seemingly arbitrary lists of bizarre conditions like "homoscedasticity" (constant variance), because we can just read these conditions from the model definitions. We will also be able to see natural ways to change these assumptions, instead of feeling trapped within some procrustean model type like regression or multiple regression or ANOVA or ANCOVA or such. These are all the same kind of model, and that fact becomes obvious once we know how to talk about models as mappings

of one set of variables through a probability distribution onto another set of variables.

4.2.1. Re-describing the water model. It's good to work with examples. Recall the proportion of water problem from previous chapters. The model in that case was always:

$$n_W \sim \text{Binomial}(n, p_W),$$

where n_W was the observed count of water points, n was the total number of points, and p_W was the proportion of water. Read the above statement as:

The count n_W is distributed binomially with sample size n and probability p_W .

Once we know the model in this way, we automatically know all of its assumptions. We know the binomial distribution assumes that each sample (coin flip) is independent of the others, and so we also know that the model assumes that samples points are independent of one another. That assumption may be a poor one, in relation to reality, but the model definition above contains it.

You're going to see models expressed in this way all throughout the rest of this book. This notation is standard in most statistics books these days, so you're learning a common language for describing statistical models. Most books don't pause to teach you how to read these statements—they assume you had a math-stats course at some point, perhaps. This book goes pretty slow with them, however.

For now, we'll focus on simple models like the above, which have only one such *stochastic node*, which will always contain a normal distribution. A stochastic node is just a mapping of a variable onto a distribution. It is *stochastic* because no single instance of the variable on the left is known with certainty. Instead, the mapping is probabilistic: some values are more likely than others, but very many different values are possible under any model. It is a *node* because these statements can form networks of causal relations, and so there can be more than one node, each potentially linking to others. One of the powerful aspects of this language of building models with stochastic nodes is that it allows us to easily define models in which the variables that define the shape of one distribution are themselves products of another distribution. We just use more than one node. I'll provide examples in later chapters, when it will make more sense.

4.2.2. The Gaussian model. The binomial model above is simple enough, but it turns out that interpreting even simple *non-linear* models like this

is much more subtle a task than interpreting *linear* models based upon normal distributions. Why the binomial is non-linear and a Gaussian model can be linear is a topic best left for a later chapter, when we return to binomial models, armed with concepts developed from inspecting the easier Gaussian models.

Let's build a linear Gaussian, or normal, model now. For the rest of this chapter and the next two chapters, we'll work with Gaussian models with only one stochastic node. In case this sounds restrictive to you right now, keep in mind that an easy majority of all statistical models that have ever been fit to data are examples of single-node Gaussian models. This isn't to argue that we don't need other models—in later chapters, I make something of a fuss about the importance of denying the Tyranny of Gauss. But Gauss isn't our enemy, unless we let him blind us to other distributions.

We'll work through this material by using real sets of data. In this case, we want a single measurement variable to model as a normal distribution. Our goal will be to compare different normal distributions and find the mean and standard deviation corresponding to the normal distribution that maximizes posterior probability of the data. As before, we'll do this under the assumption of naiveté—we have no *a priori* reason to believe that any values of the parameters are more likely than any other. And so we'll use a weak prior argument and fall back on maximum likelihood estimation, the naive posterior.

4.2.2.1. The data. The data contained in `data(Howell1)` are partial census data for the Dobe area !Kung San, compiled from interviews conducted by Nancy Howell in the late 1960's.⁴⁴ For the non-anthropologists reading along, the !Kung San are the most famous foraging population of the 20th century, largely because of detailed quantitative studies by people like Howell.

Load the data and place them into a convenient object with:

R code
4.6

```
library(rethinking)
data(Howell1)
d <- Howell1
```

What you have now is a *data frame* named simply `d`. I use the name `d` over and over again in this book to refer to the data frame we are working with at the moment. I keep its name short to save you typing. A *data frame* is a special kind of object in R. It is a table with named columns, corresponding to variables, and numbered rows, corresponding to individual cases. In this example, the cases are individuals. Inspect

the structure of the data frame, the same way you'd inspect the structure of any symbol in R:

```
str( d )
```

R code
4.7

```
'data.frame': 545 obs. of  4 variables:
 $ height: num  152 140 137 157 145 ...
 $ weight: num  47.8 36.5 31.9 53 41.3 ...
 $ age    : num  63 63 65 41 51 35 32 27 19 54 ...
 $ male   : int  1 0 0 1 0 1 0 1 0 1 ...
```

This data frame contains six columns. Each column has 545 entries, so there are 545 individuals in these data. Each individual has a recording height (centimeters), weight (kilograms), age (years), and gender. This is only a partial census. Later in the book (Chapter 7), I'll ask you to work with the full census, containing 846 individuals.

We're going to work only with the `height` column, for the moment. As this chapter progresses, I'll show you how to include the other columns in the statistical model. The column containing the heights is really just a regular old R `vector`, the kind of list we have been working with in many of the code examples. You can access this vector by using its name:

```
d$height
```

R code
4.8

Read the symbol `$` as *extract*, as in *extract the column named `height` from the data frame `d`*. All it does is give you the column that follows it.

It might seem like this whole data frame thing is kind of annoying, right now. If we're working with only one column here, why bother with this `d` thing at all? You don't have to use a data frame, as you can just pass raw vectors to every command we'll use in this book. But keeping related variables in the same data frame is a huge convenience. Once we have more than one variable, and we wish to model one as a function of the others, you'll better see the value of the data frame. You won't have to wait long.

All we want for now are heights of adults in the sample. The reason to filter out non-adults for now is that height is strongly correlated with age, before adulthood. Later in the chapter, I'll ask you to tackle the age problem. But for now, better to postpone it. You can filter the data frame down to individuals of age 18 or greater with:

```
d2 <- d[ d$age >= 18 , ]
```

R code
4.9

We'll be working with the data frame `d2` now. It should have 352 rows (individuals) in it.

The notation used in the code above is *index* notation. It is very powerful, but also quite compact and confusing. The data frame `d` is a matrix, a rectangular grid of values. You can access any value in the matrix with `d[row, col]`, replacing `row` and `col` with row and column numbers. If `row` or `col` are lists of numbers, then you get more than one row or column. If you leave the spot for `row` or `col` blank, like in `d[,]`, then you get all of the rows/columns. Typing `d[,]` just gives you the entire matrix, because it returns all rows and all columns.

So what `d[d$age >= 18 ,]` does is give you all of the rows in which `d$age` is greater-than-or-equal-to 18. It also gives you all of the columns, because the spot after the comma is blank. The result is stored in `d2`, the new data frame containing only adults.

4.2.2.2. The model. Our goal is to model these values using a normal distribution. First, go ahead and plot the distribution of heights, with `dens(d2$height)`. These data look rather normal in shape, as is typical of height data. This may be because height is a sum of many small growth factors. As you saw at the start of the chapter, a distribution of sums tends to converge to a Gaussian distribution. Whatever the reason, adult heights are nearly always approximately normal.

So it's reasonable for the moment to adopt the stance that our stochastic node should be Gaussian. But be careful about choosing the Gaussian distribution only when the plotted outcome variable looks Gaussian to you. Gawking at the raw data, to try to decide how to model them, is often not a good idea, because the data could be a mixture of different normal distributions, for example, and in that case you won't be able to detect the underlying normality just by eyeballing the density plot. Sadly, there is no substitute for thinking in this business.

So the heights are approximately normally distributed, but which normal distribution? There are an infinite number of them, with an infinite number of different means and variances. We're ready to write down the general model and seek the unique parameters that maximize the likelihood. To define the heights as normally distributed with a mean μ and standard deviation σ , we write:

$$h_i \sim \text{Normal}(\mu, \sigma).$$

In many books you'll see the same model written as $h_i \sim \mathcal{N}(\mu, \sigma)$, which means the same thing. The symbol h refers to the list of heights, and the subscript i means *each individual element of this list*. It is conventional to use i because it stands for *index*. The index i takes on row numbers,

and so in this example can take any value from 1 to 352 (the number of heights in `d2$height`). As such, the model above is saying that each height measurement is identically distributed, drawn from the same normal distribution, with a constant mean μ and standard deviation σ . Before long, those little i 's are going to show up on the righthand side of the model definition, and you'll be able to see why we must bother with them.

4.2.2.3. Brute force mapping of the naive posterior. Since this is the first Gaussian model family in the book, and indeed the first model space with more than one parameter, it's worth quickly mapping out the naive posterior through brute force likelihood calculations. This isn't the approach I encourage in many other places, because it is laborious and computationally expensive as your models become more complex. Indeed, it is usually so impractical as to be essentially impossible. But as always, I believe it is worth knowing what the target actually looks like, before you start accepting approximations of it. Later in the chapter, you'll use maximum likelihood and quadratic approximation to estimate the naive posterior density. That approach is what most published statistical analyses amount to. Once you have the samples you'll produce in this subsection, you can compare them to the maximum likelihood search approach in the next.

Here's the general strategy. First, we're going to define a grid of parameter values for both μ and σ to map out the naive posterior across. Then for every combination of unique parameter values in this grid, we compute the likelihood, $\Pr(D|\mu, \sigma)$. These values define the relative height of the naive posterior, at the point defined by a combination of μ and σ . Then I'll show you how to sample from this grid-approximated naive posterior.

First, define the grid to map out the naive posterior over. The ranges chosen for μ and σ need to be broad enough to capture most of the probability mass in the posterior, so in practice, I had to search around a bit to get this to work right. Within these ranges, the intervals between evaluated values of each parameter need to be narrow enough to get a good estimate of the shape of the posterior. But if they are too narrow, the calculations will take forever and provide little additional resolution. I can save you the some effort by just asserting the useful ranges and increment sizes in the form of two lists:

```
mu.list <- seq( from=153, to=157 , length.out=100 )
sigma.list <- seq( from=6.5 , to=9 , length.out=100 )
```

R code
4.10

This code just says to make two lists, each 100 elements long, with equally spaced values defined by the ranges in the `from` and `to` parameters.

The next step is really just a programming trick. What you need now is a way to compute the likelihood at all combinations of one value from `mu.list` and one value from `sigma.list`. There are a couple of ways to do this, but a very useful one that is useful in many diverse circumstances is to use `expand.grid`:

R code
4.11

```
post <- expand.grid( mu=mu.list , sigma=sigma.list )
```

If you inspect the resulting object, `post`, you'll see that it is a data frame with 10-thousand rows. The first column is values of μ and the second is values of σ . Each row is a unique combination. Now all you have to do is iterate over rows and compute the likelihood for each combination.

The first obstacle however is programming the likelihood calculation. You can use the built-in function `dnorm` to do the hard part, evaluate the likelihood formula for a normal distribution. But there are two other important considerations. First, you want the joint likelihood of all of the heights at once. Second, it will be much better to always do likelihood calculations on the log scale, as I emphasized in Chapter 2. Otherwise rounding error will ruin you. The right way to do this is to just add together the log-likelihood of each height in `d$height`. Remember, adding logarithms is the same as multiplying numbers on the original (probability) scale. This code will do it:

R code
4.12

```
post$nLL <- sapply( 1:nrow(post) , function(z)
  -sum( dnorm( d2$height , mean=post$mu[z] , sd=post$sigma[z] ,
    log=TRUE ) ) )
```

Take a look inside the table `post` again, and you'll see that there is a third column now, `nLL`, that holds the negative log-likelihood for each unique combination of parameter values.

It's helpful to pause right here, before taking samples from the posterior, to just visualize the negative log-likelihood landscape you just calculated. If you are using the LIBRARY of code that accompanies this book, you can easily make a contour plot from the columns of `post` with this code:

R code
4.13

```
contour.xyz( post$mu , post$sigma , post$nLL )
```

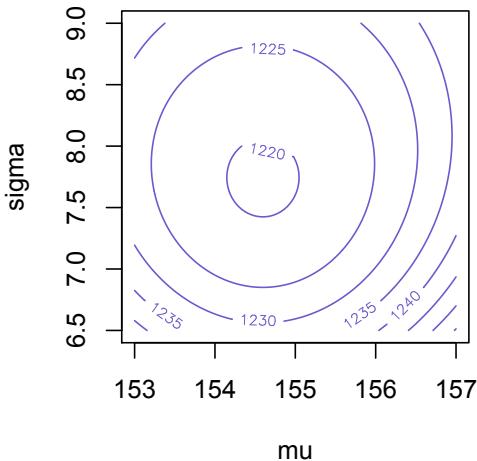


FIGURE 4.2. Contour plot of the negative log-likelihood function of the height data. The likelihood is maximized where this surface has a minimum, inside the circle near the center of the plot. The numbers on each contour indicate negative log-likelihood values. So for example, every point on the ring labeled “1220” has that same negative log-likelihood.

I reproduce this plot in FIGURE 4.2. The axes in this plot are the parameters μ and σ . The contours inside the plot represent different levels of equal negative log-likelihood. So these are like the contour lines on a topographic map. Each contour is labeled with its negative log-likelihood. Notice that the rings at the center have smaller negative log-likelihoods. What you are starting down into is a valley of negative log-likelihood. Recall from Chapter 2 that minimizing negative log-likelihood is equivalent to maximizing likelihood. And so the maximum likelihood estimate lies right at the bottom of this valley, in the middle of FIGURE 4.2.

What you see in FIGURE 4.2 is a log-transformed naive posterior. You can convert it back to probability scale, if you are careful, by just exponentiating. This line of code will work:

```
post$prob <- exp( -post$nLL + min(post$nLL) )
```

R code
4.14

Now there is yet another column in the table `post`, this time with relative likelihoods, or unstandardized posterior probabilities. The trick in the above code is that rounding error is always a threat when moving from log-likelihood to likelihood. If you use the obvious approach, like `exp(-post$nLL)`, you’ll get a vector full of zeros, which isn’t very helpful. This is a product of R’s rounding very small probabilities to zero. Remember, in large samples, all unique samples are unlikely. This is why you have to work with log-likelihoods. The code in the

box above dodges this problem by scaling all of the likelihoods by the maximum likelihood. As a result, the values in `post$prob` are not all zero, but they also aren't exactly probabilities. Instead they are relative posterior probabilities. But that's good enough for what we wish to do with these values.

To study this naive posterior in more detail, again I'll push the flexible approach of sampling parameter values from it. This works just like it did in Chapter 2, when you sampled values of p_W from the naive posterior for the proportion of water example. The only new trick is that since there are two parameters, and we want to sample combinations of them, we first randomly sample row numbers in `post` in proportion to the values in `post$prob`. Then we pull out the parameter values on those randomly sampled rows. This code will do it:

R code
4.15

```
sample.rows <- sample( 1:nrow(post) , size=10000 , replace=TRUE ,
                      prob=post$prob )
sample.mu <- post$mu[ sample.rows ]
sample.sigma <- post$sigma[ sample.rows ]
```

You end up with 10-thousand samples, with replacement, from the naive posterior for the height data. Take a look at these samples:

R code
4.16

```
plot( sample.mu , sample.sigma , cex=0.5 , pch=16 ,
      col=col.alpha("slateblue",0.1) )
```

I reproduce this plot in FIGURE 4.3. Note that the function `col.alpha` is part of the LIBRARY for this book. All it does is make colors transparent, which makes the plot in FIGURE 4.3 more easily show density, where samples overlap. Where FIGURE 4.2 is a valley with the maximum likelihood estimate (MLE) at the bottom, the plot in this case is a hill with the MLE at the peak.

Now that you have these samples, you can describe the distribution of confidence in each combination of μ and σ by summarizing the samples. Think of them like data and describe them. There are two common tasks, once you have samples from a posterior like this. The first is to characterize the shape of the naive posterior of each parameter, μ or σ , averaging over the other parameter(s). The second is to consider whether or not the parameters are correlated with one another, which will turn out to be a useful measure for answering many different questions to come later in the book. After I show you how to do these two things, I'll repeat the entire procedure, but with a subset of only 20 heights from the original 352. This will drive home something

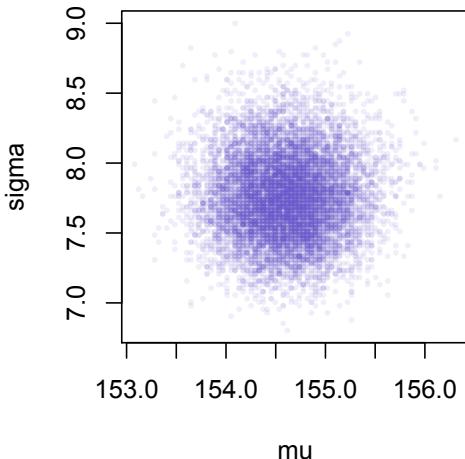


FIGURE 4.3. Samples from the naive posterior density for the heights data.

important about the shape of the naive posterior. After that, I promise to move on to the practical and useful maximum likelihood approach to this problem. As you'll see, as long as the sample size is large enough, the maximum likelihood and quadratic approximation approach provides nearly identical inferences to this exhaustive (exhausting?) full posterior approach. But you have to be mindful of sample size, especially for the variance parameter σ , as you'll see.

Okay, now for the two tasks in summarizing the joint posterior density of μ and σ . First, characterize the shapes of the posterior densities of μ and σ . That is, what does the profile of each parameter's posterior density look like? This simple question is actually hard to answer, if you attempt to calculate it directly from the likelihood formula. This is because for every value of, say, μ , you have to average across uncertainty in the other parameter, σ , in order to say how probable that value of μ is throughout the entire posterior. This is what it means to *integrate out* a parameter. If your integral calculus is fluent, it's not hard, but most scientists just don't have enough experience with integrals to do this confidently. But since you have samples from the posterior, all you have to do is compute the histogram, or density, of the samples. These will automatically correctly average over the uncertainty in the other parameter(s), because your sampling used the correct posterior probabilities. All of the hard work has been done for you already, albeit at the cost of only an approximate answer based on random numbers rather

than an exact one based on formal analysis.⁴⁵ So to plot the densities for μ and σ separately:

R code
4.17

```
dens( sample.mu , adj=.5 , col="slateblue" )
dens( sample.sigma , adj=.5 , col="slateblue" )
```

These densities are very close to being normal distributions. And this is quite typical. As sample size increases, naive posterior densities approach the normal distribution, in obeisance of the powerful central limit theorem. If you look closely, though, you'll notice that the density for σ has a longer righthand tail. I'll exaggerate this tendency a bit later, to show you that this condition is very common for variance parameters.

To summarize the widths of these densities with highest posterior density intervals (HPDI's), just like in Chapter 2:

R code
4.18

```
HPDI( sample.mu , prob=0.95 )
HPDI( sample.sigma , prob=0.95 )
```

Since these samples are just vectors of numbers, you can compute any statistic from them that you could from ordinary data. If you want the mean or median, just use the corresponding R functions. If you want the mode, there is a function in the LIBRARY for this book that computes this from a vector of samples:

R code
4.19

```
chainmode( sample.mu )
```

[1] 154.6656

Why this function has “chain” in its name will become clear in a much later chapter. It isn't important yet.

The second common and important task for these samples from the posterior is to evaluate the correlation between the parameters. This is the way to compute standard errors, as well as the diagnose some common problems in model fitting that you'll learn about later in this chapter. Again, once you have samples from the posterior, it's just a matter of summarizing them as if they were data. In this case, we want to compute the covariances between μ and σ , as well as the variances of each. From the *variance-covariance matrix* that results, it is easy to compute standard errors (Chapter 2, page XX), as well as correlations between the parameters. This variance-covariance matrix is what you get from a quadratic estimate of when you type `vcov(model)`, where `model` is a fit model.

The first step is to compute the variance-covariance matrix from the raw samples:

```
S <- cov( cbind( sample.mu , sample.sigma ) )
round( S , 7 )
```

R code
4.20

```
sample.mu sample.sigma
sample.mu  0.1714728   0.0006248
sample.sigma 0.0006248   0.0867047
```

I've named this matrix S , because it is custom to notate it with Σ , the Greek capital "S," in mathematical statistics. The diagonal from upper-left to lower-right contains the variances of μ and σ . These provide the quadratic estimate of the shape of the posterior. Remembering the lesson from Chapter 2, to get the standard errors of each parameter, just take the square root of each element of the diagonal:

```
sqrt( diag(S) )
```

R code
4.21

```
sample.mu sample.sigma
0.4140927   0.2944567
```

These are practically identical to the standard errors you will get from `mle2`, in a later subsection.

You can probably guess from the tiny covariances in the off-diagonal elements of S that μ and σ are not correlated very much. You can verify this by converting the variance-covariance matrix to a correlation matrix with `cov2cor`, which is designed for this exact purpose:

```
cov2cor( S )
```

R code
4.22

```
sample.mu sample.sigma
sample.mu  1.000000000  0.005123874
sample.sigma 0.005123874  1.000000000
```

Of course each parameter is perfectly correlated with itself, showing 1's along the diagonal. The off-diagonal correlations between the parameters however are tiny, indicating that knowing μ tells us essentially nothing about σ . In later examples in this chapter and the rest of the book, there will be some substantial and consequential correlations among parameters.

4.2.2.4. Sample size and the normality of the naive posterior. Before moving on to using maximum likelihood as shortcut to all of this inference from the naive posterior, it is worth repeating the analysis of the

heights data above, but now with only a fraction of the original data. The reason to do this is to demonstrate that, in principle, the posterior is not always Gaussian in shape. In particular, variance parameters, like σ , are usually not very normal in shape unless you have a lot of evidence. So you do need to be careful of abusing the quadratic approximation, especially when you are concerned with estimation of variance.

The deep reasons for the posterior of σ tending to have a long right-hand tail are complex. But a useful way to conceive of the problem is that variances must be positive. As a result, there must be more uncertainty about how big the variance (or standard deviation) is than about how small it is. For example, if the variance is estimated to be near zero, then you know for sure that it can't be much smaller. But it could be a lot bigger. In many analyses, particularly those involving hierarchical models with multiple variance components, the posterior density for the variance parameters is rarely symmetrical. This can be true even with a lot of data, in the case of hierarchical models, because of how information in the data informs the estimates. You can wait until much later in the book before concerning yourself with really understanding this point. For now, you just want to be wary of it.

Let's quickly analyze only 20 of the heights from the height data to reveal this issue. To sample 20 random heights from the original list:

R code
4.23

```
h2 <- sample( d2$height , size=20 )
```

Now I'll repeat all the code from the previous subsection, modified to focus on the 20 heights in h2 rather than the original data. I'll compress all of the code together, but it's just what you've already seen above.

R code
4.24

```
mu.list <- seq( from=150, to=170 , length.out=200 )
sigma.list <- seq( from=4 , to=13 , length.out=200 )
post2 <- expand.grid( mu=mu.list , sigma=sigma.list )
post2$nll <- sapply( 1:nrow(post2) , function(z)
  -sum( dnorm( h2 , mean=post2$mu[z] , sd=post2$sigma[z] ,
    log=TRUE ) ) )
post2$prob <- exp( -post2$nll + min(post2$nll) )
sample2.rows <- sample( 1:nrow(post2) , size=10000 , replace=TRUE ,
  prob=post2$prob )
sample2.mu <- post2$mu[ sample2.rows ]
sample2.sigma <- post2$sigma[ sample2.rows ]
plot( sample2.mu , sample2.sigma , cex=0.5 ,
  col=col.alpha("slateblue",0.1) ,
  xlab="mu" , ylab="sigma" , pch=16 )
```

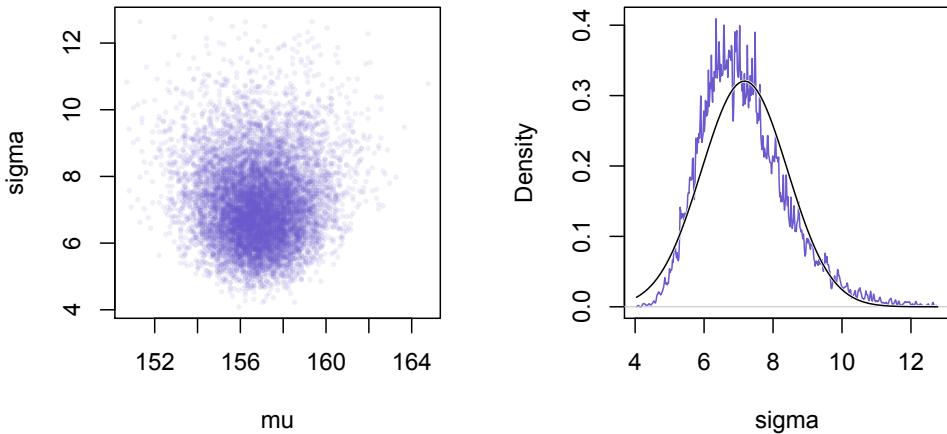


FIGURE 4.4. Samples from the naive posterior for only 20 heights, instead of the full set of 352. On the left: Samples of μ and σ drawn from the naive posterior. On the right: The naive posterior of σ , averaging over μ . The black curve is a comparison to a normal distribution with the same mean and variance. Unlike for the posterior from the full set of 352 heights, now the uncertainty in σ is asymmetrical, with a long tail towards larger values. Your own plots will look a bit different, because the 20 heights are sampled randomly.

After executing the code above, you'll see another scatter plot of the samples from the naive posterior density, but this time you'll notice a distinctly longer tail at the top of the cloud of points. I reproduce a typical plot in the lefthand plot of FIGURE 4.4. Your plot will look somewhat different, because it will have been generated from a different set of 20 random heights. The righthand plot in FIGURE 4.4 shows the estimated posterior density for σ , averaging over μ . The blue curve is the empirical density estimate, and the black curve is a normal approximation with the same mean and variance as the blue distribution. Now you can see that the posterior for σ is not normal in shape, but rather has a long tail of uncertainty towards higher values.

Now, in a very late section of this chapter, I'll show you a clever trick for addressing this problem, one that relies upon *re-parameterizing* the normal distribution. But for now, I just want to emphasize the hazards of assuming that the posterior is normal, when the sample is modest in size. Moreover, just like with inferences about the probability p_W in Chapter 2, the boundary problem persists even with large samples. If the standard deviation σ is very small and gets close to zero, then even with absolutely colossal samples, the posterior will not be Gaussian in shape. This is the boundary problem. Any time a parameter space has a hard boundary, and the maximum likelihood estimate is close to it, then the naive posterior will not be approximately Gaussian.

In the next subsection, you'll see how to produce joint posterior samples from a quick and easy maximum likelihood search. The virtue of the maximum likelihood approach is that it is usually much faster than using a grid approximation of the naive posterior. For complex models with many parameters, it is usually impractical to do what you just did just above. This is because as the number of parameters increases, the number of combinations of parameters increases geometrically. For example, if we add one more parameter and still only consider 100 values of each parameter, then there are 100^3 combinations to compute a log-likelihood at. This implies a table with one-million rows!

We could squeak out a little life in this approach by strategically choosing the parameter combinations to compute and then interpolating. But with even a few more parameters, even this becomes impractical. Since many important regression analyses involve as many as a dozen parameters, this brute force grid approximation approach just isn't a generally useful tool. Now, for simple models, it is functional, if somewhat unfamiliar to most. But for even moderately complex models, some other approach is needed.

Maximum likelihood with quadratic estimated confidence intervals is such an approach. And that's where we turn to next. But keep in mind, much later in the book, I'll show you how to use another approach to sampling from the posterior: Markov Chain Monte Carlo (MCMC). MCMC is computationally intensive, but not even half as demanding as the brute force approach. It can usefully sample from posterior densities with many dimensions (parameters). Still, as long as you have a reasonable sized sample and, especially, if you don't want to make strong inferences about variance parameters, then the maximum likelihood approach with quadratic approximation of the posterior is very useful.

4.2.2.5. *Fitting the model with `mle2`.* Our interest in maximum likelihood estimation, recall, is as a handy way to quickly make inferences about the shape of the naive posterior. The posterior's peak will lie at the maximum likelihood estimate, and we can get approximations of the posterior's shape by using the quadratic approximation to likelihood (Chapter 2, page XX).

To find the values of μ and σ that maximize the likelihood of the observed data, we'll use `mle2` again. So make sure you've loaded the `bbmle` library, and then fit the model with:

```
m <- mle2( height ~ dnorm( mean=mu , sd=sigma ) ,
            data=d2 ,
            start=list(mu=mean(d2$height),sigma=sd(d2$height)) )
```

R code
4.25

After executing this line of code, you'll have a fit model stored in the symbol `m`. Let's step through each part of this command and explain it. The structure is identical to the proportion of water examples from Chapter 2. But the details are different.

First, we have a re-statement of the stochastic node, `height ~ dnorm(mean=mu , sd=sigma)`. This just tells `mle2` that the values in `height` are assigned likelihoods from the normal density, `dnorm`, with mean `mu` and standard deviation `sigma`. The particular names `mu` and `sigma` are arbitrary. I used them for clarity, but you could substitute `pickle` and `tardis`, if you prefer. You just have to make sure that you use the same names when you define the `start` list, which I'll explain a bit later.

Note that the function `dnorm` here is a standard density function in R, just like `dbinom` from previous chapters. You can use it in isolation to compute normal likelihoods. For example, to get the likelihood of observing a measurement of 165 from a normal distribution with mean 170 and standard deviation 10, use:

```
dnorm( 165 , mean=170 , sd=10 )
```

R code
4.26

[1] 0.03520653

Second, we have to tell `mle2` where to find the data, in this case the values named `height`. So we tell it `data=d2`, just referencing the name of the data frame defined earlier. R will automatically look inside this data frame for variable names we use in the definition of the stochastic node.

Finally, this is maximum likelihood estimation, so we have to tell R where it will start searching. Remember, maximum likelihood search

tries out different models (parameter values), until it finds one that plausibly maximizes the likelihood. We use intelligent guesses for the starting values, although as you'll see in later chapters, it isn't always so easy as this. The starting place for the mean, `mu`, is given as the sample mean of the outcome variable, `height`. It turns out the this sample mean is the maximum likelihood estimate, which you could prove analytically with a little calculus, if you wanted. The starting value for the standard deviation, `sigma`, is given as the sample sample deviation. This is not the maximum likelihood estimate of σ , but it is usually very close and a good place to begin. I encourage you to play around with these starting values. In a problem as simple as this one, it won't make much difference which values you choose, but it's good to get into a good habit of using these intelligent guesses.

The names `mu` and `sigma` in the `start` list must be identical to the names for the parameters you chose in the definition of the model. Otherwise `mle2` has no way of knowing which symbols in the model definition are parameters to search over and which are data to look up.

Now take a look at the estimated maximum likelihood model:

R code
4.27

```
precis( m )
```

	Estimate	Std. Error	2.5%	97.5%
mu	154.597093	0.4120811	153.789428	155.404757
sigma	7.731327	0.2913854	7.160222	8.302432

Unlike in the proportion of water problem from Chapter 2, we now have two dimensions in which models differ from one another, the mean μ (`mu`) and standard deviation σ (`sigma`). As a result, the maximum likelihood estimate depends upon estimating both of these parameters and producing confidence estimates about each. The standard errors and 95% quadratic estimate confidence intervals in the table above are large-sample approximations of the naive posterior's shape. I encourage you to compare the HPDI's from the posterior samples in the previous sections to the quadratic confidence intervals displayed above. You'll find that they are almost identical. When the posterior is approximately Gaussian, then this is what you should expect.

In later sections of this chapter, once we have more interesting data problems and actual questions about the relationships among variables, we'll learn how to perform additional analyses with a fit model, like simulating data from it and plotting prediction intervals over the actual data. One step at a time, though.

4.2.3. But what about `lm`? Some readers will already know about the built-in R command, `lm`, for fitting a linear Gaussian regression model like the one in the previous section. If you haven't encountered `lm` yet, don't worry. I explain it at the end of this chapter. The `lm` command works very differently than `mle2`, but in most cases gives the same estimates, and requires a lot less input than `mle2` does. For example, the model you fit just above could be fit by `lm` with `lm(height ~ 1, data=d2)`. So why am I torturing the reader with `mle2`? There are two reasons, and neither of them is sadism.

First, by learning to fit these regression models with `mle2`, you are exposing their relationship to all the other models in this book and, indeed, to statistical inference more broadly. In contrast, `lm` hides everything from you, preventing the student from realizing what they are assuming when they specify the model. In other words, `lm` does a lot of thinking for you, and when you are learning regression, that isn't a good thing. You want to know the likelihood function, and `lm` lets you pretend their isn't a likelihood function at all. That's fine, once you know what you are doing, but it is absolutely dangerous, when you are learning this stuff. You also want to know how the parameters are related to the data, and `lm` hides that as well. Everything you learn by writing those long calls to `mle2` will pay off, once you get into non-linear models later. In contrast, if you start with `lm`, then the later models appear *ad hoc* and unnecessarily difficult.

Second, `lm` treats the standard deviation, σ , of the Gaussian stochastic node as a "nuisance" parameter. It doesn't even report the estimate of σ , in a direct way. As a consequence, it isn't easy to get estimates of standard error for σ , if you fit a model with `lm`. In contrast, `mle2` treats σ just like every other parameter. The uncertainty in σ does matter for prediction and inference. There are plenty of cases in which you will be okay ignoring it, but in other cases you won't. That decision should be up to you, and not the computer.

I do think `lm` is useful. Much of the time, one doesn't care about inferences about σ , and so `lm` is all you need. Also, `lm` is very handy for getting good starting values for the parameters in a call to `mle2`. I'll show you how to use it in that way, later in this chapter. For now, indulge the more-complex calls to `mle2`. You can actually do more with it, as you are about to see. It is usually true that the harder a tool is to learn, the more powerful it is. This is true of the comparison between `lm` and `mle2`. Once you learn `mle2`, you can fit a much wider range of models. More importantly, you will actually know the relationships between the

parameters and the data inside the model, so you will be able to plot model-based predictions.

4.2.4. Adding a predictor variable. Now let's look at how height in these Kalahari foragers covaries with weight. This isn't the most thrilling scientific question, I know. But it is an easy relationship to start with. Go ahead and plot height and weight against one another to get an idea of how strongly they covary:

R code
4.28

```
plot( d2$height ~ d2$weight )
```

There's obviously a relationship here: knowing a person's weight helps you predict height. Now can you make that loose observation into a quantitative model? We've already seen how to model a single Gaussian variable. But usually we're interested in something more than just estimating the mean and standard deviation of a homogenous normal distribution. Rather, as in this case of height, we usually want to relate one or more explanatory variables to one or more outcome variables. When the outcome variables are measures, a normal distribution is a reasonable starting assumption for them. How do we take our Gaussian model from the previous section and incorporate explanatory variables?

Recall the basic Gaussian model:

$$h_i \sim \text{Normal}(\mu, \sigma).$$

We fit the model to the evidence by estimating posterior probabilities of different values of the parameters, μ and σ . Under likelihood estimation, we use a naive posterior that assumes a uniform prior to get these estimates.

How do you get weight into a Gaussian model of height? Let x be the mathematical name for the column of weight measurements. Now we have an explanatory variable x , our weight measures, which is a list of measures of the same length as h , which is our height measures. We'd like to say how knowing the values in x can help us predict the values in h . But there's no x on the righthand side of the model $h_i \sim \text{Normal}(\mu, \sigma)$. Where does it go?

To answer that question, we have to decide what the hypothesis is. The hypothesis will tell us how to incorporate x into the model. The basic strategy in this business is to replace one or more of the fundamental parameters of the stochastic node— μ and σ are the parameters in this case—with *linear models*. In this case, the obvious first hypothesis is that weight predicts average height. This implies a model in which weight

influences the mean. Indeed, this is the usual approach in linear regression, which is what you are building up to right now. But by no means is it the only option, and it's important to keep that fact in mind. If you think that the standard deviation in height covaries with weight, then you could play instead with σ .

To get `weight` into the model in this way, we need to define the mean μ as a function of the values in x . For example, the simplest set of models would assume that each unit of weight—each additional kilogram—increases height by a constant amount. This makes μ , the mean height, into a *linear* function of weight. Our model now has two components, the first the familiar stochastic Gaussian node, and the second a definition that define μ as a function of x , the `weight` values:

$$h_i \sim \text{Normal}(\mu_i, \sigma), \\ \mu_i = \alpha + \beta x_i.$$

Note that the second line above has $=$ in place of \sim . This is because μ_i is not stochastic, but perfectly deterministic, in this model. All of the randomness and error in prediction relies upon the stochastic node, the normal distribution on the first line. There are several new elements in the model above, so let's step through each of them in turn.

4.2.4.1. The explanatory variable. What is that little i again doing next to x and μ ? It reminds us that each value from h , h_i , has a corresponding value from x , x_i , that determines the mean for the distribution of that case (row) in the data. This is a little weird, so it's worth saying it another way. When we construct a prediction for h_i , we have to use the corresponding x_i , not just any old value from the list x . Now here's the weird part. But since h_i is modeled to be a sample from a normal distribution, the matching value x_i doesn't determine the value of h_i , but rather only determines its mean, μ_i . The mean of the normal distribution from which h_i was drawn, μ_i , also matches each row, because it is a function of the x_i value on row i . It's as if every individual i 's height h_i is sampled from a unique normal distribution with mean μ_i . These normal distributions are not observed, but they exist in the model.

Within R, you can think of i as being a row number in the data frame. Suppose for example that you want row 10, so $i = 10$. You could extract a pair x_{10}, y_{10} with:

```
d2[ 10 , c("height","weight") ]
```

R code
4.29

```
height    weight
11 165.1 54.48774
```

Remember, the notation `d2[10,]` means give me row 10 and all columns of the data frame named `d2`.

4.2.4.2. Parameters of a linear model. The equation $\mu_i = \alpha + \beta x_i$ is a *linear model* of the mean of a normal distribution. It is *linear*, because it is the equation for a straight line, with the values x on the horizontal axis and different means μ_i on the vertical axis.

Note that in writing down this model, we've added two new Greek letters, α and β . These are the new parameters we must estimate posterior probabilities for, in order to summarize what the evidence says about the relationship between speed and distance. These parameters actually replace μ_i , in effect, because the mean for any value h_i can be defined entirely in terms of α , β and x_i . We will no longer estimate μ_i directly, but instead through estimating α and β .

So what are α and β ? The parameter α corresponds to the mean of the normal distribution when $x_i = 0$. It is the so-called y-intercept of the linear model. The parameter β is the slope of the line. For every unit increase in x_i , the model says that μ_i increases by exactly β . Thus you can think of β as having the units height/weight, or cm-per-kg. It is the number of additional (marginal) centimeters of height, on average, for every additional kilogram of weight.

Readers who had a traditional training in physics or chemistry will know how to carry units through equations of this kind. For their benefit, here's the model again, now with units of each symbol added.

$$h_i \text{cm} \sim \text{Normal}(\mu_i \text{cm}, \sigma \text{cm}),$$

$$\mu_i \text{cm} = \alpha \text{cm} + \beta \frac{\text{cm}}{\text{kg}} \times x_i \text{kg}.$$

So you can see that β must have units of cm/kg in order for the mean μ_i to have units cm.

4.2.4.3. Fitting the model. The code needed to fit this model via maximum likelihood is a straightforward modification of the kind of code you've already seen. All we have to do is incorporate our new model for the mean into the model specification inside `mle2`, and be sure to add our new parameters to the `start` list:

R code
4.30

```
m2 <- mle2( height ~ dnorm( mean=a + b*weight , sd=sigma ) ,
  data=d2 ,
  start=list(a=mean(d2$height),sigma=sd(d2$height),b=0) )
```

The parameter `mu` is gone, because it has been replaced by the linear model, `a+b*weight`, where `a` is α and `b` is β and `weight` is of course our `x` in this instance. In the `start` list, `mu` is replaced by `a`, which starts at the overall mean, just like `mu` used to. We also have to add the new parameter `b` to this list, and as is usually a conservative first guess, we start the slope out at zero (0), which is equivalent to no relationship.

In model families like this one, with several dimensions along which parameters can vary and affect prediction, maximum likelihood estimation works much as you encountered it in Chapter 2: The computer tries a bunch of different values for the parameters, slowly closing in on a combination of all parameters that minimizes the negative log-likelihood of the model. To get an idea how many calculations go into fitting a model even this simple, I encourage the reader to fit the model above a second time, this time adding the optional parameter `trace=TRUE` to the call to `mle2`. You'll get a screen full of search steps, starting with the starting values you specified in the `start` list and ending with the maximum likelihood estimates.

You'll remember from Chapter 2 that, in order to summarize the shape of the posterior distribution of models (parameter values), you learned how to compute the standard error of an estimate. That amounted to a quadratic approximation of the shape of the posterior. Everything you learned there is still correct. We can get an estimate of the curvature of the naive (flat prior) posterior distribution by computing the second derivative of the negative log-likelihood function, evaluated at the maximum likelihood estimate. Leave this tedious work to the computer. It needs to do it three different times, each time considering the derivative with respect to a different parameter, α , β and σ .

The wrinkle to add to this procedure, as you saw earlier when you visualized the joint posterior for the height data, is that once we have more than one parameter in the model family, the estimates among the parameters may be *correlated*. They vary together. Think about the slope, β , and intercept, α , in the model family fit above. There is no way to change β , the slope of the line predicting the mean, without also changing α , the y-intercept of the line. This is because as soon as you change the tilt of the line, it should hit the y-axis in a different place, if you also want to keep the likelihood high. So the combinations of α and β that produce high likelihoods will be correlated, in most cases. As a result, if you know one of these parameters, then you also have information about the other.

To degree to which two parameters share information can be quantified using the same approach that we used in Chapter 2 to estimate

the standard error of an estimate. But instead of taking the derivative of the negative log-likelihood twice with respect to the same parameter, now we take the derivative once with respect to one and then a second time with respect to the other. Some readers will recognize this description as *cross partial derivative*. How to do this isn't so important in this book—you can let the computer handle it, using the techniques you already met in Chapter 2. The important lesson is that the reason to do this is the estimate how changing the value of one parameter influences the inferred value of another. The second derivatives of each parameter and the second cross derivatives of each combination of parameters all together form a matrix, usually called the *Hessian*. This matrix is named after the 19th century mathematician Ludwig Otto Hesse, whom developed it for our current purpose.

Once we have the Hessian matrix, we can take the reciprocal of each element (look back in Chapter 2, if you've forgotten why) to get the approximate variances of each parameter's likelihood surface (along the diagonal) and covariances between each pair of parameters likelihood surfaces (off the diagonal). Call this matrix a *variance-covariance* matrix for a model. For example, here's the inferred variance-covariance matrix for the braking distance model above:

R code
4.31

```
vcov( m2 )
```

	a	sigma	b
a	3.631610e+00	-2.033566e-05	-7.909510e-02
sigma	-2.033566e-05	3.654458e-02	3.545334e-07
b	-7.909510e-02	3.545334e-07	1.758041e-03

The elements on the diagonal—from upper-left to lower-right—are the variances of each individual parameter. If we take the square-root of each of these, we get the standard errors:

R code
4.32

```
sqrt( diag( vcov(m2) ) )
```

	a	sigma	b
	1.9056784	0.1911664	0.0419290

These are the same values you see reported in the standard error column of the `summary` output of the model, if you want to print that out now and check. The command `diag` just pulls out the diagonal of a matrix.

The off-diagonal elements of `vcov(m2)` are covariances. They tell us how strongly each parameter provides the same information as another parameter. We're going to need to use these covariances a little later in

this chapter to summarize the posterior distribution for this model. But for now, it's much more convenient for interpretation to convert them to correlations. A covariance can be hard to interpret, because its maximum and minimum depend upon the variances of each variable used to compute it. To get an idea how much information a parameter shares with another, it will be easier to interpret the correlation, which just scales the covariance by the variances. That is, the correlation between two variables a and b is defined as:

$$\text{cor}(a, b) = \frac{\text{cov}(a, b)}{\sqrt{\text{var}(a) \text{ var}(b)}}.$$

All this does is re-scale the covariance so that it is always between -1 and 1 , with -1 indicating a perfect negative association (an increase in a means a proportionately equal decrease in b) and 1 indicating a perfect positive association. To get the variance-covariance matrix into a correlation matrix, you could do it yourself, since you have all the variances (on the diagonal) and covariances you'd need to compute each correlation. But R already has a convenient command for this purpose, `cov2cor`:

```
round( cov2cor( vcov(m2) ) , 5 )
```

R code
4.33

	a	sigma	b
a	1.00000	-6e-05	-0.98989
sigma	-0.00006	1e+00	0.00004
b	-0.98989	4e-05	1.00000

I embedded the call to `cov2cor` inside the function `round`, just to make the output easier to read. Looking at the output, you can see that each parameter is perfectly correlated with itself, as it should be. The diagonal is all 1's. But now in the off-diagonals, we see correlations between pairs of parameters. Note that neither `a` nor `b` is hardly correlated at all with `sigma`. This is a typical outcome for many Gaussian models, in which learning about the mean (from `a` and `b` in this case) tells us typically quite little about the error around the mean (`sigma`).

However, the parameters `a` and `b` are very strongly negatively correlated. In fact, they are almost perfectly correlated, in this example. The importance of this fact is that talking about the uncertainty in the predicted mean height for any given weight will depend upon both β and α , as well as their correlation. You can't focus on only the slope β when you construct a confidence interval for the mean.

4.2.4.4. Interpreting the estimates. Once R has done the hard work of finding maximum likelihood estimates and computing the variance-covariance matrix, you'd like to make some interpretations of these numbers. From here throughout the rest of the book, I really want the reader to think of the maximum likelihood solution as being a sort of naive posterior distribution. It gives us qualified inferences about posterior probability, provided samples are not too small and the prior is weak. I encourage this perspective, because it is true and it is empowering. Combined with taking samples from the naive posterior, as I'll teach you in a bit, it makes some tasks, like generating predictions that include uncertainty, into problems of data summary rather than problems of higher mathematics. You'll be able to automatically cope with the correlation between α and β , for example.

Let's look at the estimates displayed from `precis`:

R code
4.34

```
precis( m2 )
```

	Estimate	Std. Error	2.5%	97.5%
a	113.8779674	1.9056784	110.1429063	117.6130284
sigma	5.0720065	0.1911664	4.6973273	5.4466856
b	0.9050541	0.0419290	0.8228747	0.9872334

These are the peaks of the individual naive posterior distributions of each parameter. The standard errors allow us to make inferences about the curvature near these peaks, and the variance-covariance matrix (previous section) lets us talk about how each parameter's posterior is correlated with the others.

So what's going on with our new model that includes `weight`? First, notice that the estimate of the standard deviation has gotten smaller, now that we have included `weight` in the model. The estimate of σ shrank from 7.73 to 5.07. This is because the model is predicting the same overall variation in height by varying the mean of the normal distribution. That is, every unique value of `height` implies its own normal distribution, and if we mix these distributions together, we get the predicted overall distribution of heights. The model assumes that the standard deviations of all of these normal distributions is the same, σ . So if we are to add these distributions together and realize the original overall variance, the variance of each must be smaller than 7.73. This is why the estimate of σ has declined.

You're probably more interested in the estimate of β , b , because it tells us about the ability to predict height using weight. You can interpret the estimate, 0.905, as the expected change in height, given a unit

change in weight. So for every extra kilogram of weight, the maximum likelihood model tells us to expect 0.905 extra centimeters of height. Take a look at the confidence intervals in the output above, to get a sense of how confident we can be in this estimate. The 95% interval is pretty narrow, from 0.823 to 0.987.

It's almost always helpful to plot the estimated effects against the data. Not only does plotting help in interpreting estimates, but it also provides an informal kind of check on model assumptions. When the model's predictions don't come close to key observations or patterns in the plotted data, then you might suspect the model is badly specified and return to critique its assumptions. But even if you only treat plots as a way to help in interpreting estimates, they are invaluable. For simple models like this one, with some experience, it is easy to just read the estimates and understand what the model says. But for even slightly more complex models, especially those that include interaction effects (Chapter 6), interpreting parameter estimates is hard. Combine with this the problem of incorporating the information in `vcov` into your interpretations, and the plots are irreplaceable.

We're going to start with a simple version of that task, superimposing just the maximum likelihood estimate over the height and weight data. The we'll slowly add more and more information to the prediction plots, until we've used the entire naive posterior distribution, all of the information in the estimates and variance-covariance matrix. To superimpose the maximum likelihood estimate for mean height over the actual data:

```
plot( height ~ weight , data=d2 , col="slateblue" )
abline( a=coef(m2)[ "a" ] , b=coef(m2)[ "b" ] )
```

R code
4.35

You can see the resulting plot in FIGURE 4.5. Each point in this plot is a single individual. The black line is the maximum likelihood slope $\hat{\beta}$ and intercept \hat{a} for the mean of Gaussian distribution of heights. Notice that in the code above, I pulled the estimates straight from the fit model, using `coef` and the names of the parameters.

Plots of this sort are useful for getting an impression of the magnitude of the estimated influence of a variable, like `weight`, on an outcome, like `height`. But they do a poor job of communicating uncertainty in the estimate. You might imagine using the 95% confidence interval for $\hat{\beta}$ to plot the extreme high and low values, but this would ignore the fact that the slope and intercept are correlated in a particular way. Indeed, `a` and `b` are very strongly correlated, so if you ignore this correlation when you

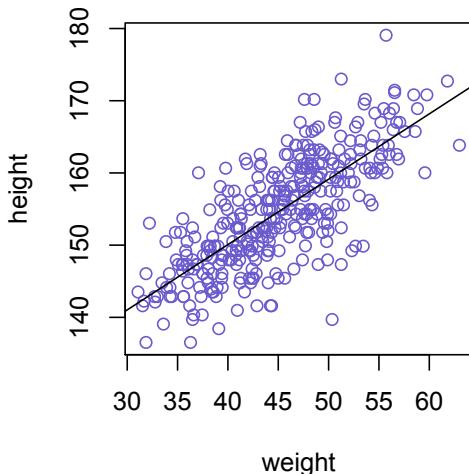


FIGURE 4.5. Height in centimeters (vertical) plotted against weight in kilograms (horizontal), with the maximum likelihood estimate for the mean plotted as a line.

quantify the uncertainty in the regression line, you'll get a very misleading answer. In general, in more complex sets of data, estimates can be correlated in a number of ways, and we need to take those correlations into account when we plot predictions of the model. Whenever changing one parameter leads to correlated changes in other parameters, you have to worry about their joint uncertainty, instead of considering each one in isolation.

So like we did in Chapter 2 for the proportion of water problem and earlier in this chapter for the height data, here we're going to sample models from the naive posterior distribution. The steps here will be the same, but now we'll have to make use of the variance-covariance matrix, to get estimates of how the parameters are correlated. And we'll be assuming, strategically, that the posterior distribution is Gaussian. We have to do it this way, to avoid the exhaustive computations needed to map out the naive posterior like we did earlier in this chapter. We'll draw random samples from a normal distribution in all three parameters— α , β and σ —at once, a so-called multivariate normal distribution. A multivariate normal is just a normal distribution for several variables (in this case, parameters). Each variable is assumed to have a normal distribution. But the variables are also correlated with one another, and so a random sample of any one variable will be correlated with the random samples of the other variables.

This might sound like a strange beast, but it'll make more sense once you visualize it. Thankfully, it's easy to draw samples from a multivariate normal in R. Here's how to build up parameter samples from our three-dimensional Gaussian naive posterior distribution:

```
library(MASS)
post <- as.data.frame(
  mvrnorm( 10000 , mu=coef(m2) , Sigma=vcov(m2) ) )
```

R code
4.36

The command that does the heavy lifting for us is `mvrnorm` (MultiVariate Random normal), which is part of the `MASS` library that is built into R already. You probably don't need to install it, but you do need to load it with the `library` command. What you've just done is take 10-thousand random samples of parameter combinations for α , β and σ from the naive posterior distribution for our model. What you end up with in the symbol `post` is 10-thousand rows, each a sample, and three columns, each a parameter. Wrapping the entire thing in `as.data.frame` just forces the resulting matrix to behave like a familiar data frame, so you can manipulate each column more easily. It's optional, but recommended.

Take a look at the first 10 rows of the resulting posterior samples:

```
post[1:10, ]
```

R code
4.37

	a	sigma	b
1	115.1964	4.992267	0.8776393
2	111.0389	5.169515	0.9758554
3	115.4833	5.133463	0.8726757
4	109.6488	5.005837	0.9812692
5	112.4637	4.678314	0.9384814
6	114.1401	4.659309	0.8940317
7	115.8804	4.729829	0.8573680
8	114.0468	5.021942	0.8919720
9	112.6180	4.860938	0.9366620
10	112.9064	4.883688	0.9340588

Each row is a correlated random sample from the joint posterior of all three parameters, using the covariances you provided from `vcov(m2)`. As a result, the values on each row are correlated with one another to the same extent. You can verify this for yourself by computing the correlations among the columns:

```
cor(post)
```

R code
4.38

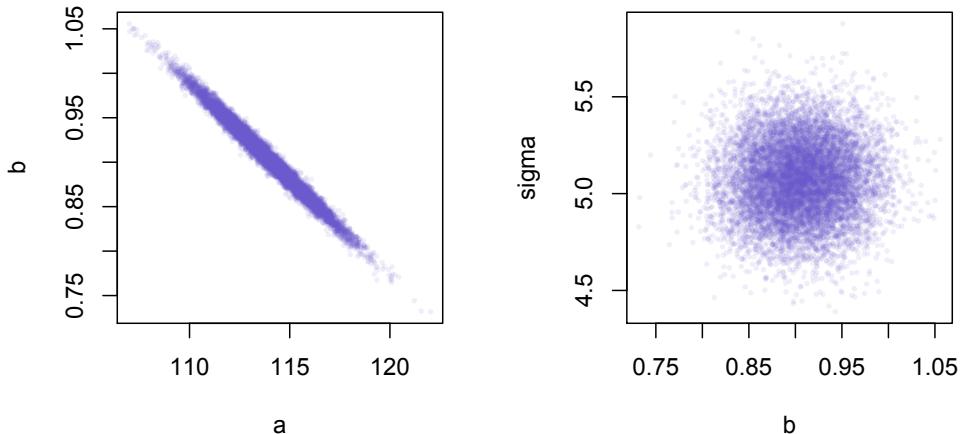


FIGURE 4.6. Samples from the naive posterior distribution for the height/weight model, `m2`. Each point is a joint sample from the posterior, and the cross in each plot is the maximum likelihood estimate. Estimates of the slope `b` and the intercept `a` are strongly negatively correlated (left), while estimates of the slope and the standard deviation `sigma` are essentially uncorrelated (right). The intercept and `sigma` are also uncorrelated (not shown).

	<code>a</code>	<code>sigma</code>	<code>b</code>
<code>a</code>	1.00000000	-0.01989374	-0.98981386
<code>sigma</code>	-0.01989374	1.00000000	0.01987521
<code>b</code>	-0.98981386	0.01987521	1.00000000

Compare that matrix to the output of `cov2cor(vcov(m2))`, the correlation matrix you computed in the previous section.

You can think of each row as being a particular random realization from the universe of models supported by the posterior. The frequency that particular values, and combinations of values, appear in `post` is proportional to the posterior probabilities. And so this list contains a lot of information about the expected relative frequency of different parameter combinations. You can plot the values in `post` now to visualize the covariances among the parameter estimates. I do this in FIGURE 4.6. By now, you should be confident in how to accomplish such a plot on your

own, so I won't show any code to help you. All you need are the columns in `post`.

What can you do with these samples, other than visualize the multi-dimensional posterior? You can easily plot the uncertainty in the mean height from them. The key is to think of this as an empirical problem. Once you have samples from the posterior, it's like you have a bunch of experiments with variable results. You just want to summarize these results. So instead of it being a problem in modeling, it can be a problem in data description.

Here's how to plot a 95% confidence interval around the slope, that incorporates uncertainty in both the slope β and intercept α at the same time. Focus for the moment on a single `weight` value, say 50 kilograms. You can quickly make a list of 10-thousand predicted means for 50 kilograms, by using your samples from the posterior:

```
mu <- post$a + post$b * 50
```

R code
4.39

The code to the right of the `<-` above takes its form from the equation for μ_i :

$$\mu_i = \alpha + \beta x_i.$$

The value of x_i in this case is 50. Go ahead and take a look inside the result, `mu`. It's a vector of predicted means, one for each random sample from the posterior. Since joint `a` and `b` went into computing each, the variation across those means incorporates the uncertainty in and correlation between both parameters. It might be helpful at this point to actually plot the density for this vector of means:

```
dens( mu )
```

R code
4.40

I reproduce this plot in FIGURE 4.7. Since the components of μ have distributions, so to does μ . And since the distributions of α and β are Gaussian, so to is the distribution of μ . Note that the posterior for μ that you see in FIGURE 4.7 is not Gaussian because you assumed that the outcome, height, is Gaussian. Instead, it is Gaussian because the posterior density of each parameter is Gaussian. The posterior can be Gaussian even when the likelihood is not, as you'll see later in this book.

Since the naive posterior for μ is a distribution, you can find confidence intervals for it, just like for any posterior density. To find the 95% highest posterior density interval (HPDI) of heights for these means, just use the `HPDI` command as usual:

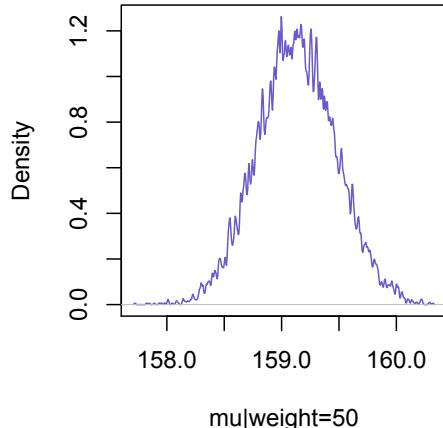


FIGURE 4.7. The naive posterior density of the mean height, μ , when weight is 50 kg. This distribution represents the relative confidence in different predicted values of the mean, as indicated by the evidence.

R code
4.41

```
HPDI( mu , prob=0.95 )
```

	lower	upper
158.4466	159.7910	

Using simple percentile intervals, invoking `PCI`, would work just as well. That's good so far, but we need to repeat the above calculations for every `weight` value on the horizontal axis, in order to draw 95% HPDI's around the maximum likelihood slope in Figure 4.5. This is made simple by strategic use of the `sapply` command:

R code
4.42

```
weight.seq <- 30:63
mu.ci <- sapply( weight.seq , function(z) HPDI( post$a + post$b*z ) )
```

A probability mass of 0.95 is the default for `HPDI`, so omitting the `prob=0.95` above is just to save space. Now `mu.ci` contains 95% lower and upper bound estimates across values of `weight` from 30 to 63 kg. You can plot these boundaries on top of scatter of data and maximum likelihood slope with a couple of calls to `lines`:

R code
4.43

```
plot( height ~ weight , data=d2 , col="slateblue" )
abline( a=coef(m2)["a"] , b=coef(m2)["b"] )
lines( weight.seq , mu.ci[1,] , lty=2 )
lines( weight.seq , mu.ci[2,] , lty=2 )
```

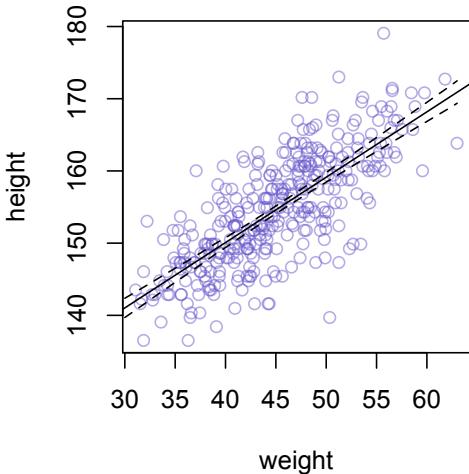


FIGURE 4.8. The !Kung height data again, now with 95% HPDI for the mean. The correlation between slope and intercept in the posterior density leads to greater uncertainty about the mean height, at both low and high weights.

You can see the results in FIGURE 4.8.

Using this approach, you can derive and plot confidence intervals for predictions for quite complicated models. It's true that it is possible to compute confidence intervals like this, from first principles. I have tried teaching the analytical approach before, and it has always been disaster. Part of the reason is probably my own failure as a teacher, but another part is that most social and natural scientists (I teach PhD students from both groups) have never had much training in probability theory and tend to get very nervous around f 's. I'm sure with enough effort, every one of them could learn to do the mathematics. But all of them can quickly learn to generate and summarize samples derived from the posterior distribution. So while the mathematics would be a more elegant approach, and there is some additional insight that comes from knowing the mathematics, the pseudo-empirical approach I just presented here is very flexible and allows a much broader audience of scientists to pull insight from their statistical modeling. And again, when you start estimating models with MCMC, this is really the only approach available. So it's worth learning now.

To summarize, here's the recipe for generating confidence intervals from the naive posterior of a fit model, using the assumption that the posterior is normal.

- (1) Sample all parameters from the naive posterior, using the multivariate normal distribution. This provides many model realizations in proportion to their posterior probabilities.
- (2) Generate predictions, using the samples from the posterior. In the example above, these predictions were for a component of the model, the mean of the normal distribution. But in principle, the predictions could be for any or all components of the model. Since you know how parameters relate to prediction, because you wrote down the model as a stochastic node, you can always generate predictions from samples from the posterior.
- (3) Use HPDI or PCI (if you want simple percentile intervals) to find lower and upper bounds for each value of some variable you want to plot predictions across. Or you could just plot the raw predictions as points, using `points`. I'll provide examples of that approach later.

To make sure you understand this procedure, and to uncover some additional aspects of the height model, let's walk through generating a 95% prediction interval for actual heights, not just the average height. This means we'll incorporate the standard deviation σ and its uncertainty as well, now.

Think about how to modify the previous code to generate actual predictions, rather than just predicted means. Remember, the statistical model here is:

$$\begin{aligned} h_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta x_i. \end{aligned}$$

What you've done so far is just use samples from the posterior to visualize the uncertainty in μ_i , the linear model of the mean. Actual predictions of heights depend also upon the stochastic node in the first line. So all you need to do is build on top of the predictions for the mean by sampling from a normal distribution. You can do all of this in one line again, by using `rnorm`, the function that samples from a normal distribution, inside of `HPDI`.

R code
4.44

```
h.ci <- sapply( weight.seq , function(z)
  HPDI(
    rnorm( n=nrow(post) ,
      mean=post$a + post$b*z , sd=post$sigma ) )
```

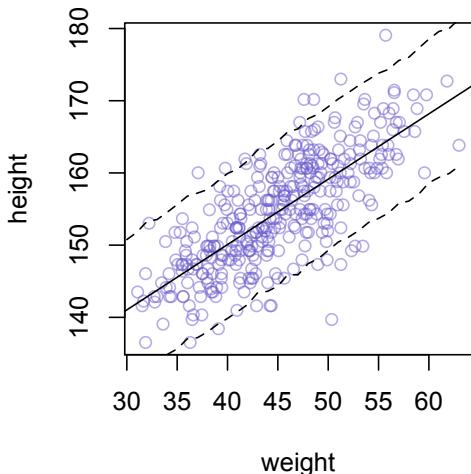


FIGURE 4.9. 95% HPD prediction interval for height, as a function of weight. The solid line is the maximum likelihood estimate of the mean height at each weight. The dashed boundaries represent the region within which we expect to find 95% of actual heights in the population.

I formatted the code above to help you see how each command is embedded in the others. As always, line breaks and spaces don't matter to R, so you can format code in any way that makes it easier for you to read. Now, instead of just computing μ for each sample from the naive posterior, we also use each computed μ to sample from a normal distribution with that mean. We use the posterior samples of σ at the same time. Then `HPDI` takes the entire list of sampled heights and computes lower and upper bounds. This procedure is repeated by `sapply` for each of the `weight` values from 30 to 63, again producing a matrix with 34 columns, one for each value of `weight`, and two rows, one the lower bound and one the upper bound. The values in the rows define an approximate 95% prediction interval for height, as they vary by `weight`.

In FIGURE 4.9, I plot this interval over the raw data. The dashed boundaries in the figure represent the region within which we expect to find 95% of actual heights in the population. There is nothing special about the value 95% here. You could plot the boundary for other percents, such as 50% and 80%, and add those to the plot. Doing so would help you see more of the shape of the predicted distribution of heights. I leave that as an exercise for the reader. Just go back to the code above and add `prob=0.8`, for example, to the call to `HPDI`. That will give you 80% intervals, instead of 95% ones.

4.3. Transformation and Log-Normal Regression

Most of the time, when we are driving a car, we don't think about the brakes as impressive pieces of technology. However, without improvements in brakes, many other improvements in automobiles would not have been possible or practical. Imagine how fast you could comfortably drive, if it took you 10-times the distance to stop your car. Speed limits would have to be much lower than they are now, and there would be little incentive to design faster passenger vehicles. And it's not just when coming to a full stop that brakes help us. Imagine trying to turn without the easy power brakes you currently enjoy.

Braking technology has come a long way in the last hundred years. Most modern passenger cars going 15 mph can brake to a full stop in under 10 feet. In 1902, Ransom E. Olds tested an experimental braking system which was a huge improvement over previous technologies, capable of bringing his Oldsmobile, traveling 14 mph, to a halt in 21.5 feet. You can play around with some braking distance data recorded in 1920, already built into R:

R code
4.45

```
data(cars)
d <- cars
str(d)

'data.frame': 50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

These are pairs of speeds, in miles-per-hour, and braking distances, in feet, for 50 trials. Values on the same row of the data frame form a pair. Go ahead and plot these data, to get an idea for the relationship between speed and braking distance, back in 1920:

R code
4.46

```
plot( d$dist ~ d$speed )
```

At the start of this chapter, I introduced the Gaussian distribution as a common and natural emergent distribution in nature. One of the ways that nature makes normal distributions is by adding together samples from other distributions. But you also saw that, since multiplication on a logarithmic scale becomes addition, even processes that multiply samples can produce normal distributions, once we take the logarithm of the measures. Such distributions are often called *log-normal*, because the logarithm of the measures is approximately normal, while the original measures are not.

In this section, you'll model log-normal measures with a linear model. The braking distance data are actually a standard case of the kind of variable that is often given this treatment. It has a minimum of zero—you can't have a negative braking distance—and the variation around the mean appears to increase as speed increases. That is, the variation is proportional to the mean. Many, but certainly not all, distributions of this sort are approximately Gaussian with constant variation around the mean, once converted to logarithms. Such a conversion is usually called a *transformation of measurement scale*.

Before we get into the details, I have to caution you that your author is not a fan of this approach. While it has its legitimate uses, the drawback to the transformation approach is that it always makes interpretation of the model more difficult and tends to hide assumptions from the analyst. For example, it means that while the statistical model you will fit will be linear on the new transformed scale, it will never be linear on the original measurement scale. These drawbacks make the use of other distributions a more attractive option in many cases. In Chapter 7, you'll learn probability distributions like the gamma distribution that can cope with measures that must be strictly positive. It's almost always better to predict on the same scale the data were measured on, and distributions like gamma make it easy to model strictly positive measures like distance or duration. Additionally, the gamma is often the natural distribution for distances and durations.

Coercing our data into Gaussian form is often just a symptom of the Tyranny of Gauss, a holdover solution from the days in which it was impractical to fit non-Gaussian models. A couple of decades ago, log-normal regression was really the only pragmatic solution. Still, even if you are personally committed to never using log-normal regressions, you should still know how to interpret and critique them, because they are quite common in the sciences, both in archives and in new research. And while transforming outcome variables entails obstacles in interpretation, the same problem doesn't usually apply to transforming predictor variables. So don't think that you should resist transformation in all cases. It's a useful tool, but like all tools in this business, you need to apply it in the right circumstances and when there aren't better tools available.

4.3.1. Transforming the data. The first step in applying a linear model to the logarithms of the braking distances is to log them. My recommendation is to create a new column in your data frame that contains the logarithms of the distances. While you could just use the `log` function inside calls to `mle2` and other commands, sometimes that approach can

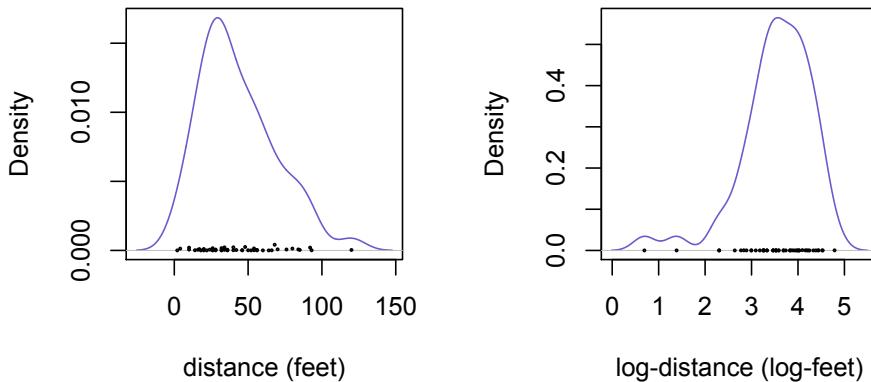


FIGURE 4.10. The effect of logging (taking the logarithm of, not cutting down and making into paper) the braking distances. On the left, the original braking distances have a rightward skew and a few observed values very close to zero. On the right, the newly-logged distances are much more symmetrical in distribution. However, those values that were originally very close to zero are now spread out on the left tail of the distribution.

go horribly wrong, for reasons that have nothing to do with science and everything to do with programming details. If you go the route of making a new column, it's much harder to get into trouble.

So let's make a new column.

R code
4.47

```
d$log.dist <- log(d$dist)
plot( log.dist ~ speed , data=d )
```

After executing the code above, you'll have a new column in the data frame `d` and see a scatter plot of the logged distances against speed. I do not reproduce that plot here. But in FIGURE 4.10, I show the density estimates of the original and logged braking distances, for comparison. The actual measures are shown by the black points along the horizontal axis. The smoothed density estimates are shown by the blue curves. These densities illustrate what typically happens when you log measures, especially when some of the original measures are close to zero. On the left, the distances are originally right-skewed, with the mode lower

than the mean. A few values are very close to zero in the original data. Once we log the distances, the righthand plot demonstrates that the logged measures are now more symmetrical. However, those distances that were originally quite close to zero have trailed behind and now look like outliers of some kind.

What happened to these small distances would happen to any values close to zero, after taking a logarithm. And this is one of the first cautions about modeling logged data. A funny thing about logarithms is that they assign half of the entire real number line—from $-\infty$ to 0—to original values between 0 and 1. As a result, small values on the original scale end up, once transformed to logs, flared out wildly on the left side of large values on the original scale. Large values on the original scale are increasingly scrunched up on the log scale, so that the right end of the distribution becomes increasingly steep. This can make data that were original right-skewed into data that are wildly left-skewed. But worse than that, if there are any zeros in the original data, then logging them will produce $-\infty$, which you can't analyze with a statistical model.

Frequently there are raw zeros in the original data, however. So to complete the log-transform, you have to add a constant to all of the original measures, before taking the logarithm. In fact, it is common to search for a constant that makes the resulting log-normal look as normal as possible. For example, we might compute another set of log-normal distances, but by adding 20 to each distance before taking the logarithm:

```
log.dist.20 <- log( d$dist + 20 )
```

R code
4.48

I plot the resulting density in FIGURE 4.11. The resulting density looks more Gaussian now. This is a common procedure, but it isn't harmless. The choice of constant matters—it affects the inferences, even once all the estimates are converted back to the original measurement scale. I'm going to show you how to fit the log-normal model, and afterwards the reader should re-do all of the fits and plots using the `log.dist.20` variable. You'll see that you get a somewhat different result. Since there's no clear way to justify the choice of the constant 20, all sorts of shenanigans are possible when transforming variables in this way. At least, for those who are victims of the Tyranny of Gauss, these shenanigans allow them to tune estimates so they can cross the arbitrary line of significance.

4.3.2. Fitting the log-normal model. Fitting this model is identical to fitting the regular linear regression model. The only difference is that

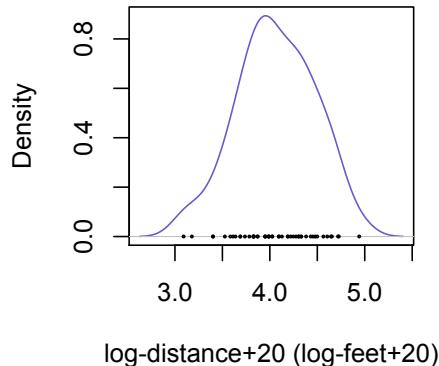


FIGURE 4.11. The braking distances after adding 20 and taking the logarithm. Adding the constant 20 has made the resulting density more normal in shape. Compare to the righthand plot in Figure 4.10.

the outcome variable is our new log-normal variable, `log.dist`. Here's the code:

R code
4.49

```
m3 <- mle2( log.dist ~ dnorm( mean=a + b*speed , sd=sigma ) ,
  data=d ,
  start=list(a=mean(d$log.dist),b=0,sigma=sd(d$log.dist)) )
precis( m3 )
```

	Estimate	Std. Error	2.5%	97.5%
a	1.6761237	0.19217485	1.29946793	2.0527795
b	0.1207652	0.01181502	0.09760816	0.1439222
sigma	0.4373148	0.04373177	0.35160210	0.5230275

Note that the parameter estimates are now on the log scale, so the meaning of $\beta \approx 0.12$ (`b`) is no longer the change in feet for every one mile-per-hour change in speed. Instead it is the change in log-feet for every one mph change in speed. What the heck is log-feet? Well, it's the logarithm of feet. Unless you work with this kind of estimate a lot, it's pretty confusing.

Some readers will know about the percentage-change interpretation of coefficients estimated on the log scale. It's very common in economics to estimate regressions of income in this way and to interpret estimates as percent changes. Gelman and Hill have a clear explanation of this interpretation in their 2007 book on multilevel modeling.⁴⁶ While it's sometimes useful, and people who use it a lot are usually wary of its flaws, I don't encourage this interpretation. In my experience, students and colleagues routinely forget that it only applies to small estimates and

then only when the percentage change is itself small. If you want to know what the model predicts for a big change in the predictor, like say a 10% change in speed, then it's no longer safe to use the estimates in this way. You need to learn a more general approach.

So how do we relate estimates on the log scale to the original measurement scale then? To make interpretation easier, we'll have to convert everything back to the original scale, a process called *back transformation*. This isn't as appealing as the simple percent-change interpretation, but at least it is always correct and exact. Let's plot both the log-normal fit and then convert everything back to the original measurement scale and plot the model again, so you can see how this back transformation is done.

As usual, sample from the naive posterior of the model and compute intervals for the mean and outcome:

```
post <- sample.naive.posterior( m3 )
mu.ci <- sapply( 0:30 , function(z) HPDI( post$a + post$b*z ) )
log.dist.ci <- sapply( 0:30 , function(z)
  HPDI(
    rnorm( nrow(post)*10 ,
      mean=post$a + post$b*z ,
      sd=post$sigma ) )
```

R code
4.50

I'm slipping in here the introduction of the command `sample.naive.posterior`, which is just a convenience function that invokes `mvrnorm` for you and forces the result into a data frame. It's part of the code LIBRARY that accompanies this book. There's no advantage to using it, other than it means writing less code.

In FIGURE 4.12, I show the prediction intervals over the raw data, on both the log-normal scale (lefthand plot) and the back transformed original measurement scale (righthand plot). You know how to make the lefthand plot, because it's the same sort of code you've already seen repeatedly in this chapter. What about back transforming to get the right-hand plot? It's worth thinking about what back transforming means. That way, you can apply this lesson to any transformation you come across. If we have a model, as we do, that states:

$$\begin{aligned} \log(y_i) &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta x_i, \end{aligned}$$

then this means the mean on the original measurement scale must be:

$$\exp(\mu_i) = \exp(\alpha + \beta x_i).$$

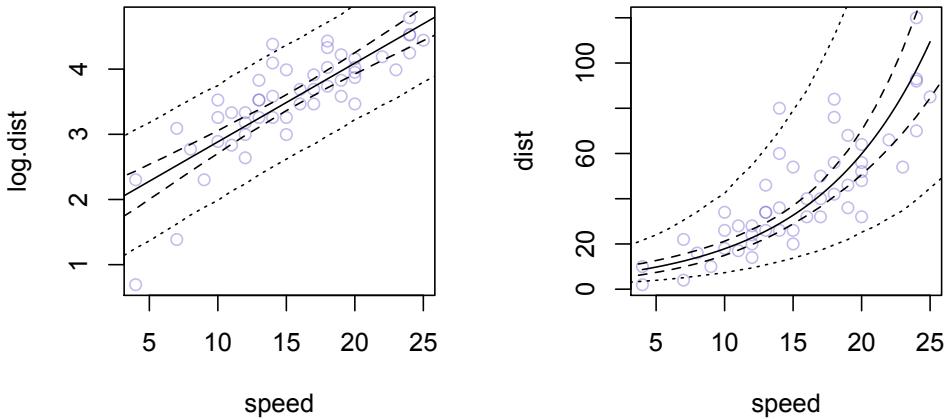


FIGURE 4.12. Log-normal regression of the braking distance data. In both plots, the solid curve indicates the maximum likelihood model of the mean and the dashed and dotted boundaries show the 95% HPDI's of the mean and overall data, respectively. The lefthand plot shows the fit of a linear model of the mean log braking distance. The righthand plot shows the same model, but now on the original measurement scale. Notice that the model is linear on the log-normal scale, as intended, but clearly non-linear on the original scale of measurement. A log-normal model with constant variance on the left also has non-constant variance on the right.

Why exponentiate the equation for the mean? Because that's the reverse of taking the logarithm. So it undoes the transformation. So to make the righthand plot in FIGURE 4.12, you just need to exponentiate the equation for the mean and the values in the computed intervals.

R code
4.51

```
plot( dist ~ speed , data=d , col="slateblue" )
curve( exp( coef(m3)[ "a" ] + coef(m3)[ "b" ]*x ) , add=TRUE )
lines( 0:30 , exp(mu.ci[1,]) , lty=2 )
lines( 0:30 , exp(mu.ci[2,]) , lty=2 )
lines( 0:30 , exp(log.dist.ci[1,]) , lty=3 )
lines( 0:30 , exp(log.dist.ci[2,]) , lty=3 )
```

The model and its predictions are linear on the log-feet scale, but quite non-linear on the original scale. The variance around the mean is also constant on the log scale, as the model assumed it was, but the variance is definitely not constant on the original scale. In this case, these are good features of the log-normal model, because it's plausible that braking distance does increase in a super-linear fashion as speed increases. Likewise, the variance plausibly scales with the mean.

You can use the back transformation approach to compute specific predictions too, of course. For example, if you want to know the effect of doubling the speed from 10 mph to 20 mph, you can compute:

```
dist.10mph <- exp( coef(m3)[ "a" ] + coef(m3)[ "b" ]*10 )
dist.20mph <- exp( coef(m3)[ "a" ] + coef(m3)[ "b" ]*20 )
dist.20mph - dist.10mph
```

R code
4.52

```
a
41.94356
```

So the model predicts an average increase of about 42 feet in braking distance, when the speed doubles from 10 to 20 mph. (Ignore that weird `a` over the prediction. That is just a lingering “name” that carries over from extracting the estimate for `a` from `coef(m3)`. It doesn’t affect the calculation at all.)

4.3.3. What to do about zeros. I said earlier in this section that you can’t take the log of the outcome variable when the variable contains any zeros, because $\log(0) = -\infty$. It’s common for people to add a small constant to the original data before taking the logs, to make sure no infinitely small values are generated. But I also argued earlier in this section that the addition of such constants does affect estimates and there are no guidelines for how to choose the magnitude of the constant. Hopefully you’ve already replicated the log-normal analysis above for the `log.dist.20` variable and seen this for yourself. The constant is not harmless.

So what are you supposed to do when there are actual zeros in the data then? Zeros are sometimes very common in real data, measured on a relevant scale. When the data in question are counts, rather than measures, zeros can in fact be quite common and even arise for a variety of reasons. In later chapters, we’ll confront a few empirical distributions of this sort and see some strategies for modeling them. For now, the important point to take to heart is that dropping the zeros from the data—a sadly common practice—is a very bad idea. Doing so ensures that your estimates will be distorted. When we get to modeling these zeros in the

later chapters, I'll return to this cautionary note and demonstrate the distortion.

4.4. Linear Non-linear Models

By now you are probably bored with the braking distance data. I admit, it's not the most exciting scientific problem. There are only two variables, distance and speed, and the questions we've asked so far aren't very interesting. But the braking distance data is good for building skills in modeling and interpreting and plotting predictions derived from the posterior. All of these lessons will be applicable to much more subtle data examples later in the book. But to be honest, I'll torment the reader again with the braking distance data in Chapter 7, when we see how to really break the Tyranny of Gauss and become comfortable with other stochastic distributions for measures.

But before we even reach Chapter 7, it is worth learning how to model non-linear, curving relationships using only linear models. You read that right: there are linear non-linear models. What makes a model *linear* is that each parameter is multiplied by some function of data and then just added to the other terms in the model. By *non-linear* here I mean only that we can plot the predictions of the model against a column of data and the predicted mean will be curved, rather than a straight line. It turns out that both things can be true of the same model; it can be a *linear non-linear* model of the mean.

The importance of being linear is that such models—like all classical regression and ANOVA and such—are very easy to fit to data. They nearly always have unique and easy-to-find maximum likelihood estimates. Convergence is a snap, whether you use heuristic maximum likelihood searches like we've done so far or fall back on reliable old Ordinary Least Squares (OLS). (If OLS is unfamiliar to you, I briefly review the relationship to maximum likelihood in a later part of this chapter.)

The importance of being non-linear is that the world is not comprised entirely of straight lines. Now, we don't have to think that a relationship is perfectly linear in order to indulge a linear model of the mean. Remember: All models are wrong, anyway. Some models are useful, but even then only within a proper domain. The easiest way to justify a linear model is to note that, over short ranges of an explanatory variable, many really non-linear relationships will appear approximately linear. For example, suppose we plant a dozen genetically identical saplings. Each sapling gets a different amount of water each day for one year. At the end of the year, we measure the heights of these trees. Here are the

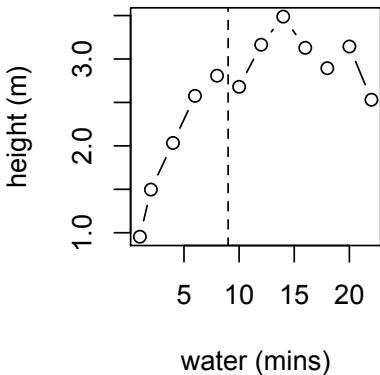


FIGURE 4.13. The tree growth data. The relationship between watering and height is hardly a straight line, but for some range of values, like those left of the vertical dashed line, the relationship is approximately linear.

raw data, with heights given in meters and water in minutes of watering per day.

```
height <- c( 0.954065, 1.496836, 2.033687, 2.575376, 2.807708,
           2.679817, 3.164558, 3.487958, 3.128547, 2.894224, 3.143698,
           2.530882 )
water <- c(1,2,4,6,8,10,12,14,16,18,20,22)
```

R code
4.53

I show these data in Figure 4.13. The relationship between height and water is hardly linear. It appears that once each tree gets enough water, more water has little effect on height. But suppose we only modeled the data to the left of the vertical dashed line in the figure. For small amounts of water, the effect is approximately linear and steep. In this way, it can be useful to use linear models, over intelligently restricted ranges of explanatory variables, to understand natural phenomena, even when the underlying causal relationships are more complex.

Of course we could use a truly non-linear model of this relationship, but at the expense of clarity and ease of estimation. These days, computers are fast enough and software capable enough that even highly non-linear models can be fit to data. But unless the analyst is careful and sufficiently experienced, it is easier to go wrong with a non-linear model—failing to find actual maximum likelihood estimates, the specter of identification, and so on.

Luckily, there is a middle stance, which allows for modeling non-linear relationships like those in Figure 4.13 but staying within the tried and true linear modeling framework. The trick is to use transformations

of the predictor variables—not the outcome variable—to introduce additional predictor variables. As long as each parameter in the linear model of the mean occurs in only one term and is multiplied by only one such variable, the overall model of the mean is still technically linear. I'm going to show you how to use this approach to fit a hypothetically non-linear model to the braking distance data, without transforming the outcome, as you did in the previous section.

4.4.1. A first parabolic model. As always, an example is needed to make this clear. Suppose we make a Gaussian model of the tree height data above, relating water to the mean height using a linear function. In mathematical form, the model is:

Replace data with !Kung height on age quadratic.

$$\begin{aligned} h_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta w_i, \end{aligned}$$

where h is the vector of heights and w the vector of watering durations. This is the same basic linear Gaussian model you used already on the braking distance data. To fit the model to our data, first encapsulate the heights and water amounts into a data frame, for convenience:

R code
4.54

```
d <- data.frame( height=height , water=water )
```

Now fit the model with your friend, `mle2`:

R code
4.55

```
m1 <- mle2( height ~ dnorm( mean=a+b*water , sd=sigma ) ,
  data=d ,
  start=list(a=mean(d$height),b=0,sigma=sd(d$height)) )
```

Before plotting the predicted mean height derived from this model, let's also define and fit a non-linear model to compare it to. The simplest and most common way to consider the idea that a relationship may not be a straight line is to make the equation for a line into an equation for a polynomial of higher order, like a parabola or a cubic equation. We'll use the parabolic model here, which is very common. The model itself just adds a new parameter and squares the prediction variable to force a parabolic relationship between the outcome (height) and predictor

(water). In math form, this is what the model looks like:

$$h_i \sim \text{Normal}(\mu_i, \sigma), \\ \mu_i = \alpha + \beta w_i + \beta_2 w_i^2,$$

Since the equation for μ_i now has a quadratic term in it, it has become the equation for a parabola. When the parameter β_2 is positive, it means the a unit increase in w_i leads to larger and larger increases in h_i , as w_i becomes larger. When $\beta_2 < 0$, the opposite is true: larger values of w_i lead to smaller and smaller increases in h_i , until the direction of the relationship actually changes and increases in w_i lead to smaller values of h_i .

Obviously this is a pretty descriptive style of modeling—we don't actually want to claim that the relationship should be perfectly parabolic. Instead, the reason to adopt this function for μ_i is so we can evaluate a curved model of the response to water, but stay within the robust linear modeling framework. Later in the book, you'll see how to model other curved relationships between predictors and outcomes. Most of them involve tucking the parameter itself, β_2 in this case, into another function, which makes the model truly non-linear.

Now fit the parabolic model, using:

```
m2 <- mle2(
  height ~ dnorm(
    mean=a + b*water + b2*water^2 ,
    sd=sigma ) ,
  data=d ,
  start=list(a=mean(d$height),b=0,b2=0,sigma=sd(d$height)) )
```

R code
4.56

If you inspect the estimates from this model, you'll see that the maximum likelihood estimate of β_2 is negative and reliably so—it's 95% confidence interval is entirely below zero. This suggests that water has a diminishing effect on height. The longer you water, the less effect each additional minute of watering has on growth. If the estimate of β_2 were instead positive, then it would suggest that watering longer has increasing returns in growth.

In Figure 4.14, I plot both the linear model, $m1$, and the parabolic model, $m2$, over the data. The blue line is the predicted mean for $m1$ and the orange curve for $m2$. The dashed boundaries in the same colors show the 95% confidence intervals for the means, computed from the naive posterior distributions. Without computing anything more, it is easy to see that the straight line does a pretty poor job, in comparison

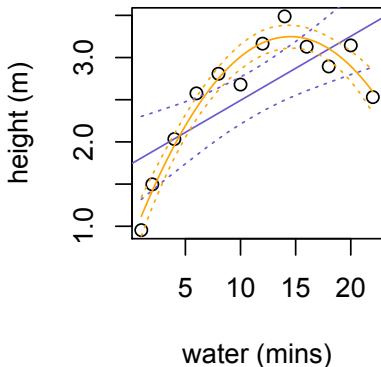


FIGURE 4.14. Straight line (blue) and parabolic (orange) models of the mean height as a function of watering. Dashed regions show the 95% confidence intervals around the means, computed from the naive posterior distributions.

to the parabola. Not only does the orange curve do a much better job of clinging to the data, but the error around it, as quantified by the naive posterior distribution, is much narrower.

Don't get too excited about polynomial models of this kind, though. In the next chapter, we'll see that the parabolic model will *always* fit the data better than the straight line model. Always. No exceptions. So then we have to find some way to decide when the parabolic model is better at prediction, other than observing that it fits the data better.

Here's the code for computing the 95% intervals of the means, for both models. The technique is identical to that used in the braking distance models.

R code
4.57

```
post1 <- as.data.frame(
  mvtnorm( 10000 , mu=coef(m1) , Sigma=vcov(m1) ) )
post2 <- as.data.frame(
  mvtnorm( 10000 , mu=coef(m2) , Sigma=vcov(m2) ) )
mu1.ci <- sapply( 1:22 , function(z) HPDI( post1$a + post1$b*z ) )
mu2.ci <- sapply( 1:22 , function(z)
  HPDI( post2$a + post2$b*z + post2$b2*z^2 ) )
```

The only new trick needed to plot these predictions is to note that you can no longer use `abline`, which just plots a line using its intercept and slope, to plot the mean for `m2`. This is because the mean for the parabolic model is not a line, of course. But luckily there is a convenient command in R, `curve`, that will draw an arbitrary continuous function.

To use it, in this case, prepare the plot by producing the scatter, and then execute:

```
curve( coef(m2)[ "a" ] + coef(m2)[ "b" ]*x + coef(m2)[ "b2" ]*x^2 ,  
add=TRUE , col="orange" )
```

R code
4.58

The model part of this is just the function that defines μ , with x substituted in everyplace where w_i was. What `curve` does is substitute in a bunch of different values for x , to plot out the curve. By also specifying `add=TRUE` in the call to `curve`, you make R add the function on the current plot, instead of making an entirely new plot. You should be able to modify earlier code now and reproduce the plot in Figure 4.14.

4.4.2. A parabolic model of braking distance. Let's apply the same technique now to the familiar braking distance data. If you squint at the earlier plots of distance against speed, you might suspect that distance actually curves upwards as speed increases, rather than being a straight line, as was assumed. Let's see how to fit such a model in this case.

In mathematical form, the model structure for considering a parabolic shape of distance as a function of speed is:

$$\begin{aligned}y_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta x_i + \beta_2 x_i^2.\end{aligned}$$

And the code to fit it is similarly:

```
data( cars )  
d <- cars  
m4 <- mle2(  
  dist ~ dnorm(  
    mean=a + b*speed + b2*speed^2 , sd=sigma ) ,  
  data=d ,  
  start=list(a=mean(d$dist),sigma=sd(d$dist),b=0,b2=0) )
```

R code
4.59

The `d <- cars` at the start there is just to make sure the distance data is in the symbol `d`, because we used the same name for the tree height data. Now for the confidence interval around the mean and for distance more generally, computed from the naive posterior:

```
post <- as.data.frame(  
  mvtnorm( 10000 , mu=coef(m4) , Sigma=vcov(m4) ) )  
mu.ci <- sapply( 0:30 , function(z)  
  HPDI( post$a + post$b*z + post$b2*z^2 ) )
```

R code
4.60

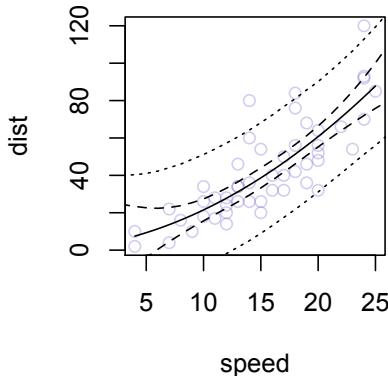


FIGURE 4.15. The parabolic model of mean braking distance as a function of speed. The dashed boundaries are the 95% HPDI for the mean, computed from the naive posterior distribution, and the dotted boundaries are the 95% HPDI for braking distance itself.

```
dist.ci <- sapply( 0:30 , function(z)
  HPDI(
    rnorm( nrow(post)*10 ,
      mean=post$a + post$b*z + post$b2*z^2 ,
      sd=post$sigma ) ) )
```

Figure 4.15 shows the resulting predictions. One of the features of using the entire naive posterior distribution to compute predictions is that you automatically find places where the model is most uncertain about the relationship in question. For example, in this case there is substantially more uncertainty about the relationship between speed and braking distance at low speeds than at high speeds, as you can see from the wider intervals on the left side of the plot in Figure 4.15. Plotting only the predicted mean itself (the solid curve) would not reveal this uncertainty. Remember, the maxim is to embrace and propagate uncertainty.

What have we gained by using a non-linear-but-linear (parabolic) model of the mean braking distance? First, it really is plausible that the relationship curves upwards at high speeds—the maximum likelihood estimate is $\hat{\beta}_2 = 0.99$, which does curve upwards. You might be tempted to use the confidence interval around β_2 to decide how confident we can be in the parabolic shape. Go ahead and compute a 95% confidence interval for $b2$. You'll find that the interval does include zero, just barely. This is just restating what you can already see in Figure 4.15—the bottom dashed boundary for the 95% confidence interval of the mean is

nearly a straight line. If the relationship is really curved, it's not curved very strongly over these speeds, from 5 mph to 25 mph.

Don't be tempted here to perform some kind of significance test, though, rejecting the hypothesis that the relationship is curved, just because the 95% confidence interval for β_2 includes zero. If you do feel tempted, please go back and read Chapter 3 again. You'll have to wait until the next chapter to evaluate in a principled way—not using P -values—whether or not the parabolic model would be better at prediction than the straight line model. But engaging in a significance test here would, as always, imply that you want to argue for some prior preference for or belief in the straight line model, the model in which $\beta_2 = 0$ exactly. Even if you really do want to adopt such a position, a significance test leaves the strength and nature of that prior preference vague and uninterpretable.

4.5. Multivariate Models

Statistical modeling takes place in the small worlds of models and nominated variables. In such a context, there are never any events that were not already nominated, no parameters that were not already specified, no variables that were not already included. But science takes place in the large world. In this world, our models are always mis-specified. They routinely miss the real causal variables and postulate the wrong relationship among variables. Even models that predict well can be wildly mistaken, and we may only realize that once we push the models outside the domain they are comfortable in.

For these reasons, one of the most important components of statistical thinking is to imagine the large world perspective on a model. How could the relationship suggested by the model be a mistake? What other, un-included variables might change the inferences? Does the structure of the model—its distributions and the functions that map variables onto their shapes—mislead us or limit our inferences?

In addressing these questions, it is usually necessary to incorporate more than one prediction variable into a model. A series of two-variable models—one outcome and one predictor—will not reveal how multiple causes affect a single outcome. They will not reveal spurious correlations—associations between a predictor and an outcome that evaporate in the presence of another predictor. For these reasons and others, we need a way to model multivariate processes. And as models include more variables, they also become harder to understand, so we'll need some new ways to interpret them.

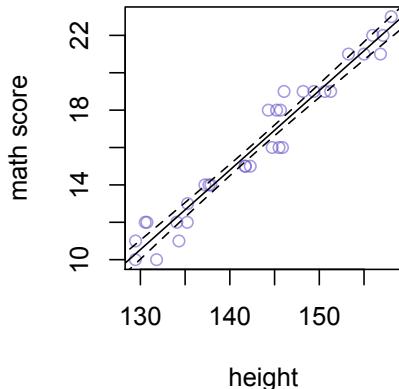


FIGURE 4.16. The regression of math score on height. Taller students answered more questions correctly, and this relationship is strong and precise.

In this section, you'll see how to include any arbitrary number of *main effects* in your linear model of the Gaussian mean. These main effects are additive combinations of variables that have hypothetically independent causal effects on the outcome. No matter what the values of any other predictor variables, each variable in the model still has a constant and direct influence on the outcome, and this influence is measured by its β coefficient.

Of course, in reality, this may not be true—the strength of effect of one cause often depends upon the presence of another cause. I think that *interaction* effects of that kind are extremely important. They are so important that I devote an entire chapter to them (Chapter 6). But it'll be easier to understand interactions, if we deal with multivariate models without them, first.

Multiple predictor variables are not always a help, however. They can also hurt, through decreasing the precision with which we estimate other effects and even removing our ability to detect strong real associations. So it won't do to just throw all of the available variables into a giant regression model. Instead, we need to be selective—building models in full light of background theory and our knowledge of how the posterior is estimated—and as a complementary approach fit different models including different variables in different relationships to one another. In the rest of this section, we deal only with the ways in which multivariate regressions can be built, interpreted, and diagnosed for problems. In the next chapter, we return to the topic of building and comparing more than one model, each structurally different than the others.

4.5.1. Spurious Association. An example will help. Let's consider a simple three variable data context that illustrates how including additional variables in a model can lead to greatly reduced estimates of the influence of other variables. In Figure 4.16, I plot the number of correct answers on a standardized mathematics test against the heights of the students, aged between 5 and 11 years, who took the test.⁴⁷ There is a very strong positive relationship between math score and height. If you had no other information about a student, knowing his or her height would tell you a lot about his or her math score.

But surely height does not *cause* a student to do better at math. It is only correlated with math score in this case, because some other variable is driving both math score and height at the same time, leading them to be strongly associated with one another. In this case, you can probably intuit the variable that is driving both: age. These students are between the ages of 5 and 11 years old. Between those ages, both height and mathematical reasoning increase in a nearly linear fashion. This leads both height and the number of correct answer on the exam to be highly correlated with one another, just because both are extremely correlated with age.

Let's look at the raw data and slowly work through this three variable problem. (These data are taken from Grafen and Hails 2002.) This kind of problem is very common in science. A triad of variables are highly correlated with one another, but only one of those variables is driving the pattern. The other correlations are *spurious*, in the sense that they don't tell us much about cause. Here are the raw data, in the form of vectors. The code below just defines the vectors and stuffs them into a convenient data frame. One reason to show you data entry in this awkward way is to show you that you can get data into R in this way. Sometimes you just don't need to make and load an external data file.

```
vmath <- c(10, 11, 10, 12, 12, 11, 12, 13, 12, 14, 14, 14,
15, 15, 15, 16, 16, 16, 18, 18, 18, 19, 19, 19, 19, 19,
21, 21, 22, 21, 22, 23 )
vyears <- c(5, 5.2, 5.4, 5.6, 5.8, 6, 6.2, 6.4, 6.6, 6.8, 7,
7.2, 7.4, 7.6, 7.8, 8, 8.2, 8.4, 8.6, 8.8, 9, 9.2, 9.4,
9.6, 9.8, 10, 10.2, 10.4, 10.6, 10.8, 11, 11.2 )
vheight <- c(129.44, 129.46, 131.81, 130.54, 130.73, 134.31,
134.07, 135.27, 135.24, 137.21, 137.66, 137.88, 141.74,
142.28, 141.71, 144.73, 145.88, 145.5, 144.3, 145.23,
145.66, 146.06, 148.19, 151.27, 149.4, 150.62, 153.26,
156.84, 155.97, 155, 157.13, 158.04 )
```

R code
4.61

```
d <- data.frame( math=vmath , years=vyears , height=vheight )
```

Now take a look at the correlations among these variables:

R code
4.62

```
cor( d )
```

	math	years	height
math	1.0000000	0.9889461	0.9764021
years	0.9889461	1.0000000	0.9889419
height	0.9764021	0.9889419	1.0000000

You might also try `pairs(d)` to get scatter plots that show these correlations. However you inspect these variables, the conclusion is clear: all three are correlated more than 95% with one another. The problem isn't just that each potential explanatory variable—`height` or `years`—is highly correlated with the outcome, `math`. It's also that both explanatory variables are highly correlated with one another. They mostly contain the same information, so how can we tell which is driving math performance?

Since you have a rich background theory in this domain—that is, you know enough about human development to know which variable is driving the others—you can make sense of these correlations. But in many cases, we don't have enough background information to immediately decide which variable is driving the others. In those cases, statistical modeling isn't going to be sufficient to reveal the causal variable, but it sure can help.

The first thing to demonstrate is that multiple two-variable (one outcome, one predictor) models don't help. Go ahead and fit two Gaussian models, the first modeling the mean math score as a linear function of height and the second modeling math score as a linear function of years of age. In both cases, you'll find a very strong positive relationship between the explanatory variable—either `height` or `years`—and the outcome, `math`. You can see as much in the `pairs(d)` plot.

But we can ask if either of those variables, `height` or `years`, is linearly related to `math`, after taking account of the other. That is, we want to know whether height (or years) predicts math score, once we have already removed the correlation. Consider the model:

$$\begin{aligned} m_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta_h h_i + \beta_y y_i. \end{aligned}$$

This model says that each student score m_i is drawn from a normal distribution with mean defined as a linear combination of each student's height h_i , years of age y_i and a constant (α). This is a simple multiple linear regression. There are two β parameters now, one for each predictor variable in the equation for the mean. I've named each them with a subscript to match the name of the variable they refer to, and I'll try to stick to that convention for most of the book. Note that these parameters are no longer the same as the slope of the regression line between each predictor and the outcome. Instead, there is now a more complex geometric interpretation, one that is increasingly hard to visualize and even conceptualize as the number of predictor variables grows. However, it is still true that each β parameter specifies the rate of change in the outcome for each unit increase in the predictor variable, and that will not change as the number of explanatory variables increases.

Fitting such a model is a straightforward extension of the code you've already seen: just add the new equation for the mean, and be sure to name each parameter in the `start` list:

```
m.math2 <- mle2( math ~ dnorm(
  mean=a + bh*height + by*years , sd=sigma ) , data=d ,
  start=list(a=mean(d$math),bh=0,by=0,sigma=sd(d$math)) )
precis( m.math2 )
```

R code
4.63

	Estimate	Std. Error	2.5%	97.5%	Pr(S)
a	3.02672089	0.003239601	3.02037139	3.03307039	< 2.22e-16
bh	-0.03185984	0.004205410	-0.04010229	-0.02361739	1.7875e-14
by	2.18418373	0.073002737	2.04110100	2.32726647	< 2.22e-16
sigma	0.56202629	0.070253780	0.42433141	0.69972117	6.2238e-16

The maximum likelihood estimates for are $\hat{\beta}_h = -0.032$ and $\hat{\beta}_y = 2.18$. Both estimates are reliably below and above zero, respectively. So we find here a small negative effect of height on math score and a large positive effect of years of age on math score. The estimate for height says that we can expect a decrease of 0.03 correct answers, for every unit increase in height, *once the effect of years has been removed*. Likewise, the estimate for years says that we can expect an increase of about 2.2 correct answers, for every additional year of age, *once the effect of height has been removed*.

FIGURE 4.17 plots these estimates. The plot on the lefthand side shows the predicted relationship between height and math score, holding years of age constant. What this means is that the predictions were generated for a fixed value of `years`, across all values of `height` used in the plot. The 95% confidence interval of the mean and prediction

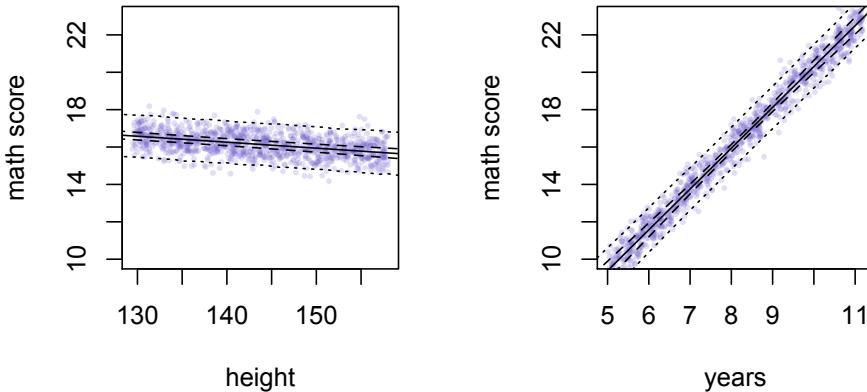


FIGURE 4.17. Multiple regression of math score on both height and years of age, model `m.math2` from the main text. Each plot shows the predicted relationship between math score and each predictor variable, height or years of age. Blue dots are simulated students from the naive posterior. Even though all three variables are very strongly correlated with one another—check `cor(d)`—the linear model concludes that height adds little to our predictions of math score, once we know years of age.

interval of the data were computed in the manner you have become accustomed to. Here's the code to produce the lefthand plot:

R code
4.64

```
post <- sample.naive.posterior( m.math2 )
height.seq <- seq( from=120 , to=160 , by=1 )
mu.ci <- sapply( height.seq , function(z)
  HPDI( post$a + post$bh*z + post$by*mean(d$years) ) )
math.ci <- sapply( height.seq , function(z)
  HPDI(
    rnorm( nrow(post) ,
      mean=post$a + post$bh*z + post$by*mean(d$years) ,
      sd=post$sigma ) ) )
plot( math ~ height , data=d , type="n" , ylab="math score" )
abline( a=coef(m.math2)[ "a"]+coef(m.math2)[ "by"]*mean(d$years) ,
  b=coef(m.math2)[ "bh" ] )
lines( height.seq , mu.ci[1,] , lty=2 )
```

```
lines( height.seq , mu.ci[2,] , lty=2 )
lines( height.seq , math.ci[1,] , lty=3 )
lines( height.seq , math.ci[2,] , lty=3 )
```

Let's step through the new parts of this sequence of commands. First, we sample from the posterior. Next, we need a list of height values to compute intervals across. The `seq` command gives us the list `height.seq`, which we can use repeatedly in the code. Then we compute the confidence interval of the mean, μ_i , across the values in `height.seq`. This code is just like the examples earlier in the chapter, but now there are two predictor variables in the equation for the mean. So what do you do with the one you aren't plotting over, `years` in this case? For truly linear models like this one, the answer doesn't matter much, as long as you use a constant value for `years`. The goal is to see how predictions respond to changes in `height` alone. So we want to fix all other predictor variables at some convenient value and change only the variable of focus right now. A convenient value for a variable you want to hold constant is typically its mean, in this case `mean(d$years)`. So you find that constant value used for `years` in both the calculation of the confidence interval `mu.ci` and the prediction interval `math.ci`. The only other new trick is to use the option `type="n"` when you invoke `plot`. What this option does is suppress the plotting of any points, but R will still set up the axes with the proper ranges and labels. Then you can plot the regression line and intervals with the more primitive drawing commands like `lines`.

What about the blue dots in the figure? These can't be the data, because there aren't that many cases in the original data. In general, it's difficult the plot model predictions over actual data, once you have more than one predictor variable that is even moderately correlated with the outcome. Go ahead and try it yourself, but removing the `type="n"` option in the code above. You see the original raw data now, but the trend line doesn't bear any obvious relationship to them. What to do? There are two common approaches, and I'll demonstrate both of them to you here.

The first approach is to simulate samples from the fit model and plot those. This is what I have done in Figure 4.17. The blue dots in those plots are simulated using the naive posterior. This is actually very easy to do, because all it means is sampling a random normal variable for each sample from the naive posterior. You already did this, when you computed the prediction intervals. This code will generate one-thousand

predicted student math scores, across a range of heights, and plot the points.

R code
4.65

```
sim.height <- runif(1000,min=min(d$height),max=max(d$height) )
sim.math <- rnorm( length(sim.height) ,
  mean=post$a + post$bh*sim.height + post$by*mean(d$years) ,
  sd=post$sigma )
points( sim.height , sim.math , col="slateblue" ,
  cex=0.5 , pch=16 )
```

The first line above just generates one-thousand random heights between the observed minimum, `min(d$height)`, and observed maximum, `max(d$height)`, in the actual data. You can change this 1000 into any number you like, from 10 to 200 to 50000, depending upon how many predicted scores you want to plot. The second line is just like the call to `rnorm` you used when you computed prediction intervals, but now without the HPDI wrapper. These are the raw points used to compute the prediction intervals, in other words. Later in the book, in Chapter ??, we'll get much more serious about simulations of this kind and use them not only to help us understand models but also to help us diagnose when models are misleading us about the true state of the world.

The second way to plot expected data against model predictions, once there is more than one predictor variable, is to use *residuals*. The residuals approach has its limitations. Principally, it doesn't work well once the model is non-linear. Secondarily, plotting predictions against residuals can grossly misrepresent model uncertainty. But for perfectly linear models like this one, residuals can actually help us understand what it could possibly mean to "remove" or "control for" the effect of a variable. You know the equation for the mean, μ_i , and it always has both variables in it, so you know that each prediction for each case i is influenced by both height and years. Sometimes people use an equivalent language of *controlling* for other variables, but this language doesn't really clarify anything. It just relabels it. This is one of those commonplace pieces of statistical language that is confusing, if you actually stop to think about it. We should never be embarrassed to ask for such language to be explained. Let's spend some time figuring out what is really meant here, and in the process, you'll see how to compute and plot residuals.

First, what exactly is a "residual"? A residual is the difference between the predicted mean value of an outcome and the actual value of

an outcome. In math-speak, the residual for a case i is:

$$r_i = y_i - \mu_i,$$

where μ_i is the linear model of the mean of the normal distribution. These residuals can be useful for a number of different purposes. Here, we'll use them to see what it means to remove the effect of a variable and to plot model predictions against the data.

The key thing to note in the definition of the residual is that the model for μ_i can be anything you like. What we're going to do is consider a linear model that predicts `height` using `years`. This model is:

$$\begin{aligned} h_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta y_i. \end{aligned}$$

Why use this model? Aren't we interested in predicting math score, not height? Well, yes. But what this model allows us to do is produce residuals for height that have removed the correlation with years. Once you fit the model above, you can compute residuals. These residuals are the degree to which each student is tall (a positive residual) or short (a negative residual) for his or her age. Once we have these residual heights, we can fit a model that tries to use them to predict math score, and that model will tell us the slope between math score and height, *after removing any linear influence of years of age*. You'll see that we get exactly the same estimate of the influence of height on math score as in the model `m.math2` above, the model that included both height and years.

Let's walk through this procedure, so you're sure of how it works. First, fit the model that tries to predict mean height as a linear function of age:

```
m.hy <- mle2( height ~ dnorm( mean=a + b*years , sd=sigma ) ,
  data=d ,
  start=list(a=mean(d$height),b=0,sigma=sd(d$height)) )
```

R code
4.66

Computing the residuals is just a matter of subtracting the predicted mean, using the maximum likelihood estimates, from the observed heights. Let's do this in two steps, just to be extremely clear about the process. This line of code will calculate μ_i , for each row of actual data, and add the predictions to the data frame:

```
d$pred.mu.hy <- coef(m.hy)[ "a" ] + coef(mu.hy)[ "b" ]*d$years
```

R code
4.67

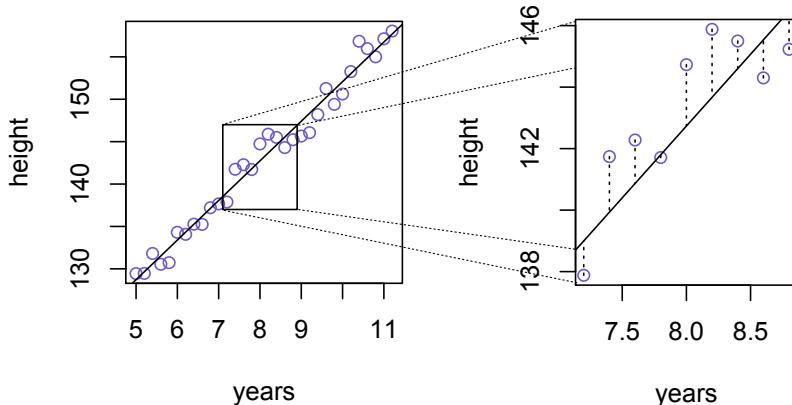


FIGURE 4.18. Computing residuals after regressing height on years of age. Each residual, the length of a dotted line in the righthand plot, represents the degree to which each student is tall (above the regression line) or short (below the line) for his or her age.

Now to compute the residuals themselves, just subtract the actual height on each row from the predicted height you just computed. Again, I add this to the same data frame:

```
R code  
4.68 d$resid.height <- d$height - d$pred.mu.hy
```

I illustrate the process of computing these residuals in Figure 4.18. Think of these residuals as a new explanatory (predictor) variable. The residual height is uncorrelated with years of age, because you explicitly removed that correlation when you regressed height on years. And so if you now regress math score on residual height, you can see ask if how strong the relationship is between math score and height, *controlling for years of age*. This is all “controlling for” usually means.

Let’s fit the regression of math score on residual height and see that you get the same estimate as in model `m.math2`:

```
R code  
4.69 m.resid.height <- mle2( math ~ dnorm( mean=a + b*resid.height ,  
sd=sigma ) , data=d ,  
start=list(a=mean(d$math),b=0,sigma=sd(d$math)) )
```

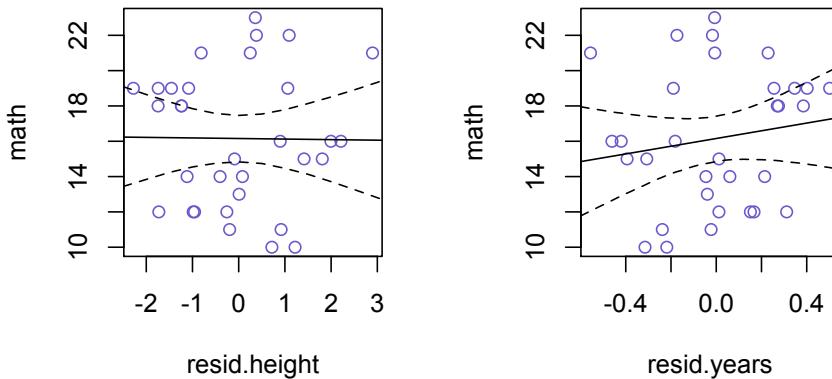


FIGURE 4.19. Bivariate regressions of the residuals of height and years. The maximum likelihood estimates of the slopes are the same as those produced by fitting a single multiple regression (Figure 4.17).

```
precis( m.resid.height )
```

	Estimate	Std. Error	2.5%	97.5%
a	16.15625000	0.6718117	14.839523	17.472977
b	-0.03185227	0.5140290	-1.039331	0.975626
sigma	3.80034116	0.4750414	2.869277	4.731405

I plot math score against the residuals and the slope from this model in the lefthand plot in Figure 4.19. If you compare the estimate for **b** to the estimated coefficient for height from `m.math2`, you'll see that they are almost exactly the same. In other words, the slope of the line in the lefthand plot in Figure 4.19 is the same as the slope in the lefthand plot of Figure 4.17. This is because the awkward two-stage regression you just performed provides the same maximum likelihood estimate as just putting both predictor variables in the same linear model.

The righthand plot is produced using the same steps, but now you start with a regression of years (outcome) on height (predictor) and then calculate residual years. Residual years produced in this way measures how much older (positive residual) or younger (negative residual) each student is for his or her height. Then estimate the regression of math

score on residual years, and you can produce the righthand plot in Figure 4.19. Again, the slope of the line in the righthand plot is the same as the slope in the righthand plot of Figure 4.17.

The point of doing regressions with the residuals is not to show you a useful way to fit models. This procedure is laborious and actually much less useful than fitting a single model with both (all) predictor variables in it. You can see, for example, that the uncertainty about the slopes in Figure 4.19, which used residuals, is much greater than the uncertainty about the slopes in Figure 4.17, which are derived from a single multiple regression model. So you should never actually fit a regression model in this way.

The value of this exercise lies instead in understanding what multiple regression is doing when you include more than one predictor variable. A model of the mean like $\mu_i = \alpha + \beta_h h_i + \beta_y y_i$ asks how useful each main effect is for predicting the mean, after taking account of the other predictor variables. For example, the estimate for β_h can be interpreted as an answer to the question: Are students who are tall for their age better at the math test? The answer is: No, because the estimate is slightly negative. If anything, students who are tall for their age are slightly worse at the test. Likewise, the estimate for β_y is an answer to the question: Are students who are old for their height better at the math test? The answer is: Yes, because the estimate is highly positive with a narrow confidence interval.

Now, keep in mind that if we only knew a student's height, we could still improve predictions of his or her math score with that information. This is because height is strongly correlated with years. The two variables contain a lot of the same information. Using either alone to predict math score is better than using neither. But to address the question of which variable, height or years, is a better predictor and possibly causal of math score, we need the multivariate model that pools the information from both variables and evaluates how much each contributes to prediction, controlling for the other.

Another way to think about what these models can accomplish is to re-imagine them in terms of experimental manipulations. Suppose it were possible to construct two groups of students in which height and years of age are uncorrelated. You might do this by making a group of students who are all the same age, say 8 years old exactly. Everyone in this group has the same birthday. Now height within this group is uncorrelated with years of age, because age does not vary. You have controlled for variation in years of age, but experimentally this time rather than just statistically. The relationship between math score and height

in this group is predicted by the model `m.math2` to be $\hat{\beta}_h \approx -0.032$. In another group, place students who are all exactly the same height, say 140 cm, but have different ages. Now age and height are uncorrelated again, but now because you have controlled for height. The predicted relationship between math score and years of age in this group is $\hat{\beta}_y \approx 2.18$.

Of course it is always better to actually have experimental control of this kind (provided it is ethical to perform the manipulations), rather than to rely upon statistical controls through multivariate models. But experiments are not always possible or practical. And moreover, no one really cares what happens in an experiment. If our models cannot predict natural observations, then they aren't very useful. Luckily, multivariate models are capable of automatically focusing on only those cases in the data that are most informative. In a sense, they construct control groups, as best they can given the available data. This is a principle reason that regression models of this kind can be so powerful for inferring complex patterns.

Let's consider another example, before turning to worries that arise from adding variables.

4.5.2. Masked influence. The math scores example demonstrates that multiple predictor variables are useful for knocking out spurious association, pretending variables. A second reason to use more than one prediction variable is to reveal the direct influences of multiple factors on an outcome, when none of those influences is apparent from bivariate relationships. This kind of problem can be thought of as a problem of masked influence. It tends to arise when there are two predictor variables that are correlated with one another, however one of these is positively correlated with the outcome and the other is negatively correlated with it.

You'll consider this kind of problem in a new data context. The data in `data(milk)`, part of the LIBRARY that accompanies this book, are information about the composition of milk across primate species, as well as some facts about those species, like body mass and brain size.⁴⁸ Milk is a huge investment, being much more expensive than gestation. Such an expensive resource is likely adjusted in subtle ways, depending upon the physiological and development details of each mammal species. We'll take a glimpse into this world. Let's load the data into R:

```
library(rethinking)
data(milk)
```

R code
4.70

```
d <- milk
str(d)
```

You should see in the structure of the data frame that you have 29 rows (cases) for 8 variables (columns). A popular hypothesis has it that primates with larger brains produce more energetic milk, so that offspring brains can grow quickly. Let's ignore the interesting phylogenetic questions for the moment. If there is no relationship, ignoring phylogeny, then there probably isn't going to be one, once we consider it.⁴⁹

The variables we'll consider are:

- kcal.per.g** : Kilocalories per gram of milk.
- mass** : Female body mass, in kilograms.
- neocortex.perc** : The percent of total brain mass that is neocortex mass.

The question here is to what extent energy content of milk, measured here by kilocalories, is related to the percent of the brain mass that is neocortex. Neocortex is the gray, outer part of the brain that is particularly elaborated in mammals and especially primates. We'll end up needing female body mass, too. But let's hold off on it for a minute.

The first model to consider is the simple bivariate regression between kilocalories and neocortex percent. You know how to set up this regression. Fit the model with:

R code
4.71

```
m1 <- mle2( kcal.per.g ~ dnorm(
  mean=a + bn*neocortex.perc , sd=sigma ) , data=d ,
  start=list(a=mean(dcc$kcal.per.g),bn=0,
  sigma=sd(dcc$kcal.per.g)) )
```

```
Error in optim(par = c(0.657647058823529, 0, 0.172899154558543),
fn = function (p) :
  initial value in 'vmmin' is not finite
```

What has gone wrong here? This particular error message can occur for many reasons. It just means that the likelihood function didn't return a valid likelihood for even the starting parameter values. In this case, the culprit is the missing values in the `neocortex.perc` column. Take a look inside that column and see for yourself:

R code
4.72

```
d$neocortex.perc
```

Each `NA` in the output is a missing value. If you pass a vector like this to a likelihood function like `dnorm`, it doesn't know what to do. After all, what's the likelihood of a missing value? Whatever the answer, it isn't a number, and so `dnom` returns a `NaN`. Unable to even get started, `mle2` (or rather `optim`, which does the real work) gives up and barks about `vmin` not being finite.

This is easy to fix, though. What you need to do here is drop all the cases with missing values. Fancier commands, like `lm` and `glm`, will drop such cases for you. But this isn't always a good thing, if you aren't aware of it. In the next chapter, you'll see one reason why. So indulge me for now. It's worth learning how to do this yourself. To make a new data frame with only complete cases in it, just use:

```
dcc <- d[ complete.cases(d) , ]
```

R code
4.73

This makes a new data frame, `dcc`, that consists of the 17 rows from `d` that have values in all columns. Now let's work with the new data frame. Once again with feeling:

```
m1 <- mle2( kcal.per.g ~ dnorm(
  mean=a + bn*neocortex.perc , sd=sigma ) , data=dcc ,
  start=list(a=mean(dcc$kcal.per.g),bn=0,
  sigma=sd(dcc$kcal.per.g)) )
```

R code
4.74

All that is new there is `data=dcc` instead of `data=d`. Take a look at the estimates now:

```
precis(m1)
```

R code
4.75

	Estimate	Std. Error	2.5%	97.5%
a	0.353331989	0.470779865	-0.569379591	1.27604357
bn	0.004503308	0.006941251	-0.009101293	0.01810791
sigma	0.165702581	0.028419026	0.110002313	0.22140285

Focus on the `bn` row. That is the estimate for the rate of change in kilocalories for each one percent increase in neocortex mass. The estimate is very close to zero—a change from the smallest neocortex percent in the data, 55%, to the largest, 76%, would result in an expected change of only:

```
coef(m1)[“bn”] * ( 76 - 55 )
```

R code
4.76

0.09456946

Less than 0.1 kilocalories. The kilocalories in the data range from less than 0.5 to more than 0.9 per gram, so this effect isn't so impressive. More importantly, it isn't very precise. The 95% confidence interval of the parameter extends about the same distance on both sides of zero. You can plot the predicted mean and 95% prediction interval for the mean to see this more easily:

R code
4.77

```
post <- sample.naive.posterior( m1 )
mu <- sapply( 0:100 , function(z) mean( post$a + post$bn*z ) )
mu.ci <- sapply( 0:100 , function(z) HPDI( post$a + post$bn*z ) )
plot( kcal.per.g ~ neocortex.perc , data=dcc , col="slateblue" )
lines( 0:100 , mu )
lines( 0:100 , mu.ci[1,] , lty=2 )
lines( 0:100 , mu.ci[2,] , lty=2 )
```

I display this plot in the upper-left of Figure 4.20. The maximum likelihood estimate is weakly positive, but it is highly imprecise.

You can fit another bivariate regression, with the logarithm of body mass as the predictor variable now, and you'll find that log-mass is negatively correlated with kilocalories, but also with a rather imprecise confidence interval. Why the logarithm of mass instead of the raw mass in kilograms? It is often true that scaling measurements like body mass are related by magnitudes to other variables. Taking the log of a measure translates the measure into magnitudes, like the way geologists measure earthquakes. So by using the logarithm of body mass here, we're saying that we suspect that the magnitude of a mother's body is related to milk energy, in a linear fashion. I leave it to the reader to replicate the results of the model that regresses kilocalories on log-mass. You should get an estimated slope for `log(mass)` around -0.32 with a standard error of about 0.02. I display this estimate and 95% confidence interval of the mean in the upper-right of Figure 4.20. This influence does seem stronger than that of neocortex percent, although negative.

Now let's see what happens when we add both variables at the same time to the regression. Fitting the joint model is simple enough:

R code
4.78

```
m2 <- mle2( kcal.per.g ~ dnorm(
  mean=a + bn*neocortex.perc + bm*log(mass) , sd=sigma ) ,
  data=dcc ,
  start=list(a=mean(dcc$kcal.per.g),bn=0,bm=0,
             sigma=sd(dcc$kcal.per.g)) )
precis(m2)
```

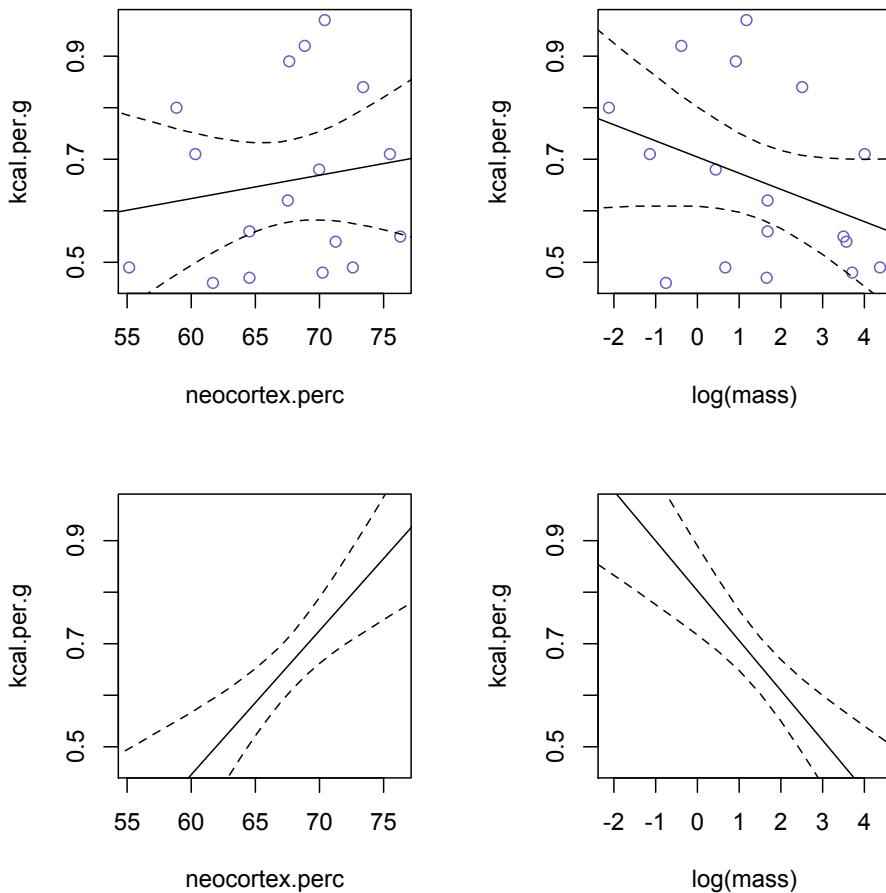


FIGURE 4.20. Milk energy and neocortex among primates. In the top two plots, simple bivariate regressions of kilocalories per gram of milk on (left) neocortex percent and (right) log female body mass show weak and unreliable effects. However, on the bottom, a single regression with both neocortex percent and log body mass suggests strong countervailing effects of both variables. Both neocortex and body mass are associated with milk energy, but in opposite directions, masking one another's effects unless both are considered simultaneously.

	Estimate	Std. Error	2.5%	97.5%
a	-1.08525313	0.467639749	-2.00181020	-0.16869607
bn	0.02793064	0.007274019	0.01367382	0.04218745
bm	-0.09640217	0.022460374	-0.14042370	-0.05238065
sigma	0.11479320	0.019688725	0.07620401	0.15338239

What has happened? Both effects are stronger now. The estimate for the effect of neocortex percent has increased more than 6-fold, and it's 95% confidence interval is now entirely above zero. The estimated effect of body mass is more strongly negative. By incorporating both predictor variables in the regression, the estimated influence of both has increased.

Let's plot the intervals for the predicted mean kilocalories, for this new model. Here's the code for the relationship between kilocalories and neocortex percent. We'll use the mean log body mass in this calculation, showing only how predicted energy varies as a function of neocortex percent:

R code
4.79

```
post <- sample.naive.posterior( m2 )
mean.log.mass <- mean( log(dcc$mass) )
mu <- sapply( 0:100 , function(z)
  mean( post$a + post$bn*z + post$bm*mean.log.mass ) )
mu.ci <- sapply( 0:100 , function(z)
  HPDI( post$a + post$bn*z + post$bm*mean.log.mass ) )
plot( kcal.per.g ~ neocortex.perc , data=dcc , type="n" )
lines( 0:100 , mu )
lines( 0:100 , mu.ci[1,] , lty=2 )
lines( 0:100 , mu.ci[2,] , lty=2 )
```

This plot is displayed in the lower-left of FIGURE 4.20. The analogous plot for `log(mass)` is shown in the lower-right. I leave it to the reader to modify the code above to replicate the plot in the lower-right.

Why did adding neocortex and body mass to the same model lead to larger estimated effects of both? This is a context in which there are two variables correlated with the outcome, but one is positively correlated with it and the other is negatively correlated with it. In addition, both of the explanatory variables are positively correlated with one another. As a result, they tend to cancel one another out. This is another case in which regression automatically finds the most revealing cases and uses them to produce estimates. What the regression model does is ask if species that have high neocortex percent *for their body mass* have higher milk energy. Likewise, the model asks if species with high body mass *for their neocortex percent* have higher milk energy. Bigger species, like apes, have milk with less energy. But species with more neocortex

tend to have richer milk. The fact that these two variables, body size and neocortex, are correlated across species makes it hard to see these relationships, unless we statistically account for both.

Multivariate models are amazing tools. They can resolve problems of impostor variables, like in the case of the student math scores, and they can reveal masked effects, as in this case. But multivariate models aren't magic. In particular, just adding all of the available predictor variables to a model is nearly always a bad idea. Let's consider an example.

4.5.3. When adding variables hurts. It is commonly true that there are many potential predictor variables to add to a regression model. In the case of the primate milk data frame, for example, there are 7 variables available to predict any column we choose as an outcome. Why not just fit a model that includes all 7? Aside from the fact that such an approach is more data dredging than hypothesis evaluation, there are a couple of good, purely statistical reasons to never do this. The first is *multicollinearity*, a nasty word for a simple phenomenon. Once you know to watch for it, it's easy to spot and cope with. We'll confront this problem in this section. The second is the *bias-variance tradeoff*, which is a huge and perspective-changing problem that occupies us for much of the next chapter.

Multicollinearity just means very strong correlation between two or more explanatory variables. The consequence of it is that the estimates will be essentially random and standard errors will be huge, even if all of the variables are strongly correlated with the outcome variable. This will make sense, once you work through an example. So in order to explore the problem of multicollinearity, let's return to the primate milk data from the previous section. Let's get back the original data again:

```
data(milk)
d <- milk
```

R code
4.80

In these data, we have the variables `perc.fat` (percent fat) and `perc.lactose` (percent lactose) that we might use to model the total energy content, `kcal.per.g`. You're going to use these three variables to explore a natural case of multicollinearity.

Start by modeling `kcal.per.g` as a function of `perc.fat` and `perc.lactose`, but in two bivariate regressions. You are a pro at setting up these models by now. Here they are:

```
m1 <- mle2( kcal.per.g ~ dnorm( mean=a + bf*perc.fat , sd=sigma ) ,
  data=d ,
```

R code
4.81

```

start=list(a=mean(d$kcal.per.g),bf=0,sigma=sd(d$kcal.per.g)) )
m2 <- mle2( kcal.per.g ~ dnorm( mean=a + bl*perc.lactose , sd=sigma ) ,
            data=d ,
            start=list(a=mean(d$kcal.per.g),bl=0,sigma=sd(d$kcal.per.g)) )
precis(m1)
precis(m2)

```

	Estimate	Std. Error	2.5%	97.5%
a	0.30113692	0.0356388771	0.231286005	0.37098784
bf	0.01002011	0.0009691012	0.008120709	0.01191952
sigma	0.07326201	0.0096218700	0.054403494	0.09212053

	Estimate	Std. Error	2.5%	97.5%
a	1.16643890	0.0427542037	1.08264220	1.250235595
bl	-0.01057760	0.0008302943	-0.01220495	-0.008950253
sigma	0.06175145	0.0081109925	0.04585420	0.077648701

The maximum likelihood estimate for the effect of percent fat is 0.01, with 95% confidence interval [0.008, 0.012]. The estimate in the second model for percent lactose is -0.01 , with 95% interval $[-0.012, -0.009]$. These estimates are essentially mirror images of one another, with the effect of `perc.fat` being as positive as the effect of `perc.lactose` is negative. Both are rather precise estimates. We might conclude that both variables are reliable predictors of total energy in milk, across species. The more fat, the more kilocalories in the milk. The more lactose, the fewer kilocalories in milk.

Now watch what happens when we place both predictor variables in the same regression model.

R code
4.82

```

m3 <- mle2( kcal.per.g ~ dnorm(
            mean=a + bf*perc.fat + bl*perc.lactose , sd=sigma ) , data=d ,
            start=list(a=mean(d$kcal.per.g),bf=0,bl=0,
                       sigma=sd(d$kcal.per.g)) )
precis(m3)

```

	Estimate	Std. Error	2.5%	97.5%
a	1.007377342	0.199988099	0.615407870	1.399346814
bf	0.001952400	0.002399254	-0.002750052	0.006654853
bl	-0.008708898	0.002438773	-0.013488805	-0.003928991
sigma	0.061057689	0.008019794	0.045339182	0.076776196

Now the estimated effects of both `perc.fat` and `perc.lactose` are smaller, and both estimates now have standard errors more than twice as large as in the bivariate models. In the case of percent fat, the estimated effect

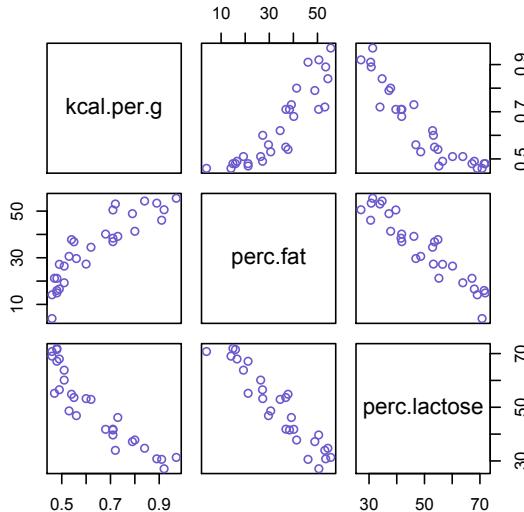


FIGURE 4.21. A pairs plot of the total energy, percent fat, and percent lactose variables from the primate milk data. Percent fat and percent lactose are strongly negatively correlated with one another, providing essentially the same information.

is essentially zero. What has happened here? Are we to conclude that percent fat does not in fact predict energy content of primate milk?

What has happened is that the variables `perc.fat` and `perc.lactose` contain almost entirely the same information. They are substitutes for one another. As a result, when you include both in a regression, the estimates become unreliable. Remember, linear models of this kind use each main effect to ask the question, *how does this variable influence the outcome, holding the other variables constant?* Another way to phrase this question is, *once we know the other variables, does this variable help us predict the outcome?* When two or more predictor variables contain mostly the same information, then the answer to this question will usually be “no.” This will be true even though either variable in isolation is strongly correlated with the outcome. This is not a case of multiple predictor variables helping inference. Instead, here they hurt inference.

In the case of the fat and lactose, these two variables form essentially a single axis of variation. The easiest way to see this is to use a *pairs* plot:

```
pairs( ~ kcal.per.g + perc.fat + perc.lactose ,
      data=d , col="slateblue" )
```

R code
4.83

I display this plot in FIGURE 4.21. This kind of plot is confusing at first, but it is very useful for finding possible cases of multicollinearity. A pairs plot is a square matrix of individual bivariate scatterplots.

Along the diagonal, the variables are labeled. In each scatterplot off the diagonal, the vertical axis variable is the variable labeled on the same row and the horizontal axis variable is the variable labeled in the same column. For example, the two scatterplots in the first row in FIGURE 4.21 are `kcal.per.g` (vertical) against `perc.fat` (horizontal) and then `kcal.per.g` (vertical) against `perc.lactose` (horizontal). Notice that percent fat is positively correlated with the outcome, while percent lactose is negatively correlated with it. Now look at the rightmost scatterplot in the middle row. This plot is the scatter of percent fat (vertical) against percent lactose (horizontal). Notice that the points line up almost entirely along a straight line. These two variables are negatively correlated, and so strongly so that they are nearly redundant. Either helps in predicting `kcal.per.g`, but neither helps once you already know the other.

You can compute the correlation between the two variables with the `cor` command:

R code
4.84

```
cor( d$perc.fat , d$perc.lactose )
```

[1] -0.9416373

That's a pretty strong correlation. How strong does a correlation have to get, before you should start worrying about multicollinearity? Correlations do have to get pretty high before this is a serious problem. You can prove this to yourself with a little simulation experiment. Suppose we have only `kcal.per.g` and `perc.fat`. Now we construct a random predictor variable, call it `x`, that is correlated with `perc.fat` at some predetermined level. Then fit the regression model that tries to predict `kcal.per.g` using both `perc.fat` and our random fake variable `x`. Record the standard error of the estimated effect of `perc.fat`. Now repeat this procedure many times, at different levels of correlation between `perc.fat` and `x`.

If you do this, you'll get a plot like that in Figure 4.22. The vertical axis is the average standard error across 100 regressions using simulated correlated predictor variables `x`. The horizontal axis shows the intensity of correlation between `x` and `perc.fat`. When the two variables are uncorrelated, on the left side of the plot, then the standard error of the estimate is small. This means the estimate is fairly precise. As the correlation increases—and keep in mind that we aren't adding any information here, just a correlated string of random numbers—the standard error inflates. But the effect is far from linear; it accelerates rapidly, as the correlation increases. Above a correlation of 0.9, the standard

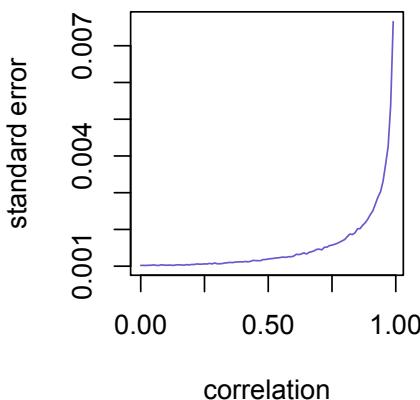


FIGURE 4.22. The effect of correlated predictor variables on the precision of estimates. The vertical axis shows the standard error of the estimated effect of speed on braking distance. The horizontal axis is the correlation between speed and another variable added to the model. As the correlation increases, the standard error inflates.

error increases very rapidly, approaching in fact ∞ as the correlation approaches 1. The code for producing this plot is below.

```

sim.coll <- function( r=0.9 ) {
  require(MASS)
  d$x <- rnorm( nrow(d) , mean=r*d$perc.fat ,
    sd=sqrt( (1-r^2)*var(d$perc.fat) ) )
  m <- lm( kcal.per.g ~ perc.fat + x , data=d )
  sqrt( diag( vcov(m) ) )[2] # std err of perc.fat coefficient
}
rep.sim.coll <- function( r=0.9 , n=1000 ) {
  se <- replicate( n , sim.coll(r) )
  mean(se)
}
r.seq <- seq(from=0,to=0.99,by=0.01)
se <- sapply( r.seq , function(z) rep.sim.coll(r=z,n=100) )
plot( se ~ r.seq , type="l" )

```

R code
4.85

Why should the standard error be infinite, when the correlation is 1? When two variables are perfectly correlated, they contain exactly the same information. Inside the equation for the mean of a normal distribution, this creates a very weird result. Suppose for example that we have two perfectly correlated predictor variables x_1 and x_2 . Both are really the same list of numbers. So when you place both into a linear

regression as main effects, you are saying that the mean is:

$$\mu_i = \alpha + \beta_1 x_{1i} + \beta_2 x_{2i},$$

but since $x_1 = x_2 = x$, this is just the same as:

$$\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i = \alpha + (\beta_1 + \beta_2) x_i.$$

Now when you try to find the maximum likelihood estimates for β_1 and β_2 , things will go horribly wrong. Suppose the maximum likelihood estimate for the variable x , entered in the model alone, is $\hat{\beta}$. This means that when we include both x_1 and x_2 —just like including x twice—the computer will now try to find a sum $\beta_1 + \beta_2$ that is equal to $\hat{\beta}$. But there will be an infinite number of combinations $\beta_1 + \beta_2$ that will equal $\hat{\beta}$, whatever its value! This is why the standard error of the estimates for β_1 and β_2 will be infinite, because there will be infinite uncertainty in them. Not only will β_1 and β_2 have infinite (or near infinite) standard errors, but the two parameters will be strongly negatively correlated, because as one increases, the other must decrease to maintain the sum at the same value.

When the variables in question are not perfectly correlated, as in the primate milk data, then this problem is ameliorated somewhat. But the same phenomenon explains the increase in standard error, before it explodes to infinity. You can see the symptom of this problem, the very large range of paired values that sum to the same value, when you inspect the posterior distribution of the parameters. Let's do this now for the model `m3` above.

R code
4.86

```
post <- sample.naive.posterior( m3 )
cor(post)
```

	a	bf	bl	sigma
a	1.00000000	-0.97712289	-0.98885012	0.01056465
bf	-0.97712289	1.00000000	0.94121578	-0.01050994
bl	-0.98885012	0.94121578	1.00000000	-0.01072272
sigma	0.01056465	-0.01050994	-0.01072272	1.00000000

(You might also try a pairs plot of the posterior, `pairs(post)`, to help you visualize these correlations.) The distributions of `bf` (the parameter corresponding to `perc.fat`) and `bl` (the parameter corresponding to `perc.lactose`) are correlated 0.94. When two parameters are this strongly correlated, it means that when one changes, the other changes. They are nearly redundant. This is commonly true of the intercept, a/α , in a linear regression, which is mostly harmless. But when it is also true

of so-called beta-coefficients, the slopes on factors, it is far from harmless.

What can be done about multicollinearity? The best thing to do is be aware of it. You can anticipate this problem by checking the predictor variables against one another in a pairs plot. Any pair or cluster of variables with very large correlations, over about 0.9, may be problematic, once included as main effects in the same model. However, it isn't always true that highly-correlated variables are completely redundant. If you look back at the student math scores example (Section 4.5.1, page 192), you'll see that those variables are also very highly correlated with one another. However, in that case they are uncorrelated in exactly the right way so that `years` provides information that `height` does not. So you can't know just from a table of correlations nor from a matrix of scatterplots whether multicollinearity will prevent you from including sets of variables in the same model. The best way to diagnose this problem is just to look for a huge inflation in standard errors, when both variables are included in the same model.

Different disciplines have different conventions for dealing with collinear variables. In some fields, like large portions of psychology, it is typical to engage in some kind of data reduction procedure, like *principal components* or *factor analysis*, and then to use the components/factors as predictor variables. In other fields, like my own, this is considered voodoo, because factors are notoriously hard to interpret, and there are always hidden arbitrary decisions that go into producing them. It is also difficult to use such constructed variables in prediction. A defensive approach is to show that models using any one member from a cluster of highly correlated variables will produce the same inferences and predictions. For example, sometimes rainfall and soil moisture are highly correlated across sites in an ecological analysis. Using both in a model meant to predict presence/absence of a species of plant may run afoul of multicollinearity, but if you can show that a model with either produces nearly the same predictions, it will be reassuring that both get at the same underlying ecological dimension.

The problem of multicollinearity is really a member of a family of problems with fitting models, a family sometimes known as *non-identifiability*. When a parameter is non-identifiable, it means that the structure of the data and model do not make it possible to estimate the parameter's value. Another very common source of non-identifiable parameters occurs in the context of categorical variables. So that's where we'll turn next.

4.6. Categorical Variables

A common question for statistical methods is to what extent an outcome changes as a result of an intervention or the presence or absence of a factor. For example, consider the different species in the milk energy data again. Some of them are apes, while others are New World monkeys. We might want to ask how predictions should vary when the species is an ape instead of a monkey.

Most readers will already know that variables like this can be included in linear models. But what is not widely understood is how these variables are included in a model, because the computer does all of the work for us most of the time. There are some subtleties here that are worth considering. After all, a predictor variable like `mass` is a measure. Including it in the equation for μ_i is straightforward. But a variable like `clade`, which has one of four possible values, is trickier. Only numbers can be included in the equation for μ_i , first of all. So you might assign each unique value in `clade` a unique number, like:

clade	number
Strepsirrhine	1
New World Monkey	2
Old World Monkey	3
Ape	4

But this won't do, because it implies that apes are somehow 4-times more something than are Strepsirrhines. In reality, these categories have no order to them at all. We just want to be able to distinguish them from one another, assigning each unique case to a unique clade. And we want to allow for prediction in each clade to be different from the other clades.

4.6.1. Dummy variables. The universal solution to this kind of problem is to break up the single categorical variable `clade` into a series of *dummy* variables. These variables are also known as *indicator* or *flag* variables. Here's the basic notion. Suppose you add a variable to the data frame that takes the value 1 when the species is a New World monkey and takes the value 0 otherwise. Let's add this variable to the data frame:

R code
4.87

```
d$clade.NWM <- ifelse( d$clade=="New World Monkey" , 1 , 0 )
d$clade.NWM
```

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

So only those rows (cases) in which `clade` indicates a New World monkey get a 1 for this new variable. Now if we include the above dummy

variable in the equation for μ_i :

$$\mu_i = \alpha + \beta c_{1i},$$

where c_{1i} is an arbitrary mathematical name for the dummy clade variable you just created. This equation only predicts two different means. First, when the case in question i is not a New World monkey, then $c_{1i} = 0$. The mean prediction becomes:

$$\mu_i = \alpha + \beta \times 0 = \alpha.$$

Second, when the case in question is indeed a New World monkey, then $c_{1i} = 1$. Then the mean prediction becomes:

$$\mu_i = \alpha + \beta \times 1 = \alpha + \beta.$$

In effect, a case that is a New World monkey has a constant amount β added to its mean prediction. In other words, the category ends up with a different intercept. So it would be better to say perhaps:

$$\mu_i = \alpha + \alpha_{\text{NWM}} c_{1i}.$$

The parameter α_{NWM} now stands for the difference between the average of New World monkeys and the other clades.

4.6.2. Dummy variables and the danger of non-identifiability. This approach extends easily to more categories. We'd like to distinguish the other three clades as well. It turns out that we only need 3 dummy variables, to distinguish 4 unique categories from one another. In general, if there are n categories, then we only need $n - 1$ dummy variables. Why only $n - 1$? Here comes the easy mistake that can result in *non-identifiable* parameters (see previous section). Suppose you just charge in and make a dummy variable for every unique clade, and then include all of them in the model, with an equation for the mean:

$$\mu_i = \alpha + \alpha_{\text{NWM}} c_{1i} + \alpha_{\text{OWM}} c_{2i} + \alpha_{\text{SC}} c_{3i} + \alpha_{\text{A}} c_{4i}.$$

This will not go well for us. The reason is that we've stumbled into trying to estimate a sum again, and there will be a very large number of solutions to this problem. The mistake was to include both α , an intercept that applies to all cases, as well as a unique intercept for each clade. So the predicted means for each clade will look like:

clade	μ_i
Strepsirrhine	$\alpha + \alpha_S$
New World Monkey	$\alpha + \alpha_{\text{NWM}}$
Old World Monkey	$\alpha + \alpha_{\text{OWM}}$
Ape	$\alpha + \alpha_A$

What's wrong with this? Since α is always added, there are no cases that allow us to estimate it uniquely. Depending upon the algorithm you use to fit the model, various bad things can happen. Most commonly, you end up with huge standard errors and somewhat random estimates for the intercepts. For example, if you fit the model with `mle2`, predicting `kcal.per.g`:

R code
4.88

```
mBAD <- mle2( kcal.per.g ~ dnorm(
  mean=a + a.NWM*clade.NWM + a.OWM*clade.OWM + a.S*clade.S
  + a.A*clade.A , sd=sigma ) , data=d ,
  start=list(a=0,a.NWM=0,a.OWM=0,a.S=0,a.A=0,
  sigma=sd(d$kcal.per.g)) )
precis(mBAD)
```

	Estimate	Std. Error	2.5%	97.5%
a	0.511280744	0.62906574	-0.72166544	1.7442269
a.NWM	0.203166868	0.63022032	-1.03204225	1.4383760
a.OWM	0.277061255	0.63082449	-0.95933203	1.5134545
a.S	-0.003268066	0.62695491	-1.23207711	1.2255410
a.A	0.034320539	0.63022971	-1.20090699	1.2695481
sigma	0.114506159	0.01503628	0.08503559	0.1439767

Look at those standard errors. They are large enough to create considerable uncertainty about the sign of every estimated α parameter. Furthermore, they look odd, because they are all about the same magnitude, even though some clades have more species in them than the others, and therefore should enjoy more precise estimates. And the estimates for all of the α parameters are also very strongly correlated:

R code
4.89

```
round( cov2cor( vcov(mBAD) ) , digits=3 )
```

	a	a.NWM	a.OWM	a.S	a.A	sigma
a	1.000	-0.998	-0.997	-0.997	-0.998	0
a.NWM	-0.998	1.000	0.995	0.995	0.996	0
a.OWM	-0.997	0.995	1.000	0.994	0.995	0
a.S	-0.997	0.995	0.994	1.000	0.995	0
a.A	-0.998	0.996	0.995	0.995	1.000	0
sigma	0.000	0.000	0.000	0.000	0.000	1

These massive correlations are exactly the kind of thing that happens, when you have more parameters than you can estimate.

Let's fit the model again, now dropping the dummy variable for apes. What this does is *alias* the general intercept α to be the mean prediction for apes. Effectively, this makes Ape the baseline clade against which

the others are estimated. The other α parameters will then be differences between the prediction for an ape and the other clades. It doesn't matter which category you assign to the overall intercept in this way. Fitting the model again, now without α_A :

```
mOK <- mle2( kcal.per.g ~ dnorm( R code
  mean=a + a.NWM*clade.NWM + a.OWM*clade.OWM + a.S*clade.S ,
  sd=sigma ) , data=d ,
  start=list(a=0,a.NWM=0,a.OWM=0,a.S=0,
  sigma=sd(d$kcal.per.g)) )
precis(mOK)
```

	Estimate	Std. Error	2.5%	97.5%
a	0.5455563	0.03816936	0.47074576	0.62036692
a.NWM	0.1688877	0.05397965	0.06308955	0.27468588
a.OWM	0.2427752	0.06035113	0.12448916	0.36106125
a.S	-0.0375585	0.06386906	-0.16273957	0.08762256
sigma	0.1145083	0.01503697	0.08503633	0.14398019

That's better. Now each of the category intercepts is more precisely estimated. Also notice that the previous unreliable estimates add up to be nearly the same as these better estimates. For example, the prediction from model `mBAD` for the mean `kcal.per.g` for an ape was $a+a.A \approx 0.51+0.03 = 0.54$. This is about the same as the predicted mean from the newer, more precise model: $a \approx 0.55$. You can verify that the other estimates share the same relationship. This is a symptom of R estimating the sum, in `mBAD`, with tremendous uncertainty in the components of that sum. What we've gained is much much more precision about the estimates and a much easier time interpreting the meanings of the parameters.

Before moving on to add other predictor variables to the model, let's consider how to plot predictions for a model like this, in which all of the prediction variables are categorical. First, let's sample from the naive posterior, as usual. As always, this means we are assuming the joint posterior is multivariate Gaussian and that it is derived from a non-influential prior.

```
post <- sample.naive.posterior( mOK )
```

R code
4.91

Now we want to compute the predicted mean, μ , for each clade. This is actually easier than the tasks you've used the posterior for in earlier examples. To compute the mean prediction for an ape:

R code
4.92

```
mu.A <- mean( post$a + post$a.NWM*0 + post$a.OWM*0 + post$a.S*0 )
```

What I've done here is just compute one μ per sample from the posterior, using the equation for μ_i . But I've also assigned zeros for the dummies for NWM, OWM and Streps. This leaves only α , a . Then the code finds the average of all of those μ_i values. The result is the expected mean milk energy of an ape. The other three clades just replace one of the 0's in the code above with a 1:

R code
4.93

```
mu.NWM <- mean( post$a + post$a.NWM*1 + post$a.OWM*0 + post$a.S*0 )
mu.OWM <- mean( post$a + post$a.NWM*0 + post$a.OWM*1 + post$a.S*0 )
mu.S <- mean( post$a + post$a.NWM*0 + post$a.OWM*0 + post$a.S*1 )
```

Notice that the dummy variable for NWM is set to 1 in the first line of code, for `mu.NWM`. The next two lines place the 1 in another place, the appropriate dummy variable for that category. At the end, we have four mean predictions, one for each clade.

You'll also want confidence intervals, as always. All you really need to do here is use `HPDI` instead of `mean`:

R code
4.94

```
mu.ci.A <- HPDI(
  post$a + post$a.NWM*0 + post$a.OWM*0 + post$a.S*0 )
mu.ci.NWM <- HPDI(
  post$a + post$a.NWM*1 + post$a.OWM*0 + post$a.S*0 )
mu.ci.OWM <- HPDI(
  post$a + post$a.NWM*0 + post$a.OWM*1 + post$a.S*0 )
mu.ci.S <- HPDI(
  post$a + post$a.NWM*0 + post$a.OWM*0 + post$a.S*1 )
```

And just to be thorough, let's do actual prediction intervals too, this time. That is, let's compute the 95% interval for predicted species milk energies, not just the mean milk energy. As earlier in the book, the easiest way to do this is to just leverage the `rnorm` command to sample from the implied normal distribution of milk energies.

R code
4.95

```
n <- 10 * nrow( post )
mu.pi.A <- HPDI( rnorm( n ,
  mean=post$a + post$a.NWM*0 + post$a.OWM*0 + post$a.S*0 ,
  sd=post$sigma ) )
mu.pi.NWM <- HPDI( rnorm( n ,
  mean=post$a + post$a.NWM*1 + post$a.OWM*0 + post$a.S*0 ,
```

```

sd=post$sigma ) )
mu.pi.OWM <- HPDI( rnorm( n ,
  mean=post$a + post$a.NWM*0 + post$a.OWM*1 + post$a.S*0 ,
  sd=post$sigma ) )
mu.pi.S <- HPDI( rnorm( n ,
  mean=post$a + post$a.NWM*0 + post$a.OWM*0 + post$a.S*1 ,
  sd=post$sigma ) )

```

Note that the standard deviation in each case is also a sample from the naive posterior.

Now to plot everything. What you'll display here is a plot with clades on the horizontal axis and milk energy on the vertical axis. To accomplish this, you need to set up the clades as numeric ordered values, just for the sake of plotting. Then it's easy as always to plot the individual species milk energies.

```

clade.num <- c( rep(4,5) , rep(2,9) , rep(3,6) , rep(1,9) ) - 0.15
plot( d$kcal.per.g ~ clade.num , col="slateblue" ,
  ylab="mean kcal.per.g" , xlab="clade" , ylim=c(0.25,1.05) ,
  xlim=c(0.75,4.25) , xaxt="n" )
axis( 1 , at=1:4 , labels=c("Ape","NWM","OWM","Strep") )

```

R code
4.96

The code uses `clade.num` as a stand-in for clade, to provide a horizontal coordinate for each category. These are then offset by -0.15 , so the data points won't overlap the intervals you are about to add to the plot. The only other new trick here is the use of `axis` to specify arbitrary labels for the horizontal axis. The option `xaxt="n"` in the call to `plot` suppresses the horizontal axis, and then the call to `axis` in the last line draws it, using the clade labels.

Now display the predicted means for each category:

```
points( 1:4 , c(mu.A,mu.NWM,mu.OWM,mu.S) , pch=16 )
```

R code
4.97

And then display the confidence intervals of these means:

```

lines( c(1,1) , mu.ci.A , lwd=3 )
lines( c(2,2) , mu.ci.NWM , lwd=3 )
lines( c(3,3) , mu.ci.OWM , lwd=3 )
lines( c(4,4) , mu.ci.S , lwd=3 )

```

R code
4.98

And finally the prediction intervals are drawn:

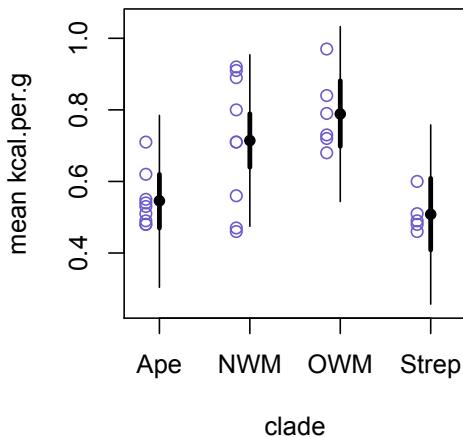


FIGURE 4.23. Dummy variables and model predictions. The blue points are milk energies for individual primate species, grouped by clade. The solid black points are predicted means for each clade, the thick line segment around the mean is the 95% interval of the mean, and the thin line around that is the 95% prediction interval.

R code
4.99

```
lines( c(1,1) , mu.pi.A )
lines( c(2,2) , mu.pi.NWM )
lines( c(3,3) , mu.pi.OWM )
lines( c(4,4) , mu.pi.S )
```

Figure 4.23 shows the result of all of this work. Each blue point in the plot is a species. The solid black points are the predicted means for each clade. The thick line segment around each predicted mean is the 95% interval of that mean. The thin line segment around that is the 95% prediction interval for milk energy.

4.6.3. Adding regular predictor variables. There is no obstacle now in including other predictor variables, like `perc.fat` or `log(mass)`. Just add them to the equation for the mean, as you normally would. For example, to add the influence of `perc.fat` to the model with dummies for the clades:

$$\mu_i = \alpha + \alpha_{NWM}c_{1i} + \alpha_{OWM}c_{2i} + \alpha_S c_{3i} + \beta_F F_i,$$

where F_i is the percent fat for the i -th case in the data and β_F is the slope that measures the influence of F on predictions about the mean milk energy. The key insight here is that dummy variables adjust intercepts, and their parameters are *marginal intercepts*. Continuous predictor variables like F in contrast create variable effects, depending upon the value F_i for each case.

When you reach the chapter on interactions, we'll return to the relationship between dummy variables and continuous predictors. Since dummy variables turn some fixed adjustment on or off for each row i , they can be used to produce all manner of contingent predictions, depending upon category memberships. Among these contingent effects is the adjustment of slopes themselves, allowing different categories in the data to bear different relationships to a predictor variable.

[Possibly add !Kung data example of height on age sex to show mix of continuous and categorical variables.]

4.7. Centering Variables

Introduce centering here or wait until chapter on interactions? Explain using scale command; numerical advantages; interpretation advantages.

4.8. Ordinary Least Squares

Many readers will know single node Gaussian regression models by another label, *ordinary least squares* regression, or OLS for short. OLS is a way of estimating the parameters of a linear regression. Instead of searching for the combination of parameter values that maximizes the likelihood, OLS instead solves for the parameter values that minimize the sum of the squared residuals. When the variance around the mean is symmetrical, as it is with Gaussian distributions, and the mean itself is strictly an additive combination of terms of the form *parameter* \times *data*,⁵⁰ then OLS is a simple and fast way to compute the maximum likelihood estimates. It is extremely useful and one of the key reasons that additive Gaussian models are so common.

But it is a disservice to students to teach OLS first, as many courses do. OLS is a special case of maximum likelihood. All of the non-Gaussian models presented in later chapters—generalized linear models—rely upon maximum likelihood. So too does linear regression. That is why I presented everything about Gaussian regression in terms of maximum likelihood, first. But OLS is extremely powerful, when it is appropriate. So it's worth learning, but only with the knowledge that it is providing estimates of the naive posterior, through its descendent relationship to maximum likelihood estimation.

In this section, I explain the mechanics of OLS and its connection to maximum likelihood. Certainly there are other ways to justify OLS, but the maximum likelihood justification is the only one that provides a cogent link to Bayesian inference about the posterior. And so it is the most important.

4.8.1. Fitting a model with OLS. The most common and simplest way to fit a Gaussian linear model with OLS is to use the `lm` command. The syntax for this command is rather different from the syntax for a more flexible command like `mle2`. I prefer to teach Gaussian models using `mle2`, because it forces one to specify all the parameters and their relations to the data. Using `lm` makes it rather invisible how the parameters enter into the model, and this is no bueno for learning. If you don't know what the statistical model is, then you will be prone to using the model in silly contexts.

Now that you've suffered through `mle2` for a while now, it's safe to see the `lm` shortcut. Suppose we have the Gaussian model:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta_1 x_i + \beta_2 z_i, \end{aligned}$$

where x and z are just two predictor variables. To fit this model with `mle2`, you have to define all of the parameters and their starting values. To fit the same model, using OLS, this is all that is needed:

```
lm( y ~ x + z , data=d )
```

The `lm` command does not require nor even allow you to name the parameters. But it requires a lot less input.

Where is the intercept in `lm`? Technically, it's still there. These two lines are functionally identical:

```
lm( y ~ x + z , data=d )
lm( y ~ 1 + x + z , data=d )
```

And if you want to fit a linear model with no predictor variables, just the intercept, use syntax like:

```
lm( y ~ 1 , data=d )
```

Finally, if you want to force the model to have no intercept at all, use:

```
lm( y ~ 0 + x + z , data=d )
```

That zero on the righthand side of the model formula means "make the intercept exactly zero." This effectively removes α from the model, fitting a model like:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \beta_1 x_i + \beta_2 z_i. \end{aligned}$$

If you have a categorical variable to include, `lm` will happily compute the dummy variables for you, provided you are careful to tell it that the variable is categorical. For example:

```
lm( y ~ x + z + as.factor(clade) , data=d )
```

will add a categorical variable `clade` to the model. You can make and add dummy variables yourself, of course. But most of the built-in commands like `lm` will try their best to manufacture the dummies for you.

The `lm` command is quite powerful, and it has a bunch of optional abilities, such as weights. More importantly, the reduced syntax that `lm` uses to specify linear models is quite standard, across many R commands. So you'll see it again later. Keep in mind that while the parameters and intercept are hidden in such an abbreviated formula, the model that is fit the data does always include them. In order to plot predictions from the fit model, you still need to know how the parameters relate to the data and to one another.

4.8.2. How OLS works. OLS finds the parameter estimates that minimize the sum of the squared residuals. The residuals are just the differences between the actual outcome values y and the predicted means μ . That is, for each row i (or case) in the data, the regression model predicts an average. This average comes from the equation for the mean, μ_i . Subtract that predicted mean μ_i from the actual value y_i to get a residual r_i for row i . We did this already in Section 4.5.1, page 198. Now square all of these residuals and add them up to get the *sum of squared residuals*: $\sum_i r_i^2$. This quantity is what OLS minimizes.

It turns out that with a little mathematics, it's possible to derive a formula that yields the unique combination of parameter values that will minimize this sum of squares. The matrix algebra involved isn't all that advanced, but you can look it up in any number of excellent mathematical statistics textbooks. You'll find a solution looking something like:

$$\hat{\mathbf{B}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y},$$

where $\hat{\mathbf{B}}$ is the vector of estimated coefficients, \mathbf{X} is a matrix with explanatory variables as columns and cases as rows (like a data frame) and \mathbf{y} is a vector of the outcomes. In R code, this formula is:

```
solve( t(X) %*% X ) %*% t(X) %*% y
```

R code
4.100

This looks pretty strange, but thankfully it's not important to understand everything about this line of code. I put it here to demystify the operation of `lm` a little—it's just using a formula. An example will help. To replicate the OLS estimates for the braking distance data built into R:

```
data(cars)
X <- cbind( rep(1,nrow(cars)) , cars$speed )
```

R code
4.101

```
y <- cars$dist
solve( t(X) %*% X ) %*% t(X) %*% y
```

```
[,1]
[1,] -17.579095
[2,]  3.932409
```

The only trick of note here is that the intercept, α , needs to be added to X as a column of 1's. That's what the `rep(1, nrow(cars))` is about. The *design matrix* approach that treats the intercept in this way is still common in some fields, because of the historical dominance of linear models and OLS. In one department at my university, graduate students actually include design matrixes like this in their dissertation proposals. Compare the above output to that of using `lm` to do the fit:

R code
4.102

```
coef( lm( dist ~ speed , data=cars ) )
```

```
(Intercept)      speed
-17.579095    3.932409
```

This formula is one important reason why OLS is so powerful and useful. Instead of searching through likelihood space for these estimates, they can be found straight away with a formula. Now, there are a lot of tricks involved in coding the numerical algorithms hidden inside R to make the magic happen, but that doesn't diminish the power and convenience of estimating parameters in this way. Once we move to non-linear, non-additive models, there will not in general be any formula of this kind that provides unique maximum likelihood estimates of the parameters. In these cases, the most common solutions are the likelihood search approaches you encountered in Chapter 2 and so-called Monte Carlo methods that don't maximize anything, but instead sample from distributions. You'll meet Monte Carlo estimation in Chapter 11.

4.8.3. OLS as a special case of maximum likelihood. There are a number of ways to justify this procedure. However, the justification we are interested in is the only one that allows us to calculate the naive posterior densities of the parameters. Provided that the outcome is normally distributed and has constant variance, OLS is just a special case of maximum likelihood estimation. You can find a proof of this in many places, so I'm not going to replicate the proof here.

The intuition you need is just that the negative log-likelihood function for the Gaussian distribution is proportional to the squared distance

between the actual outcomes and the predictions, $\sum_i(y_i - \mu_i)^2$. So minimizing this is like minimizing the squared residuals.

4.8.4. Using OLS to find starting values. If you fit a linear Gaussian model using OLS, you don't get much information about σ , the standard deviation of the normal distribution. But if you care about prediction, you want some information about uncertainty in σ , so you need to be explicitly likelihood-based. There's no problem using a command like `mle2` to fit a Gaussian model. That will give you a naive posterior that includes σ .

However, this introduces the problem of finding good starting values for the parameters, and sometimes maximum likelihood has trouble finding the estimates that OLS provides, because it must search where OLS just solves. There's nothing stopping us from starting with the OLS solution, however, and then adjusting it as we estimate σ at the same time as every other parameter, using `mle2`.

The hard way to do this would be just to fit the model first with `lm` and then manually entering the estimates into the `start` list for `mle2`. The LIBRARY of code that accompanies this book provides a command that essentially does exactly that, but saves you a lot of typing.

Function `lm.to.mle2`:

```
m.lm <- lm( dist ~ speed , data=d )
m.mle2 <- lm.to.mle2( m.lm , data=d )
```

R code
4.103

4.9. Goodbye Sigma, Hello Tau

4.9.1. The typical Gaussian formula. You've seen the likelihood function—also known as a density—for the Gaussian distribution before:

$$L(y_i|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \mu)^2}{2\sigma^2}\right).$$

This thing looks pretty fearsome, I know. But it's really a very simple distribution and the formula above is easy to remember, once you know what each piece is about. There are two main parts: (1) the normalizing constant in front and (2) the exponential shape in back.

The normalizing constant, $1/\sqrt{2\pi\sigma^2}$, is just there so that the entire probability density—the area under the bell curve—always sums to unity. It has to sum to unity, because otherwise it wouldn't be a correct, coherent probability density. The probabilities of all events must some

to 1, or else math goes boom. So this weird looking normalizing constant is found by solving for the expression that makes the integral of the exponential part equal to 1.

Why is π in there? A satisfying mathematical answer to that question would take a lot more space. The casual and not-entirely-wrong answer is that the constants e and π , as well as $i = \sqrt{-1}$, are deeply related to one another through the unit circle that you learned in secondary school. So it's common for π to pop up in an analysis that involves the area under an exponential.

Why 2π ? That's because π isn't actually a fundamental measure of the unit circle. Instead, twice π is. I place this comment here, because it always incites a fit in someone, and yet it is manifestly true. Our mathematics are littered with pointless factors of 2, because of an historical accident.⁵¹ The take away point for the reader is just that you shouldn't be surprised to find 2π more often than only a single π .

The more interesting and useful part is the exponential shape in back. This quadratic exponential is what gives the normal distribution its bell shape. The squared distance from the mean, $(y_i - \mu)^2$, does the work. The negative out in front makes the curve go down, rather than up. And the double variance on the bottom, $2\sigma^2$, just stretches the bell horizontally. It scales it.

4.9.2. Precision instead of deviation. The traditional form of the normal density above uses σ as a measure of the magnitude of error around the mean. It is a measure of how much values *deviate* from the mean. Another way to think about this distribution is to instead think about the *precision*, how tightly values cluster around the mean. The precision is just the inverse of the deviation.

It turns out that there is a common alternative parameterization of the normal that just substitutes $\sigma = 1/\tau$, where τ is called the *precision*. A normal density with low precision, small τ , will have a wide variance. A normal distribution with large precision, large τ , will have a narrow variance. An infinitely precise distribution will have zero variance.

Once we substitute $\sigma = 1/\tau$ and simplify a bit, the density function becomes:

$$L(y_i|\mu, \tau) = \frac{\tau}{\sqrt{2\pi}} \exp\left(-\frac{\tau^2(y_i - \mu)^2}{2}\right).$$

This is the same formula, really. It is possible to focus on this formula and pull some insight from it about the nature of the Gaussian density, but that sort of discussion is a bit out of the scope of this book.

Instead, I'd like to focus on the incredibly useful work that the τ version of the normal density can do for us. Usually, it helps us address two problems with estimating Gaussian models. Therefore, it is usually to be preferred to the σ parameterization. The first problem is a numerical one, a problem with how computers do math. The second problem relates to the expected skew in the naive posterior of σ that you encountered earlier in this chapter. There are other advantages of τ , such as its natural relationship to the exponential family of distributions. But it is its pragmatic advantages that motivate us here.

4.9.3. Numerical advantages of τ .

4.9.4. Better estimates of σ , using τ . Earlier in this chapter (page 153), you saw that the naive posterior density for σ won't be normal, when the sample size isn't very large. This causes problems when we seek to estimate the naive posterior by using quadratic approximations of its shape. If the posterior isn't nearly normal, then the quadratic approximation can produce bad estimates. In particular, it will tend to underestimate the uncertainty about how large σ might be. This in turn will affect predictions derived from the model, as well as the confidence intervals or HPDI's. And if the maximum likelihood estimate of σ is close to zero, then the quadratic estimate is a standout disaster—it will extend the naive posterior into negative numbers, something that isn't possible for σ . More typically, the quadratic estimate will vastly underestimate the extent of the right tail of the density.

Now, the quadratic estimate is just an estimate, and if you aren't too concerned with variance parameters, then you can often get away with ignoring these problems. But there's actually no reason to put up with this issue, because it turns out that τ can rescue us, to a large extent. What I am going to explain now is how finding the maximum likelihood estimate of τ actually provides for a naturally skewed estimate of the naive posterior of σ . Later in the book, when you see how to do MCMC estimation, this trick won't be necessary, but it will still help address the numerical computation issues.

Here's the basis of the trick. A bunch of mathematics suggests that an idealized distribution for the naive posterior of σ should be an inverse-Gaussian.⁵² What the heck is that? An inverse-Gaussian density is just the distribution of random values drawn from a Gaussian and then inverted. Inversion in mathematics can mean several different things, but in a general context it usually means dividing one by a value to compute the value's reciprocal. So if you have a bunch of Gaussian random

numbers, you just compute the reciprocal of each. The resulting values will have an inverse-Gaussian density.⁵³

Now here's the magic trick. If σ is expected to have an inverse-Gaussian naive posterior, then τ should have a regular Gaussian naive posterior. This is because $\tau = 1/\sigma$. If you invert an inverse density, you get the plain density back. So now if you estimate τ instead of σ when you use maximum likelihood estimation, and then estimate the naive posterior with the handy quadratic approximation, you get an estimated Gaussian naive posterior for τ . Theory tells us that τ should be approximately Gaussian, so this is good news. Then you can invert that naive posterior to get an appropriately skewed posterior for σ ! It may seem like a trick, but it's actually supported by a bunch of formal analysis.

Here's how it works.

Let's return to the height example from earlier in the chapter. Sampling 20 heights again, we get a reduced set of data that you've already seen should have a skewed naive posterior for σ :

fix this too when revising chapter to use !Kung census data.

R code
4.104

```
y <- rnorm( 20 , mean=1 , sd=1 )
```

Now find the maximum likelihood estimates and sample from the naive posterior, with quadratic approximation, as you've done several times in this chapter:

R code
4.105

```
msigma <- mle2( y ~ dnorm( mean=a , sd=sigma ) , data=list(y=y) ,
  start=list(a=mean(y),sigma=sd(y)) )
post1 <- sample.naive.posterior( msigma )
```

Take a look at the density for `post1$sigma`. It's approximately Gaussian, because you assumed it was. But it shouldn't be, as you've already seen in this chapter.

Now let's do the maximum likelihood estimation over again, using τ instead of σ . You can actually use almost exactly the same code. Here's a working model:

R code
4.106

```
mtau <- mle2( y ~ dnorm( mean=a , sd=1/tau ) , data=list(y=y) ,
  start=list(a=mean(y),tau=1/sd(y)) )
```

All I've done is set the standard deviation inside the Gaussian density function to $1/\tau$, which is equal to σ . Then in the `start` list of parameter values, I set the initial guess to the inverse standard deviation.

Now let's finish up by sampling from the naive posterior again and showing the density for σ that the estimates in `mtau` imply.

```
post2 <- sample.naive.posterior( mtau )
post2$sigma <- 1/post2$tau
```

R code
4.107

The second line of code above produces the posterior samples for σ , using the samples from the Gaussian posterior of τ . While we assumed that τ is Gaussian, this does not mean that we also assume that σ is Gaussian. You can see for yourself by plotting the density of the samples in `post2$sigma`. Compare that density to `post1$sigma`, the naive posterior produced by estimating σ directly. The new density has the shape we expect from the theory, and you can use the brute force grid computation approach you met earlier in the chapter to confirm that the skewed density for σ that we estimated here gets much much closer to what you'd get from directly computing the naive posterior.

Of course, in most interesting problems, there are too many parameters to directly estimate the naive posterior, and so we have to fall back on an approach like maximum likelihood or (much later in the book) MCMC. But in these simple examples, you can confirm that this trick provides better estimates of the posterior, as theory says it should. Once you learn MCMC estimation, again you can confirm that this τ -to- σ trick provides an estimate of the right density. And since there is no cost in computation, aside from a final step of inverting the samples for τ , there's no reason not to use it, if you care at all about the shape of the posterior density of the variance.

Function `lm.to.mle2`:

```
m.lm <- lm( dist ~ speed , data=d )
m.mle2 <- lm.to.mle2( m.lm , data=d , tau=TRUE )
```

R code
4.108

5 Comparing Models

This chapter introduces two major approaches to comparing multiple *model families*, models which differ in their numbers of parameters and their relations to data, but nevertheless attempt to predict the same outcomes. First, the common model comparison criteria BIC and AIC are developed and related to one another as approximations of posterior probability of entire model families. As approximations of posterior probability, these measures can be used to quantify uncertainty over model families, similarly to how posterior probability was used in previous chapters to average over uncertainty in parameters. Second, but equally important, is to compare estimates across models that include and omit different covariates, as a way of checking how robust inference is to variation in exact model structure.

5.1. Firing the Weatherperson

Suppose in a certain city, a weatherperson issues predictions for rain or shine on each day of the year. Sometimes the weatherperson is right, but sometimes he or she is wrong. The proportion of days with each prediction and outcome are given in the table below.

		Actual	
		Rained	Sunny
Predicted rain	1/4	1/2	
	0	1/4	

The weatherperson was right 50% of the time, and $2/3$ of the time that he or she predicted rain, it was sunny. Suppose that a new weatherperson shows up and says he can do a better job than this, just by always predicting sunny weather. This is what the newcomer's predictions look like:

		Actual	
		Rained	Sunny
Predicted rain	0	0	
	1/4	3/4	

and so he should get the current weatherperson's job. Should he? It's true that the current weatherperson was correct only 50% of the days in the year (the sum of the diagonal). It's also true that 75% of the days in the year were sunny, so if one always predicts sun, one will be correct 75% of the time. But who is the better weatherperson?

It turns out to be quite hard to answer this question. If we simply count the frequency of correct predictions, then the newcomer wins with $75\% > 50\%$. The first weatherperson actually predicts rain 75% when it actually rains only 25% of the time. But suppose we wished to use the predictions to avoid getting rained on. If we go to work with an umbrella on only those days predicted to rain, the first weatherperson never gets us caught unprepared, even though he or she is batting only $1/3$ for predicting rain correctly. The second weatherperson, however, gets us caught in the rain every time. Now, who should get the job, the one who is more accurate overall or the one who is more accurate when it matters?⁵⁴

Choosing to fire the weatherperson or not must depend upon considerations other than just the overall rate of correct predictions. And as it is with weatherpersons, so it is with statistical models. So far, we have leaned on the posterior probability distribution to compare models. That is the inferential step, to estimate posterior probability. But when it comes to choosing a model, a precise definition of purpose is needed. For example, suppose we are attempting to predict the wind speed of a hurricane. The trouble with hurricane winds is that the amount of damage they impose rises sharply as the speed increases. As a result, even a small posterior probability of catastrophic winds might be sufficient for us to decide to order an evacuation. Consideration of costs and benefits arises as a natural part of statistical inference, in cases like this. In contrast, sometimes we just want to find a model that minimizes the expected error, in which case the maximum likelihood estimate is often (though not always) a good bet. And as you'll see in this chapter, there are other ways to measure predictive accuracy that lead to yet other ways to judge the relative values of models, even when we ignore costs and benefits.

This chapter is about how to compare models, when those models are *model families* that differ in ways other than just the value of a parameter. Different model families have different numbers of parameters, and those parameters can have different meanings. The posterior probability distribution across model families can be computed, however, and used as a measure of uncertainty, just like in previous chapters. But some new and thrilling wrinkles appear, once Bayes' theorem references families of models instead of just different parameter values within the same model family. The main obstacle is that models with more parameters—often called more *complex*—always fit better than models with fewer parameters. Despite their better fit, more complex models routinely perform worse than simpler models do in predicting new data. Therefore some way of penalizing model families for complexity is needed.

There are two principled and common ways to introduce such a penalty. First, computing the posterior probability of a model family automatically penalizes complex families, because such families must average over all the uncertainty in each parameter. This penalty leads to the common Schwartz criterion, or BIC, which is an approximation of the posterior probability of the model family. Second, even after averaging over the uncertainty in parameters, we have to select prior probabilities for each model family, in order to finish computing the posterior. BIC typically uses a uniform prior, judging each model family to be equally likely, *a priori*. Another approach is to use a prior probability distribution derived from *information theory* and a concern with overfitting. Such a prior is designed to help us find the model family that minimizes information loss inherent in using a model to approximate reality. This approach leads to the Akaike Information Criterion, or AIC, which averages over parameter uncertainty just like BIC does, but also uses a savvy prior that is a function of both sample size and model family complexity. The neat epistemological truth about AIC is that it is designed to and capable of favoring model families that are not true but nevertheless make better predictions. If you are like most scientists unfamiliar with this literature, that will strike you at first as being impossible. Shouldn't the true model family always predict best? No. Once you finish this chapter, this claim will seem much more natural.

We'll also see that there are useful ways to compare models that do not explicitly reference posterior probability at all. By fitting multiple, structurally-different models, we can ask whether inference about a particular predictor variable is *robust* to the details of the model. This informal version of model comparison is quite common in some fields, such as economics, in which formal model comparison techniques are still rare.

This approach, however, is not in conflict with comparison premised on posterior probabilities. Instead, it is a complementary approach that is much simpler and usually more intuitive than all the fuss we'll make about estimating posterior probabilities.

Here's the plan for the chapter. First, I'll get specific about what a *model family* is. Then I'll explore the problems that arise when we use more complex model families to explain data. Nevertheless, nature is often complex, so we need some ways of finding good complex models. Then we'll look at how Bayes' theorem can be applied to entire model families. The posterior probabilities of models lead us to the two major classes of criteria for comparing models. The first is Bayes Factors and approximations of them using BIC. The second is AIC, the Akaike Information Criterion, and similar criteria like DIC. I relate both of these classes of criteria to Bayes' theorem and show how they can be understood as approximations that aim at different goals. Explaining how BIC and AIC differ will also require a very brief discussion of *information theory*, in order to define AIC's goal properly. In the final sections of the chapter, I demonstrate the equally useful but more informal approach of building tables of model estimates to explore whether or not particular parameter estimates are *robust* to variations in the other predictor variables that are included in a model.

In later chapters, approaches based both on posterior probabilities (BIC, AIC) and on the informal table of estimates will be applied to both linear and non-linear regressions. When we arrive at hierarchical models in Chapter 10, we'll briefly return to the issue of computing posterior probabilities of model families, since hierarchical models introduce some new conceptual wrinkles.

5.2. Model Families

Models can differ from one another in many ways. So far in this book, we've stuck to explicitly comparing models that differ only in the values of their parameters. These models were all of the same exact structure—they had the same number of parameters in the same relations to the data. But they made different predictions, because their parameters took on different values. Think of all of the models that differ only in the precise values of their parameters as being members of the same *model family*.

But models can also differ in other ways. They can have different stochastic nodes, such as Gaussian or binomial. Different kinds of stochastic distributions have different fundamental parameters that determine their shapes. In the case of Gaussian models, these fundamental

parameters are μ and σ (or τ), but other kinds of nodes have other parameters. Later in the book, you'll build models that contain more than one stochastic node, and then how these nodes connect to one another creates important differences among models. Models can also have different functions that relate each fundamental parameter—like μ —to one or more explanatory (or predictor) variables. So far we've only explored linear functions in this role, but in principle your imagination is the only limit (although in practice, you might not be able to effectively estimate what you imagine). Models like these that differ in their stochastic nodes or the functions that relate them to data, or even how much data they reference, are members of different *model families*.

5.2.1. An example. We could write down two linear regressions of braking distance (previous chapter) on speed:

$$\begin{aligned} M_1 : y_i &\sim \text{Normal}(\mu_i, 15.07), \\ \mu_i &= -17.6 + 3.9x_i. \\ M_2 : y_i &\sim \text{Normal}(\mu_i, 13.8), \\ \mu_i &= -5.6 + 3.24x_i. \end{aligned}$$

The first model, M_1 , is the maximum likelihood estimate. The second model, M_2 , is sampled from the naive posterior density. Both of these come from the same model family, because they have all of the same parameters in the same relation to one another and to the data. All that differs are the precise values of these parameters. Models M_1 and M_2 are members of the same model family, \mathcal{M}_1 :

$$\begin{aligned} \mathcal{M}_1 : y_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta x_i. \end{aligned}$$

I will adopt the convention of referring to entire model families with “ \mathcal{M} .” Call this “fancy M .” I’ll use subscripts to distinguish different families. I’ll refer to members of the same model family with a regular roman “ M ,” using subscripts to refer to specific members of the family.

For an example of another model family, recall from the previous chapter, you learned how to fit a quadratic regression model to the braking distance data. The quadratic model comprises a different model family than the simple regression model family \mathcal{M}_1 , because it contains an additional parameter, even though in essence it uses all of the same data. Label the quadratic family \mathcal{M}_2 :

$$\begin{aligned} \mathcal{M}_2 : y_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta x_i + \beta_2 x_i^2. \end{aligned}$$

5.2.2. Nested model families. The families \mathcal{M}_1 and \mathcal{M}_2 are related to one another in a special way: \mathcal{M}_1 is *nested* within \mathcal{M}_2 . What this means is that we can choose specific fixed values for one or more of the parameters in \mathcal{M}_2 and arrive at the family \mathcal{M}_1 . In this case, all we have to do is fix $\beta_2 = 0$ always, and the families become equivalent.

Readers accustomed to likelihood ratio tests may mistakenly think that nesting is a requirement for comparing model families. Model families do not need to be nested in this way in order to be compared. However, some of the inferential methods used in comparing models, like producing predictions that average over model uncertainty, are made easier when the model families are nested. You'll see exactly why when we meet truly non-linear (non-additive) models in later chapters.

Recognizing the existence of nested models like this leads naturally to the question, why not just ignore simpler nested families like \mathcal{M}_1 and always fit the complex family that includes it as a special case? Why isn't the posterior for the parent family \mathcal{M}_2 the same as the posterior over both families? This is a very good question, and it turns out that finding a good answer for it will require some more conceptual work. The basic obstacle is that parameters must be estimated—or *trained*—on data. And that's the problem we turn to now.

5.3. The Problem with Parameters

In the previous chapter, we saw how adding variables and parameters to a model can help to reveal hidden effects and improve estimates. You also saw that adding variables can hurt, in particular when predictor variables are highly correlated with one another. But what about when the predictor variables are not highly correlated? Would it be safe to just add them all?

The answer is “no.” There are two principle concerns with just adding parameters and variables. The first is that adding parameters—making the model more complex—always improves the fit of a model family to the data. By “fit” I mean the $-\log\text{-likelihood}$ or some measure, like R^2 , that is ultimately derived from it. Fit always improves, even when the variables you add to a model are just random numbers, with no relation to the outcome. So it's no good to choose among models using fit to the data. Second, while more complex model families always fit the data better, they often predict new data worse. Models that have too many parameters tend to have higher *variance* than do simpler models. This means that complex model mis-predict new data by a larger distance. But simple models, with too few parameters to adequately capture the

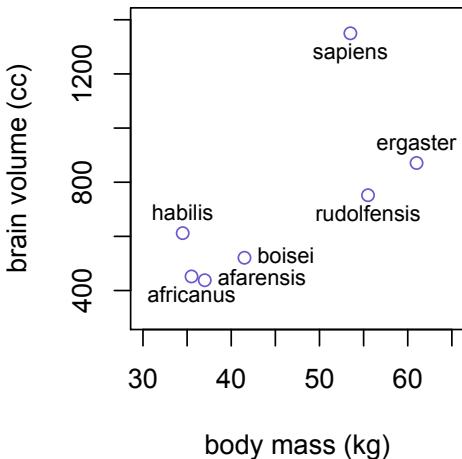


FIGURE 5.1. Average brain volume in cubic centimeters against body mass in kilograms, for six hominin species. What model best describes the relationship between brain size and body size? (Data from McHenry and Coffing 2000.)

data-generating process, tend instead to be *biased*, systematically over-predicting or under-predicting the data. So we can't always favor either simple models nor complex models. Optimal prediction of new data requires navigating between the rocks of bias and variance. Let's examine both of these issues in the context of a simple data example.

5.3.1. More parameters always improve fit. The data displayed in Figure 5.1 are average brain volumes and body masses for seven hominin species. Let's get these data into R, so you can work with them.

```
species <- c( "afarensis", "africanus", "habilis", "boisei",
           "rudolfensis", "ergaster", "sapiens")
brainvolcc <- c( 438 , 452 , 612, 521, 752, 871, 1350 )
masskg <- c( 37.0 , 35.5 , 34.5 , 41.5 , 55.5 , 61.0 , 53.5 )
d <- data.frame(species=species,brain=brainvolcc,mass=masskg)
```

R code
5.1

Now you have a data frame, d, containing the brain size and body size values. It's not unusual for data like this to be highly correlated—brain size is correlated with body size, across species. A standing question, however, is to what extent particular species have brains that are larger than we'd expect, after taking body size into account. A very common solution is to fit a linear regression that models brain size as a linear function of body size. Then the remaining variation in brain size can be

modeled as a function of other variables, like ecology or diet or species age.

But why use a linear function to relate body size to brain size? It's not clear why nature demands that the relationship be a straight line. Why not consider a curved model, like a parabola? Indeed, why not a cubic function of body size, or even a quintic model? I agree that there's no reason yet given to suppose *a priori* that brain size scales only linearly with body size. Indeed, many readers will prefer to model a linear relationship between log brain volume and log body mass (an exponential relationship). But that's not the direction I'm headed with this example. The lesson here will arise, no matter how we transform the data. Even after a log transform of both variables, there's no reason to insist that linear is the only imaginable relationship.

Let's fit a series of increasingly complex model families and see which function fits the data best. We're going to use `lm` to fit these models, instead of using `mle2`. Take a look back at Section 4.8, if you've forgotten or missed how these two methods of fitting linear models relate to one another. In this case, using `lm` will get us expediently to the lesson of this section. Since we're not going to be interested in inferences that depend strongly upon the standard deviation of the outcomes, σ , using `lm` here is fine. As always, you can use `lm` to get the maximum likelihood estimates for the main effects and then use `mle2` to get more information about the posterior of σ . The LIBRARY of code that accompanies this book provides the command `lm.to.mle2` for this purpose (see Section 4.8.4, page 227).

The simplest model that relates brain size to body size is the linear one, which is a *first-degree polynomial*. It will be the first model family we consider:

$$\begin{aligned}\mathcal{M}_1 : v_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta_1 m_i.\end{aligned}$$

This is just to say that the average brain volume v_i of species i is a linear function of its body mass m_i . Now fit this model family to the data using `lm`:

R code
5.2

```
m1 <- lm( brain ~ mass , data=d )
```

Instead of pausing to plot the fit model, like we did in previous chapters, let's focus on the question of how well this model fits the data. The fundamental measure of fit used by maximum likelihood is just the

likelihood. You can extract the log-likelihood of the fit model with the standard `logLik` command:

```
logLik( m1 )
```

R code
5.3

```
'log Lik.' -47.46249 (df=3)
```

Log-likelihood gets smaller (more negative) as the fit gets worse. As usual, nothing can be said about how good this fit is, from just the isolated log-likelihood. We have to compare likelihoods, instead. We'll do that in a moment, once you fit another model. The (`df=3`) at the end of the output is the parameter count. There are three parameters that must be estimated from data: α , β_1 and σ . Note that σ has been counted here, even though `lm` tries hard to hide σ in its summary output.

An alternative measure of fit that works for linear models is R^2 , the proportion of total variation in the outcome variable that has been “explained” by the model. I put “explained” in scare quotes, because explanation implies understanding, and that may not be the case with such models. What is really meant here is that the linear model predicts some proportion of the total variation in the data it was fit to. The remaining variation is just the variation of the residuals (see section 4.5.1, page 198). You can easily compute R^2 yourself with:

```
1 - var(resid(m1))/var(d$brain)
```

R code
5.4

```
[1] 0.490158
```

Take a look at the bottom of the output from `summary(m1)` and you'll find the same number, labeled there “multiple R-squared.” Again, it's hard to say much about this number, unless we can compare it to the R^2 from some other model.⁵⁵

Let's get some other models to compare to the fit of `m1`. We'll consider five more model families, each more complex than the last. Each of these model families will just be a polynomial of higher degree. For example, a second-degree polynomial that relates body size to brain size is a parabola. In math form, it is:

$$\begin{aligned}\mathcal{M}_2 : v_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta_1 m_i + \beta_2 m_i^2.\end{aligned}$$

This model family adds one more parameter, β_2 , but uses all of the same data as \mathcal{M}_1 . Fit this model to the data using `lm` also:

R code
5.5

```
m2 <- lm( brain ~ mass + I(mass^2) , data=d )
```

That weird `I()` surrounding the squared mass just tells R to treat whatever is inside “as is.” If you don’t specify the squared mass in this way, then `lm` actually ends up dropping the term entirely, for rather boring reasons that have nothing to do with science and everything to do with programming. Just be careful to always use the “as is” wrapper when you use `lm` and similar commands with transformed predictor variables, and you’ll be fine. Or you could do as I advised in Chapter 4 and just make a new column in the data frame that contains the squared values. I used the `I()` approach here, so you’ll know what it is, when you inevitably come across it in other books or in another person’s code.

Now let’s fit the rest of the model families. The model families \mathcal{M}_3 , \mathcal{M}_4 , \mathcal{M}_5 , and \mathcal{M}_6 are just third-degree, fourth-degree, fifth-degree, and sixth-degree polynomials built and fit in the same way. Here are the model families:

$$\mathcal{M}_3 : v_i \sim \text{Normal}(\mu_i, \sigma),$$

$$\mu_i = \alpha + \beta_1 m_i + \beta_2 m_i^2 + \beta_3 m_i^3.$$

$$\mathcal{M}_4 : v_i \sim \text{Normal}(\mu_i, \sigma),$$

$$\mu_i = \alpha + \beta_1 m_i + \beta_2 m_i^2 + \beta_3 m_i^3 + \beta_4 m_i^4.$$

$$\mathcal{M}_5 : v_i \sim \text{Normal}(\mu_i, \sigma),$$

$$\mu_i = \alpha + \beta_1 m_i + \beta_2 m_i^2 + \beta_3 m_i^3 + \beta_4 m_i^4 + \beta_5 m_i^5.$$

$$\mathcal{M}_6 : v_i \sim \text{Normal}(\mu_i, \sigma),$$

$$\mu_i = \alpha + \beta_1 m_i + \beta_2 m_i^2 + \beta_3 m_i^3 + \beta_4 m_i^4 + \beta_5 m_i^5 + \beta_6 m_i^6.$$

And here is the code to fit all of them to the data:

R code
5.6

```
m3 <- lm( brain ~ mass + I(mass^2) + I(mass^3) , data=d )
m4 <- lm( brain ~ mass + I(mass^2) + I(mass^3) + I(mass^4) ,
           data=d )
m5 <- lm( brain ~ mass + I(mass^2) + I(mass^3) + I(mass^4)
           + I(mass^5) , data=d )
m6 <- lm( brain ~ mass + I(mass^2) + I(mass^3) + I(mass^4)
           + I(mass^5) + I(mass^6) , data=d )
```

In Figure 5.2, I display the maximum likelihood estimates of the mean brain size as a function of body size, for the first four model families. Each plot overlays the predicted mean, by extracting the coefficients and

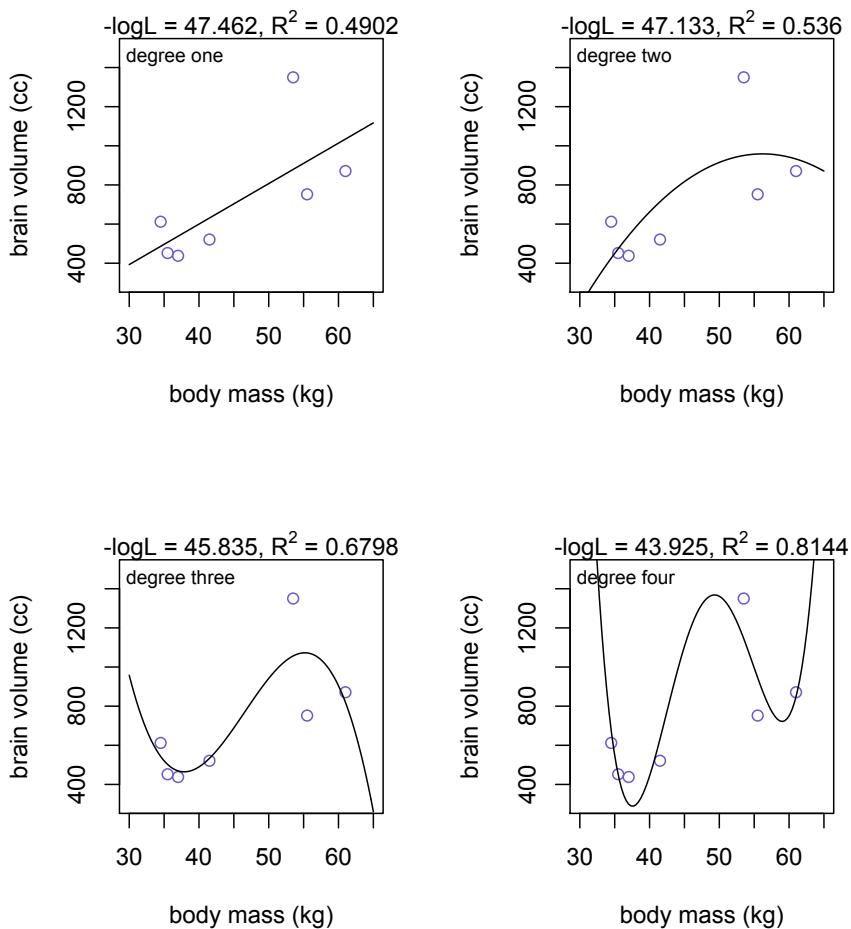


FIGURE 5.2. Polynomial linear models of increasing degree, fit to the hominin brain size and body size data. Each plot shows the predicted mean for a regression model of different complexity. The fit of the model, measured by $-\log$ -likelihood, is displayed above each plot. Smaller values indicate better fits. Upper-left: A simple linear regression, a polynomial of degree one. Upper-right: A parabolic regression, a polynomial of degree two. Lower-left: A cubic regression, a third-degree polynomial. Lower-right: A fourth-degree polynomial regression.

plugging them into the formula for μ_i . For example, to overlay the cubic model:

R code
5.7

```
k <- coef(m3)
curve( k[1] + k[2]*x + k[3]*x^2 + k[4]*x^3 , add=TRUE )
```

Each plot also displays the $-\log\text{-likelihood}$ and R^2 for the fit model. As the degree of the polynomial defining the mean increases, the fit always improves. The fourth-degree polynomial has an R^2 value 30% higher than the linear model! The more complex models fit better, because they can bend and swing in order to get the path of the mean closer to each actual data point.

It's easier to compare these fits with a table.

Model	k	$-\log L$	R^2
\mathcal{M}_1	3	47.462	0.4902
\mathcal{M}_2	4	47.133	0.5360
\mathcal{M}_3	5	45.835	0.6798
\mathcal{M}_4	6	43.925	0.8144
\mathcal{M}_5	7	34.082	0.9889
\mathcal{M}_6	8	$-\infty$	1

The column k is just the parameter count for each model family. Moving down the $-\log\text{-likelihood}$ column, more complex models always have smaller values, indicating better fit. Moving down the R^2 column, more complex models always have higher values, indicating more variance explained. In fact, the most complex model family has 8 parameters, and it fits the data perfectly, attaining an $R^2 = 1$.

However, you can see from looking at the paths of the predicted means that the higher degree polynomials are increasingly absurd. Figure 5.3 shows the most complex model family, \mathcal{M}_6 , with the same kind of plot as in Figure 5.2. This model fits so well in fact that there is no variation left, because the predicted mean passes exactly through every data point. This leads to the $-\log\text{-likelihood}$ vanishing towards $-\infty$, as the standard deviation $\hat{\sigma}$ vanishes to zero and R^2 reaches 1.

The fit is perfect, but the model is clearly ridiculous. Notice that there is a gap in the body mass data, because there are no fossil hominins with body mass between 55kg and about 60kg. In this region, the predicted mean brain size from the higher-degree polynomial models has nothing to predict, and so the models pay no price for swinging around wildly in this interval. In Figure 5.3 especially, the swing is so extreme that I had to extend the range of the vertical axis to display the depth at

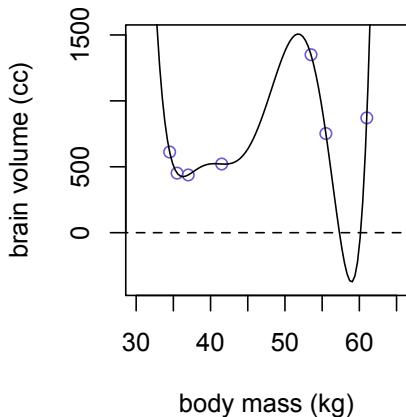


FIGURE 5.3. A sixth-order polynomial linear model of brain size as predicted by body size. The predicted mean now passes exactly through each data point, resulting in a perfect fit. However, the predicted mean also swings wildly above and below the observed data, predicting absurd brain volumes at unobserved body masses.

which the predicted mean finally turns back around. At around 58kg, the model predicts a negative brain size! Clearly, this is absurd. But it is what the model predicts. It pays no price in fitting for this absurd prediction, because there are no cases in the data with body mass near 58kg.

Why does the sixth-degree polynomial fit perfectly? Because it has enough parameters to assign one to each point of data. \mathcal{M}_6 's equation for the mean has 7 parameters:

$$\mu_i = \alpha + \beta_1 m_i + \beta_2 m_i^2 + \beta_3 m_i^3 + \beta_4 m_i^4 + \beta_5 m_i^5 + \beta_6 m_i^6,$$

and there are 7 species to predict brain sizes for. So effectively, this model assigns one parameter to just reiterate each observed brain size. This is a general phenomenon: If you adopt a model family with enough parameters, you can fit the data exactly. But such a model will make rather absurd predictions for yet-to-be-observed cases.

5.3.2. Caught between bias and variance. One tradition in statistical inference is to discuss this phenomenon—the guaranteed better fit of more complex models, but the risk that they will predict worse for new data—in the broader context of a tradeoff between *bias* and *variance*. I hope only to provide adequate intuitions about this tradeoff here. Readers interested in the mathematical details should follow the endnote.⁵⁶

5.3.2.1. The many-headed monster of variance. What you've seen above is the phenomenon of *variance*, or *over-fitting*. To understand the problem of variance, it helps to conceptualize the choice of model family as a

learning problem. We have some experience, which is the current data. We wish to use this experience to make inferences that will help us anticipate future data. When there is only one model family, this reduces to a problem of Bayesian updating of parameter values.⁵⁷ But what about when there is more than one model family?

The basic problem is that parameters must be estimated, or *trained*. Complex models fit very well the data they are estimated on, the *training* data. With enough parameters, they can pass through every case exactly. However, once generalized to new data, such models usually predict very poorly. The reason is that complex models picture the world as being too similar to the training data. This makes their predictions very sensitive to the particular training data set used. This sensitivity in turn hurts prediction.

This is the problem of *over-fitting* and *variance*. The *variance* is variation in predictions across different training data. A complex model will be very sensitive to the exact composition of the training data, curving around in order to get the mean close to every case. Once confronted with new data to predict, such a model will tend to predict absurd cases, like the imaginary hominin species with a body mass of 58kg and brain volume of -229cc (Figure 5.3). The model will be inaccurate in prediction, precisely because it is so sensitive to the training data. Removing any one species from the training data in the hominin brains example would lead to large changes in the parameter estimates and therefore in the predictions.

Another way to talk about the variance half of the dilemma is to call it *over-fitting*. What is being over-fit is the error variance of the model. Imagine for example that the true relationship between body mass and brain size is approximately linear, but with a generous standard deviation σ around the mean. As you add parameters to the model, these parameters start standing in for the random error variance around the mean. They are explaining what are in essence random numbers. Of course, you don't have to believe that the variance around the mean is truly random. Instead, it's more likely that causal processes we have no information about—like the details of measurement error or the amazing machinery of genetic recombination—are generating it. Regardless, the point stands that, once the complex model tries to generalize to new cases, it will have *over-fit* the training data and produce poor predictions.

5.3.2.2. The whirlpool of bias. The other horn of this dilemma is *bias*. While complex model families are prone to over-fit, simple model families are likewise prone to *under-fit* and demonstrate systematic errors in

prediction of new data. Systematic error in prediction that is insensitive to the particular training data used is called *bias*.

Consider for example the conceptually simple case of the true data-generating process being a quadratic model, a parabola. When you fit a linear model to such data, the error variance σ must grow to substitute for the inability of the linear mean to bend with the data. Larger error variance σ means lower average predictive accuracy on average, regardless of the training data used. In other words, models that are too simple relative to the real processes generating the data try to make the best of a bad job by making wide vague predictions. This means they accept larger error in prediction overall. Such models are *biased*.

Unfortunately, perfect prediction is not a realistic goal. It isn't possible to reduce both bias and variance to zero. But it is possible to search for model families that minimize the sum of bias and variance. This is the heart of the bias-variance tradeoff: We hope to find model families that are complex enough to be largely unbiased but also simple enough to avoid too much sensitivity to the training data. Minimizing either bias or variance is hardly ever the answer. Usually, by accepting some bias, we can find a model that has lower overall prediction error. Likewise, by accepting more variance, we can lower bias. Depending upon the model type (the nature of the stochastic nodes and relations of data to parameters) and how the model is estimated, the location of the optimal tradeoff can move substantially.⁵⁸

5.3.2.3. An example: The Zipper. Let's consider these concepts in a particularly contrived but illuminating example. Consider the following bivariate regression problem:

```
d <- data.frame( y=c(rep(c(1,-1),4),1) , x=1:9 )
```

R code
5.8

I've plotted these data in Figure 5.4. The pairs of y and x values form a zig-zag zipper-like pattern. That's why I call this example "the zipper." While I would hope that no one would just start fitting polynomial regressions to data that looks like this, this example will work well to illustrate the different natures of bias and variance, as well as why bias afflicts simpler models and variance afflicts complex ones. The horizontal line and the swinging curve in the figure are, respectively, a degree one regression (M_1) and a degree six regression (M_6). The equations for the means are:

$$M_1 : \mu_i = \alpha + \beta_1 x_i.$$

$$M_6 : \mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \beta_5 x_i^5 + \beta_6 x_i^6.$$

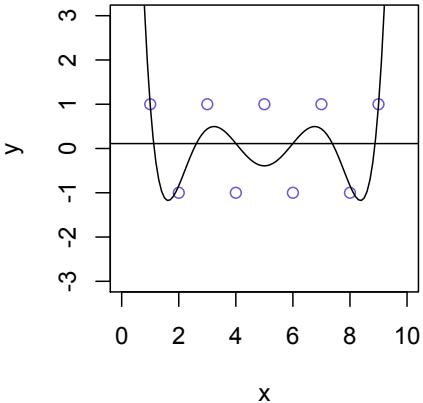


FIGURE 5.4. The Zipper, an illustration of bias and variance. The nine pairs of y and x values form a zig-zagging zipper-like shape. The straight line is a degree one linear regression and the curve is a degree six polynomial regression, both fit to all nine cases.

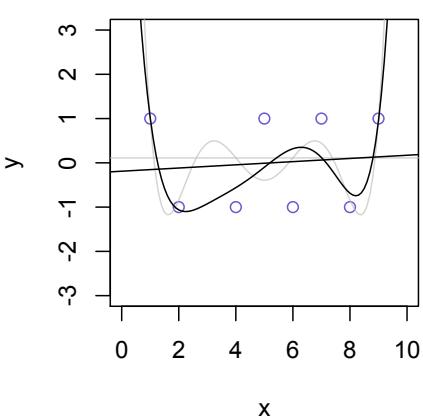


FIGURE 5.5. The Zipper, now re-fitting both model families after removing the third row of data. The linear model, \mathcal{M}_1 , changes only slightly. It still lies about the same distance from all of the points. In contrast, the complex model, \mathcal{M}_6 , has changed path a lot more. Repeating this exercise for each row of the data generates the curves shown in Figure 5.6.

Now let's consider what happens to each of these model families as we remove each pair of y and x values, one at a time. Suppose we remove row 3 from the original data, and then re-fit both model families to the 8 cases that remain. Figure 5.5 plots this exercise, with the gray line and curve showing the original fits to all 9 cases and the black line and curve showing the new maximum likelihood curves. Because the complex model family is so much more flexible, its new fit is quite different from the original fit. The simple linear model family, however, changes very little.

We can repeat this exercise for every row in the data, removing each row one at a time, refitting both models and saving the results. Here's a short piece of code to simplify this procedure:

```
m1.list <- vector("list",9)
m6.list <- vector("list",9)
for ( i in 1:9 ) {
  dtemp <- d[-i,]
  m1.list[[i]] <- lm( y ~ x , data=dtemp )
  m6.list[[i]] <- lm( y ~ x + I(x^2) + I(x^3) + I(x^4)
    + I(x^5) + I(x^6) , data=dtemp )
}
```

R code
5.9

A notable programming trick here is the `d[-i,]` index notation. The minus symbol “-” in front of the index i tells R to remove that row from the matrix d . This lets us easily make a new data frame with only the row i missing. The code then just fits both models to the new data and then repeats this process for each row of the original data. So in each pass, both models are fit to 8 rows from the original data, with a different row missing from the original 9. You can think of each pass being a different training data set.

After the code finishes, the symbols `m1.list` and `m6.list` each contain nine fit models. You can extract an individual model by using the list element double brackets: `m1.list[[3]]` gives you the 3rd model in the list, for example. Figure 5.6 displays the predictions of all of the individual lines and curves in these lists. The lefthand plot shows the 9 different degree one regressions. The righthand plot shows the 9 different degree six regressions. The simple model family \mathcal{M}_1 varies much less across training sets. The slope does change from set to set, but the error in prediction is still quite similar across all of the sets. This is what is meant by *bias*—the simpler model family does rather badly on average. But at the same time, it behaves this way because it is not flexible and changes quite little in response to dropping any one case from the training data. And so the error in prediction simultaneously has low variance, across training sets. In contrast, the complex model family \mathcal{M}_6 varies hugely across training sets. Dropping any one row results in massive swings in the path of the predictions. This model family has high *variance*—the complex model family predicts poorly because it is too sensitive to the exact composition of the training data. But at the same time, it has low bias, because the average prediction for each row i is much closer to the actual y_i value. It's the extreme error for the omitted rows in each training set that hurts \mathcal{M}_6 's overall accuracy.

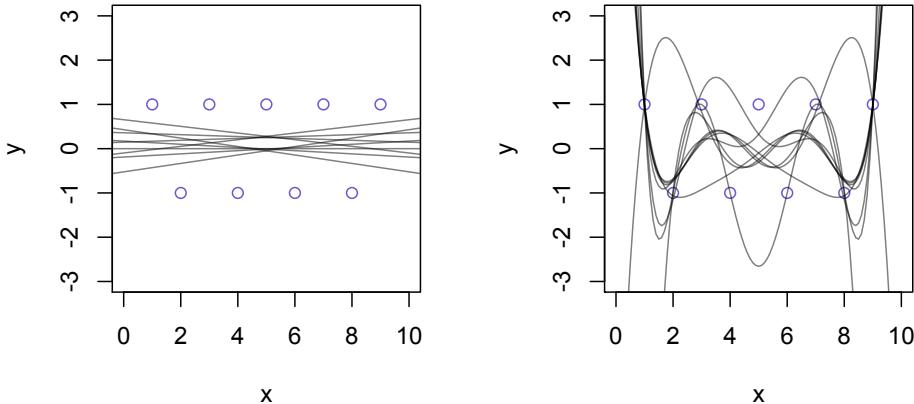


FIGURE 5.6. The Zipper comes to a close. Each line (left) or curve (right) is the maximum likelihood fit to a single training data set, omitting a unique row from the original data. The simple degree one regression (left) varies little across training sets. It systematically mis-predicts the data, but does so to about the same degree across all the training sets. It is biased but has low variance. In contrast, the complex degree six regression (right) changes its path tremendously from one training set to another. This model family has low bias but high variance.

Overly-simple models are biased. They systematically make mistakes in prediction. As a result, they are insensitive to the exact composition of the training data and so have about the same crummy accuracy on new data that they have on training data. Overly-complex models in contrast have high variance. They make massive prediction mistakes whenever new data are unlike training data. But they have low bias, because the average prediction for any one point, across training sets, is quite close to the actual point. In every training set in which the point in question was not present, however, the model does an incredibly bad job at prediction. Both model types—too simple and too complex—end up with overall poor accuracy, high *mean squared error* for example.⁵⁹

5.3.2.4. Cross-validation. The zipper example you've just seen used a technique known as *cross-validation*. Specifically, it was *leave-one-out*

cross-validation. Some readers will have already encountered this approach. It is one family of methods for navigating between bias and variance. Cross-validation evaluates models by sequentially or randomly leaving out cases in the data. So for n cases in the actual data, cross-validation generates a large number of new data sets containing a subset of the original cases, either by leaving some out or by resampling with replacement. Model families are compared based upon their average predictive accuracy on the omitted cases or another resampled set, across all of the training sets. This approach can be laborious, but it does provide a proven way to resolve the bias-variance tradeoff. It does so, because cross-validated models are evaluated on their predictive accuracy and not their fit.

It turns out that this version of cross-validation tries to estimate the same fundamental quantity that AIC does.⁶⁰ But before we can address this connection, we need to focus again on Bayes' theorem.

5.4. Bayes' Theorem and Model Families

Parameters are a problem. The world is a complex place, and so we can often use more variables and more parameters in order to make better predictions. And so in order to escape the bias of simple models, we add parameters. But parameters are not harmless, because they must be estimated and therefore induce variance. A variety of approaches to resolving this dilemma have been developed and are useful in practice. This chapter is working towards explanations of two of the most common metrics for addressing the parameter problem, BIC and AIC. But there are many other approaches, such as cross-validation and stepwise procedures and tally rules and boosting.

But what does this soup of approaches have to do with Bayes' theorem? Shouldn't it be possible to just compute the posterior probability of model families, and thereby let the logic of probability tell us what the evidence says about model complexity? Well, yes. But Bayes' theorem only solves part of the problem. There are some obstacles, as you'll see. Don't get the impression that this area is completely settled. Unlike the use of Bayes' theorem to compare models within a family, comparison of models across families is still an active and relatively new area of research. There are various obstacles, and approximations and assumptions must be used to dodge these obstacles. These approximations reduce the generality of the results, but they do lead to useful metrics, provided we keep their limitations in mind.

In this section, you'll see how to generalize the calculation of posterior probabilities to entire families of models. In doing this, you'll see

that Bayes' theorem does include an automatic penalty for complexity. After this section, you'll be ready to get into BIC and AIC and appreciate how both of these metrics are approximating applications of Bayes' theorem, applied to model families.

5.4.1. The posterior probability of a model family. Defining the posterior probability of a model family is a straightforward application of Bayes' theorem. Let's consider a simple case, in which there are two structurally different models, \mathcal{M}_1 and \mathcal{M}_2 . Then Bayes' theorem tells us that the posterior probability of \mathcal{M}_1 in light of the evidence D is just:

$$\Pr(\mathcal{M}_1|D) = \Pr(\mathcal{M}_1) \frac{\Pr(D|\mathcal{M}_1)}{\Pr(D)}. \quad (5.1)$$

The terms in this expression are the same as when we met Bayes' theorem in Chapter 2. They just reference different objects, now. Instead of \mathcal{M}_1 being a model defined by a unique parameter value, \mathcal{M}_1 is now a family of models that share the same structure. This difference does make a difference in calculation. Here's why.

When the models to compare are from the same model family, computing the posterior above presents no new obstacles. You've implicitly used this formula many times already, whenever you sampled from the naive posterior. Depending upon the problem and the prior you wish to adopt, there's no guarantee that computing the posterior will be easy. But whatever problems we might encounter in computing posterior probabilities for models from the same parametric family, they tend to get worse when we try to compare models that are actually structurally different. The reason is that the posterior probability of the model family above, $\Pr(\mathcal{M}_1|D)$, actually includes inside of it all of the posterior probabilities of all of the models that are a member of it. Posteriors inside posteriors? If you are a normal person, this is starting to sound a little mad.

Actually, having a posterior inside your posterior isn't crazy. It's just a matter of applying Bayes' theorem twice. Suppose the family \mathcal{M}_1 has only one adjustable parameter named θ ("theta"). Then the posterior probability density for any particular value of θ named θ_j is just, from Bayes' theorem as in Chapter 2:

$$\Pr(\theta_j|D, \mathcal{M}_1) = \Pr(\theta_j|\mathcal{M}_1) \frac{\Pr(D|\theta_j, \mathcal{M}_1)}{\Pr(D|\mathcal{M}_1)}. \quad (5.2)$$

All I've done here is make every piece of the formula conditional on the model family, \mathcal{M}_1 . Otherwise, it's just the same Bayes' theorem used to define the posterior probabilities of the possible values of the parameter

θ . The left side is still just the posterior, the first term on the right is still just the prior, and the ratio is still just the likelihood over the normalizing constant. We never worried about conditioning everything on \mathcal{M}_1 before, because there was only one family of models, so there was no need to. But of course all of the posteriors you've estimated already have been conditional on the identity of the model family you assumed. The above just makes this explicit, because now there is more than one family to consider.

Now notice that the term $\Pr(D|\mathcal{M}_1)$ on the bottom of the righthand side of (5.2) also appears in (5.1), on the top. That is, the *likelihood* of the data D conditioned on the entire model family \mathcal{M}_1 is the same thing as the normalizing constant in the expression for the posterior probability of each parameter value within the family. The math tells us this is true, but it also makes sense, once you stop to think about it. The probability $\Pr(D|\mathcal{M}_1)$ —which was written just as $\Pr(D)$ in Chapter 2—is just the average likelihood across all possible parameter values. That is, the normalizing constant manages to normalize the posterior, because it just takes each product $\Pr(\theta_j|\mathcal{M}_1) \Pr(D|\theta_j, \mathcal{M}_1)$ and adds it to all the others. If you want to ensure that a collection of numbers sums to one—as we should in the case of a posterior density—then you can just add them all together and divide each by that sum. This is all the normalizing constant $\Pr(D|\mathcal{M}_1)$ does in (5.2). It's just the likelihood of the data, averaged over values of θ , using the prior. More precisely:

$$\Pr(D|\mathcal{M}_1) = \int \Pr(\theta|\mathcal{M}_1) \Pr(D|\theta, \mathcal{M}_1) d\theta,$$

which is just the mathy way of saying “add up the products of the likelihood and prior, for all continuous values of θ .” (The symbol for integration, \int , looks like a funny “S” because it stands for summation.) So when we see $\Pr(D|\mathcal{M}_1)$, the likelihood of model family \mathcal{M}_1 , show up again in the posterior for the entire model family, (5.1), now we know how to compute it, in principle. It is an *average* likelihood for the model, averaged over the prior probabilities of the individual values of the free parameter θ .

5.4.2. Marginal likelihood. This probability, $\Pr(D|\mathcal{M}_1)$, is sometimes called *marginal likelihood*. What is “marginal” about it is that it computes the metaphorical margin of a table containing the posterior probabilities of all of the individual models in the family \mathcal{M}_1 . Sums and averages are often displayed at the margins of tables, and this is where the terminology comes from.

5.4.3. A penalty for complexity. The marginal likelihood gets rid of the parameter θ by summing across uncertainty in its value. This is also sometimes called *integrating out* the parameter. That doesn't mean that the marginal likelihood throws away the uncertainty in the parameter. Instead, the uncertainty about the value of the parameter has a huge effect on the likelihood. It provides a natural penalty for the model family \mathcal{M}_1 having a free parameter.

In this way, the posterior probability of a model family automatically incorporates one kind of answer to the parameter problem. To understand why, consider the common case of assuming a uniform prior for $\Pr(\theta|\mathcal{M}_1)$ (see page XX, Chapter 2, to remind yourself about uniform priors). The likelihood $\Pr(D|\theta, \mathcal{M}_1)$ is peaked at the maximum likelihood estimate $\theta_j = \hat{\theta}$, whatever it may be. And so the effect of the prior probabilities is to water down this peak, making the likelihood over the entire family $\Pr(D|\mathcal{M}_1)$ lower, often much much lower, than the likelihood at the maximum likelihood estimate $\Pr(D|\hat{\theta}, \mathcal{M}_1)$.

In contrast, a model "family" with no adjustable parameters does not get watered down in this way. If it fits the data well, it pays no penalty for flexibility. Model families with adjustable parameters do. Likewise, the more free parameters a model family has, the more dimensions the marginal likelihood $\Pr(D|\mathcal{M}_1)$ has to average over, making the likelihood of the model family smaller and smaller. In turn, the posterior probability of the model family, $\Pr(\mathcal{M}_1|D)$, tends to decline as we add more free parameters to a model. Now, if the new parameters help enough, the improved fit to the data can make up for having to average over the uncertainty in the parameters. But the fact that the posterior probability of models automatically penalizes model families that are more complex stands in stark contrast to measures like the maximum likelihood or that dinosaur R^2 , both of which always increase with model complexity.

A curious thing about this natural penalty in the marginal likelihood is that it varies with sample size, not just with the number of parameters. This is because the likelihood across values of a parameter becomes increasingly peaked as the sample size increases. If the likelihood function becomes sufficiently peaked, then the marginal likelihood of the family will be dominated by the maximum likelihood and suffer little penalty for flexibility. In a later section of this chapter, we'll see a popular estimate of this Bayesian penalty and therefore how it combines sample size and the number of parameters.

5.4.4. Bayes Factors. A common approach to comparing model families is to consider only the ratios of posterior probabilities of models, a

posterior odds ratio. In such a ratio, the normalizing constant will be the same in both probabilities, and therefore it will divide out. For example, if we want to compare two model families \mathcal{M}_1 and \mathcal{M}_2 , we need to compare their posteriors, $\Pr(\mathcal{M}_2|D)$ and $\Pr(\mathcal{M}_1|D)$. Using the definition of the posterior probability, this implies:

$$\frac{\Pr(\mathcal{M}_1|D)}{\Pr(\mathcal{M}_2|D)} = \frac{\Pr(\mathcal{M}_1) \Pr(D|\mathcal{M}_1) / \Pr(D)}{\Pr(\mathcal{M}_2) \Pr(D|\mathcal{M}_2) / \Pr(D)} = \frac{\Pr(\mathcal{M}_1) \Pr(D|\mathcal{M}_1)}{\Pr(\mathcal{M}_2) \Pr(D|\mathcal{M}_2)}.$$

$\Pr(D)$ just divides out, since it is on both the top and bottom. This leaves only the priors, $\Pr(\mathcal{M}_1)$ and $\Pr(\mathcal{M}_2)$, together with the marginal likelihoods, $\Pr(D|\mathcal{M}_1)$ and $\Pr(D|\mathcal{M}_2)$.

The ratio of marginal likelihoods in such a ratio is known as a *Bayes Factor*.⁶¹ Use of these ratios was suggested by the physicist Harold Jeffreys, a famous opponent of Fisher's *P*-values, as a solution to hypothesis testing in a Bayesian framework. Typically, analysts continue to assume uniform prior probability, such that $\Pr(\mathcal{M}_1) = \Pr(\mathcal{M}_2)$, and then the problem reduces to one of comparing marginal likelihoods. This is just how computing the posterior probability across parameter values reduced to the likelihood function in Chapter 2. However, there's nothing about this procedure that precludes using informative priors.

5.4.5. Trouble in Posteriorland. Applying Bayes' theorem to families of models provides a natural and logical way to compare structurally different models. This method of comparison automatically incorporates a kind of penalty for model complexity, addressing the parameter problem. But there are at least three obstacles to using these posterior probabilities, whether in the form of Bayes Factors or otherwise.

5.4.5.1. Computation. First, computing the marginal likelihoods, like $\Pr(D|\mathcal{M}_1)$, contains some pitfalls. Unless the statistical problem is of a special type, calculating the marginal likelihood of the model family can be a daunting numerical project. As the dimensionality of the model family increases—you add more predictor variables, for example—the problem gets worse. In practice, we can assume multivariate normality and all of the other things we've already been assuming about the posterior and make the job easier. But as the complexity of a model family increases, the amount of evidence needed to make these assumptions reasonable increases. Now, for simple tasks like comparing two normal distributions or even many multiple regressions, Bayes Factors can be computed. But our universe of models is bigger than that, so this isn't going to be a universal solution.

5.4.5.2. *Choice of prior within the family can matter for comparisons across families.* Second, a related problem with computing the marginal likelihood is that the prior you choose for the parameters within the model family can have unanticipated effects on comparisons across families.⁶² Here's the quick explanation. While the details of a flat prior across parameters within a model family will have essentially no effect on inferences about parameters, the details will matter once we start comparing families of models. The reason is that the marginal likelihood is proportional to the prior for parameters. This means that arbitrary differences in magnitude between priors across model families can dominate comparisons of families, even though those differences have no effect on inferences about parameters within families. This is no bueno.

Here's a more mathematical explanation. I will forgive readers for skipping ahead to the next section. Recall that the posterior probability of a family \mathcal{M}_1 is:

$$\Pr(\mathcal{M}_1|D) = \Pr(\mathcal{M}_1) \frac{\Pr(D|\mathcal{M}_1)}{\Pr(D)}.$$

Now whatever priors we choose for model families, $\Pr(\mathcal{M}_1)$ etc., the factor $\Pr(D)$ averages over these to produce a proper posterior. But the priors for the parameters within each model family are still lurking inside here. Let's expand that marginal likelihood again, to reveal them:

$$\Pr(D|\mathcal{M}_1) = \int \Pr(\theta|\mathcal{M}_1) \Pr(D|\theta, \mathcal{M}_1) d\theta.$$

There they are, $\Pr(\theta|\mathcal{M}_1)$. You might think we can just adopt uniform or near-uniform flat priors here, as is quite common. Indeed, it makes little difference for inferences within a family what prior we use, as long as there is enough data to make the likelihood dominate. For example, a common and useful way to specify an uninformative prior is to state the prior as a normal distribution with very large variance σ^2 (very low precision τ). This makes the prior very flat and renders it uninformative, since over the relevant range of the prior, it is essentially a constant: $\Pr(\theta|\mathcal{M}_1) = K_1$. K_1 just represents a constant value here. Another approach is to adopt a uniform prior over a fixed range, in which case we determine the value of K_1 by the width of the range of parameter values it covers. Either way, the likelihoods dominate inference about parameters, as you saw back in Chapter 2. The value of K_1 doesn't matter.

But something troubling happens when we go to compute the marginal likelihood, $\Pr(D|\mathcal{M}_1)$. Now the entire marginal likelihood ends up being proportional to this constant, K_1 . So when we go to compare the family \mathcal{M}_1 to another family \mathcal{M}_2 with constant prior K_2 , the ratio K_1/K_2

will influence the posterior probabilities of the model families. This wouldn't be so troubling, except that the values of these constants can be easily manipulated by specifying different volumes of non-zero ranges for the parameters, or by changing the variance of a (multi-variate) Gaussian prior. We adopted the uninformative (or nearly uninformative) priors within families in order to ignore prior probability. But now prior probability has resurfaced and snarls at us, triumphantly.

This problem is not well-appreciated in the literature, possibly because it is not an issue in all applications. And maybe there is a clever trick about to emerge that will solve this problem for us. As is typical in the sciences, we have tools that are known to work over some finite domain of problems, but we only later come to truly understand why they work and why they also sometimes fail. The solution to this problem with priors, whatever it is, will make inference about model families more powerful, because it will help us define its proper domain of usefulness.

My own hunch is that this problem arises from the mistake of focusing on model truth rather than on model utility. As you'll see later in the chapter when you consider AIC, it can make sense to adopt adjustable prior probabilities for parameters and models that aim to help us reduce estimation error or achieve desired outcomes. The prior is part of the model, to my mind, just like the way we search for the maximum likelihood estimate is part of the model. If it influences inferences, then it is part of the model, and we shouldn't ask what is true but rather what is useful.

But this matter is far from settled, and I don't want the reader to get any other impression. Statistics is just as argumentative and controversial as every other part of science. It serves us well to remember that. You know a cold, dead field when everyone in it agrees.

5.4.5.3. The bias-variance dilemma. Third, the bias-variance tradeoff cannot be escaped. This tradeoff means in practice that the true model family, even if it exists in the set of considered model families, will routinely not be the best model for prediction. So even when the marginal likelihoods are available, they may not fully resolve the dilemma of selecting a model that performs best in prediction. Indeed, it is now widely appreciated that the natural penalty for complexity in the marginal likelihood does not result in an optimal solution to the bias-variance tradeoff.⁶³

5.4.5.4. What to do? Partly because of and partly despite these obstacles, there are two metrics that have become quite common for producing estimates of posterior probabilities of model families. These are the Schwarz criterion, or BIC, and Akaike Information criterion, or AIC. There are other metrics out there, but in nearly all cases they can be classed as members of the BIC family or the AIC family. Members of each family share an objective, but they do not always perform equally well on the same problem. So I expect the sizes of both families to grow over time, as refinements are made for special classes of models or estimation problems.

Still, it makes sense to focus on the contrast between BIC and AIC as an introduction to the model comparison approach. These two are omnipresent in statistical software now, and they are very easy to compute and use. In the next two sections of this chapter, you'll get an introduction to BIC and AIC, focusing on heuristic explanations of how they are derived and therefore the problems they attempt to solve. Afterwards, you'll see how to use either or both metrics (or any related metric) to compare and average model families.

5.5. BIC

Suppose you want to compute the posterior probability density across several model families, using Bayes Factors and uniform prior probabilities of the families. However, suppose you have no idea how to go about averaging over the naive posterior densities. What you need is a way to estimate the marginal likelihood of each family. In 1978, Gideon Schwarz showed that an estimate of the relative marginal $-2 \log\text{-likelihood}$ of a model family \mathcal{M}_i with k_i parameters and fit to n observations is:

$$\text{BIC}_i = -2 \log \Pr(D|\mathcal{M}_i(\hat{\theta})) + k_i \log(n),$$

where $\Pr(D|\mathcal{M}_i(\hat{\theta}))$ is just the likelihood of the maximum likelihood member of the model family. So to compute this measure, we just find the maximum likelihood estimates for the model family, multiply the logarithm of the maximum likelihood by -2 (if you are familiar with deviance, that's all this is), and finally add $\log(n)$ times the number of parameters in the model family. The term $k_i \log(n)$ is the approximate penalty for complexity. In practice, it prevents BIC from doing what raw maximum likelihood would, always choosing the most complex model family.

This criterion has since come to be called the Bayesian Information Criterion, hence the familiar abbreviation BIC. But there is really nothing about information in this criterion. It is called BIC by analogy to

AIC, which we'll turn to next. AIC was derived first, and Schwarz explicitly contrasted BIC to AIC in his 1978 paper.⁶⁴ You'll still sometimes see BIC referred to more appropriately as the Schwarz criterion, but this terminology is fading.

It doesn't so much matter what we call this thing. What matters is that we understand the assumptions that underlay it, in order to understand what useful work it can do for us.

5.5.1. Where BIC comes from. I'm not going to present the mathematical derivation here. Readers interested in the details of how to derive BIC can find various derivations in a number of places.⁶⁵ But more to the point, knowing how to manipulate the mathematics isn't so helpful for most scientists. Nevertheless, knowing the assumptions behind BIC is crucial for understanding when you will be comfortable using it to produce estimates. With that goal in mind, here's the strategy for deriving it.

The goal of the measure BIC is to provide an estimate of the marginal likelihood of a model family \mathcal{M}_i , without having to actually average over all of the k_i parameters in it. Because of this goal, you can think of BIC as a large-sample estimate of the marginal likelihood of a model family. As the sample size increases, the posterior distribution for each of the k_i parameters becomes increasingly Gaussian in shape. This is true because of the central limit theorem—it doesn't matter what the stochastic node is. Once we assume the posterior is multivariate normal—just what I encouraged you to assume in previous chapters, as an easy and usually-conservative approach—then it is possible to write down the marginal likelihood, provided you know the variance-covariance matrix (`vcov`). If the sample is large enough, even the prior probability over parameters ceases to be an obstacle, because as you saw in Section 2.5.1 (page 51), with enough evidence, only the strongest sorts of priors exert much influence on the posterior. This means that BIC doesn't strictly assume uniform priors for the parameters in the model family. Instead, it assumes that there's enough evidence to bury the prior.⁶⁶

Because the Gaussian density is so easy to work with mathematically, nice things happen in the algebra. Assuming that the variance-covariance term gets small enough to be ignored—which is likely when the sample is large—then an estimate of the relative posterior probability of the model family \mathcal{M}_i with k_i parameters is:

$$\Pr(\mathcal{M}_i|D) \propto \Pr(D|\mathcal{M}_i(\hat{\theta})) \frac{1}{\sqrt{n^{k_i}}}.$$

Taking the logarithm of the above approximation gets us almost all the way to BIC:

$$\log \Pr(\mathcal{M}_i|D) \propto \log \Pr(D|\mathcal{M}_i(\hat{\theta})) - \frac{1}{2}k_i \log(n).$$

The first term on the right is just the log-likelihood at the maximum likelihood estimate. The second term is the expected penalty to that likelihood, for model complexity. Finally, multiply everything by -2 to arrive at BIC. The final step with the -2 isn't necessary nor important. It's there mainly for historical reasons, because statisticians have both utilitarian and conventional reasons for scaling log-likelihoods as deviances. Remember, "deviance" usually just means twice the negative log-likelihood. Since so many statistics programs already report deviance for a fit model, if the definition of a measure like BIC uses deviance, it makes calculation easier.

5.5.2. Computing posterior probabilities from BIC. BIC is commonly used in combination with uniform priors for model families, and it can seem like this is also one of its assumptions. BIC doesn't really demand uniform priors over model families. BIC itself is just an estimate of the marginal deviance of the model family. You can use it with another prior probability density. Here's how. In general, for m model families, the BIC-estimated posterior probability of a family \mathcal{M}_i is:

$$\Pr(\mathcal{M}_i|D) = \frac{\exp(-\frac{1}{2}\text{BIC}_i) \Pr(\mathcal{M}_i)}{\sum_{j=1}^m \exp(-\frac{1}{2}\text{BIC}_j) \Pr(\mathcal{M}_j)}.$$

In the typical use of BIC, one assumes uniform priors across all the m model families: $\Pr(\mathcal{M}_i) = \Pr(\mathcal{M}_j) = 1/m$. As a consequence the prior probabilities divide out and have no influence on the posterior probabilities.

But there's nothing stopping you from using another set of priors over model families. Indeed, there are sometimes good reasons to adopt explicitly informative priors across models. This isn't a call for using any old subjective prior that we want to. Instead, some priors are special and useful, because they relate to particular objectives. This is the winding path that leads us to the other major type of model comparison criterion, AIC.

5.6. AIC

A few years before Schwarz derived BIC as an estimate of marginal likelihood, Hirotugu Akaike (said like ah-ka-ee-kay, not like uh-KAI-kee) published the other famous metric for comparing model families.

Akaike called this metric “an information theoretic criterion,” AIC for short. It has a very simple formula with the same structure as BIC. The AIC of a model family \mathcal{M}_i with k_i parameters is:

$$\text{AIC}_i = -2 \log \Pr(D|\mathcal{M}_i(\hat{\theta})) + 2k_i.$$

This is the classic AIC.⁶⁷ A very common refinement to AIC that is almost always better in practice is AICc⁶⁸:

$$\text{AICc}_i = -2 \log \Pr(D|\mathcal{M}_i(\hat{\theta})) + \frac{2k_i}{1 - (k_i + 1)/n}.$$

Just like BIC, both AIC and AICc are the deviance of the maximum likelihood member of the family plus a penalty that is proportional to the number of parameters.

Why is the penalty in AIC different than the $k_i \log(n)$ of BIC? Akaike used an argument based upon *information theory* instead of an attempt to estimate a Bayes Factor, and so the origins of AIC are rather distinct from those of BIC. Nevertheless, everything in inference about models can be related to Bayes’ theorem ultimately, and so there is a deep connection between BIC and AIC. Despite this deep and real connection, the two metrics aim at different goals and end up with different penalties for complexity.

Since AIC is based upon information theory, we’ll have to spend a little time in this section introducing the relevant formal definition of information. If you are like most natural and social scientists, information theory is something that you’ve heard of. It has been buzzing at your ear maybe for years. You might be vaguely aware that it has something to do with encryption and entropy. But it is also true that information theory is useful in many contexts, far outside of statistical inference and the study of communication as they are usually defined. Indeed, there is an entire branch of Bayesian statistics—one not covered in this book—that uses information theory to construct objective prior probabilities.⁶⁹ The growing presence of so-called *maximum entropy* or “Maxent” models is fundamentally an offshoot of this branch. So having some conceptual understanding of it will yield unanticipated and possibly large dividends in your future.

As is typical of this book, I go light on the mathematics. The point is to provide enough intuition so the practicing scientist can appreciate why AIC has the properties that it does and that BIC does not.

5.6.1. Information and uncertainty. Think back to the weather forecasters at the start of this chapter. Whichever weatherperson we decide to hire, as long as there is any correlation between the predictions and

the actual weather, then everyone would agree that the weather forecasts reduce our *uncertainty* about the weather. The question is which set of forecasts reduce uncertainty more. More starkly, when the day actually arrives and we can experience the actual weather ourselves, then there is no longer any uncertainty, but we still have the problem of characterizing how uncertain we were to begin with. That is, how much has learning the weather reduced our uncertainty? The measured decrease in uncertainty will be our definition of *information*.

Information: The reduction in uncertainty derived from learning an outcome.

To use this definition, what we need is a principled way to quantify the uncertainty inherent in a probability distribution.

5.6.1.1. *Quantifying uncertainty.* Suppose for example that there are two possible weather events on any particular day: Either it is sunny or it is rainy. Each of these events occurs with some probability, and these probabilities add up to one. What we want is a function that uses the probabilities of shine and rain and produces a measure of uncertainty. What properties should such a measure of uncertainty possess? There are three intuitive desiderata.

- (1) The measure of uncertainty should be continuous. If it were not, then an arbitrarily small change in any of the probabilities, for example the probability of rain, would result in a massive change in uncertainty.
- (2) The measure of uncertainty should increase as the number of possible events increases. For example, suppose there are two cities that need weather forecasts. In the first city, it rains on half of the days in the year and is sunny on the others. In the second, it rains, shines and hails, each on 1 out of every 3 days in the year. We'd like our measure of uncertainty to be larger in the second city, where there is one more kind of event to predict.
- (3) The measure of uncertainty should be additive. What this means is that if we first measure the uncertainty about rain or shine (2 possible events) and then the uncertainty about hot or cold (2 different possible events), the uncertainty over the four combinations of these events—rain/hot, rain/cold, shine/hot, shine/cold—should be the sum of the separate uncertainties.

In 1948, Claude Shannon showed that there is only one function that satisfies these desiderata.⁷⁰ This function is usually known as *information entropy*, and has a surprisingly simple definition. If there are n different

possible events and each event i has probability p_i , then the unique measure of uncertainty we seek is:

$$H(p_1, p_2, \dots, p_n) = -E(\log(p_i)) = -\sum_{i=1}^n p_i \log(p_i). \quad (5.3)$$

In plainer words, *the uncertainty contained in a probability distribution is equal to the average negative log-probability of an event.* “Event” here might refer to a type of weather, like rain or shine, or a particular species of bird or even a particular nucleotide in a DNA sequence.⁷¹

While it’s not worth going into the details of the derivation of H , it is worth pointing out that nothing about this function is arbitrary. Every part of it derives from the three requirements above. In particular, you might recognize the negative log-probability as akin to negative log-likelihood, which has been your constant companion throughout this book. These –log-probabilities do not appear here by fiat, perhaps because we wanted to develop a measure of uncertainty that we could relate to log-likelihood. Instead, we arrived at a measure of uncertainty that we will later relate to log-likelihood, by pure serendipity.

5.6.1.2. Maximizing uncertainty. Following Peter Elias’ sage advice,⁷² I’m not a fan of interpreting the function H too generally. It is what it is. But interpretations of its pieces aside, you can immediately turn H to shed new light on the omnipresent assumption of a flat or uninformative prior probability density.

You saw in Chapter 2 that a flat prior leads to the posterior being simply proportional to the likelihood. But that flat prior is also special in terms of its information content: For any finite interval, the flat uniform distribution has the greatest uncertainty. It maximizes H . You can prove this to yourself by considering an extreme case in which there are only two possible values of a parameter, θ_1 and θ_2 . Each of these values has a prior probability, $p_1 = \Pr(\theta_1)$ and $p_2 = \Pr(\theta_2)$. The information entropy of this distribution is:

$$H(p_1, p_2) = H(p, 1-p) = -p \log(p) - (1-p) \log(1-p).$$

If there is a value of p that maximizes this expression, it lies where the derivative with respect to p is equal to zero:

$$\partial H / \partial p = \log(1-p) - \log(p) = 0.$$

The only value of p that will work is the one that makes $\log(p) = \log(1-p)$, and that must be $p = 1/2$. When $p_1 = p_2 = 1/2$, this is a uniform probability distribution. The same approach generalizes to any number of discrete parameter values, and the continuous case from there.

If you have a finite range of parameter values to define a prior density to, then the uniform density is the uniquely most uncertain density you can choose. This makes it a good candidate for representing complete ignorance or indifference.

Over the full number line, from $-\infty$ to ∞ , things change. Now you have to declare some knowledge about the value of the parameter, even if it is modest. It turns out that if all you can say about the parameter's value is the mean and variance of its prior density, then the distribution that will maximize H is the normal distribution. In this way, the Gaussian distribution is another natural candidate for representing honest ignorance.

And so on. Given partial information about a probability distribution, usually in terms of its moments, a distribution can usually be found that maximizes H and still satisfies those constraints. This is what *maximum entropy* or Maxent is about, using information entropy to fill in probabilities under the assumption of ignorance. Its applications can be quite broad.⁷³

5.6.1.3. From uncertainty to accuracy. It's nice to have a way to quantify uncertainty, and therefore information as a difference in uncertainty. H provides this. But how can we use information entropy to compare model families? The key lies in *divergence*:

Divergence: The additional uncertainty induced by using probabilities specified by one distribution to try to predict observations from another distribution.

Suppose for example that the true distribution of events is $p_1 = 0.25, p_2 = 0.75$. If we believe instead that these events happen with probabilities $q_1 = 0.75, q_2 = 0.25$, how much additional uncertainty have we introduced, as a consequence of using $q = \{q_1, q_2\}$ to approximate $p = \{p_1, p_2\}$?

The formal answer to this question is based upon H , and has a similarly simple formula:

$$D_{\text{KL}}(p, q) = \sum_i p_i (\log(p_i) - \log(q_i)).$$

This is often known as *Kullback-Leibler divergence* or simply K-L divergence, named after the people who introduced it for this purpose.⁷⁴ This divergence is just the difference between two entropies: The entropy of the target distribution p and the entropy arising from using q to predict p . When $p = q$, we know the actual probabilities of the events. In that

case:

$$D_{\text{KL}}(p, q) = D(p, p) = \sum_i p_i (\log(p_i) - \log(p_i)) = 0.$$

There is no additional uncertainty induced when we use a probability distribution to represent itself. Duh!

5.6.1.4. K-L divergence and the bias-variance tradeoff. The real utility of this measure for our purpose is instead to contrast different approximations to p . As an approximating function q becomes more accurate, $D_{\text{KL}}(p, q)$ will shrink. If we have a series of candidate distributions, then the candidate that minimizes the expected divergence will be the one that maximizes expected out-of-sample prediction. Since predictive models specify probabilities (or likelihoods) of events (observations), we can use divergence to choose among model families, as long as our objective is to choose model families that perform well in prediction.

Unfortunately, to use D_{KL} to compare model families has a serious snag. What are to do about p , the target probability distribution? When we are designing a communications system, then p will be known, or at least known to a very close approximation. But when we want to find a model q that is the best approximation to p , the Truth, there is no way to access p directly. We wouldn't be doing statistics, if we already knew p .

It helps that we are only interested in comparing the divergences of different candidate q 's, different approximating functions. In that case, most of the "Truth" just subtracts out. Suppose for example we have two candidate densities q and r . The difference in K-L divergence between the two is:

$$\begin{aligned} & D_{\text{KL}}(p, q) - D_{\text{KL}}(p, r) \\ &= \sum_i p_i (\log(p_i) - \log(q_i)) - \sum_i p_i (\log(p_i) - \log(r_i)). \end{aligned}$$

A little simplification leads to $\log(p_i)$, the Truth, vanishing from this difference:

$$\begin{aligned} &= \sum_i p_i (\log(p_i) - \log(q_i) - \log(p_i) + \log(r_i)), \\ &= \sum_i p_i (\log(r_i) - \log(q_i)) = -(\text{E} \log(q_i) - \text{E} \log(r_i)). \end{aligned}$$

The difference in divergences of the two models is a difference in expected log-probability. This implies that each divergence is proportional

to:

$$D_{\text{KL}}(p, q) \propto -E \log(q_i).$$

With a little mathematical squinting, you can start to see that this is something like the negative log-likelihood of each kind of event, averaged over the true probabilities of the events. Provided the sample of events is large and representative, then just averaging over the data will take care of the expectation part. Or at least that's not what's stopping us now.

What's stopping us now is the fact of estimation. We shouldn't jump in yet and just use log-likelihood in place of $\log(q_i)$, because when it comes to fitting models, then q_i will have been trained on the observations. As a result, if we just use the log-likelihood to estimate D_{KL} , then the estimate will be biased by over-fitting. We know that it won't be biased in the other direction, towards under-fitting, because we already know that maximum likelihood always selects the most complex model family. Maximum likelihood alone protects against under-fitting. So the bias must be to artificially decrease the deviance of complex models, to over-fit. We seek to estimate a penalty that we can add to the –log-likelihood in order to estimate the relative divergence.

This penalty term is what Akaike derived. As a first order large-sample approximation, the bias of a model family \mathcal{M}_i with k_i parameters, fit by maximum likelihood, is just k_i . And so we arrive at an estimate of relative (to other approximating models) expected K-L divergence:

$$D_{\text{KL}}(\mathcal{M}_i) \propto -\log \Pr(D|\mathcal{M}_i(\hat{\theta})) + k_i,$$

where $\Pr(D|\mathcal{M}_i(\hat{\theta}))$ just means the likelihood of the maximum likelihood member of the family, as usual. Since we are adding the penalty k_i , the more complex the model family, the bigger the expected overfitting and the larger the correction. If you multiply the above relative estimate of $D_{\text{KL}}(\mathcal{M}_i)$ by 2, you end up with AIC.

AIC, with the simple penalty $2k_i$, is an approximation at the same order of accuracy as BIC. But in practice, it is almost always better to use another approximation, one that corrects for the case in which a model is so complex that the number of parameters approaches the number of rows in the data. The most common and likely most robust correction of this kind is:

$$\text{AICc}_i = -2 \log \Pr(D|\mathcal{M}_i(\hat{\theta})) + \frac{2k_i}{1 - (k_i + 1)/n}.$$

As long as there is enough data to render the likelihood function reasonably Gaussian in shape, then this correction will either do a better

job than AIC or act the same.⁷⁵ Note that as k_i grows towards n , the penalty tends towards infinity, which is what we actually want. When a model has as many parameters as there are cases in the data, then it will infinitely over-fit. At the other end, when $(k_i + 1)/n \approx 0$, then $\text{AICc} \approx \text{AIC}$.

5.6.2. Computing posterior probabilities from AIC. Just like with BIC, one use of AIC is to provide approximate posterior probabilities of model families. Since AIC, like BIC, is scaled as a deviance, the formula here is very similar to the formula on page 260. However, AIC incorporates its own priors, $\Pr(\mathcal{M}_i)$, and so those do not appear now:

$$\Pr(\mathcal{M}_i|D) \approx \frac{\exp(-\frac{1}{2}\text{AICc}_i)}{\sum_{j=1}^m \exp(-\frac{1}{2}\text{AICc}_j)}.$$

What are these hidden prior probabilities of AIC? The next section makes them explicit.

5.7. Comparing AIC and BIC

If you look in the professional literature, you'll find that individual researchers often have a strong preference for either BIC-like or AIC-like metrics. Those who favor BIC tend to favor its pure Bayesian origins. Those who favor AIC tend to reject the notion that there is any "true" model and accept instead the notion that reality is much more complex than we imagine it to be. While BIC and AIC can be used in very similar ways (see page 272), and in fact have very similar formulas, they have different objectives and their posterior probabilities have different meanings. So it pays to understand their differences.

5.7.1. Each is better when it is better. Despite wishes to the contrary, there is no universally appropriate way to compare and select models. Discarding null hypothesis significance testing does not lead naturally to another tyrannical ritual. Instead, choosing among model families must depend upon our beliefs about the large world and our intended use for models. For example, if we think—and it turns out to be true—that the phenomena of study are influenced by only a few important factors, then we should penalize complex models more strongly than if we instead believe that the phenomena are influenced by many small effects that combine in subtle ways. If we don't believe the "true" data generating model has been fit to the data, then it doesn't make sense to try to find the true model. Some newer approaches to choosing among model families, like "boosting," confront these same questions, often more explicitly. So I'm not going to end up arguing that you should always use

either BIC or AIC or anything else. There cannot be a single universally optimal way of quantifying model family uncertainty, because there is not a single purpose to which we put our models. Moreover, all metrics like BIC and AIC are approximations, and so they will be misleading in some cases, whenever the assumptions embedded in the approximations are violated. They also rely upon various large-sample assumptions, and so they will usually be unreliable when making inferences for meager amounts of evidence.

But we can say that BIC and AIC are each better under certain conditions, because of the nature of their assumptions. When the data generating model is plausibly in one of the model families fit to the data, then BIC tends to perform better than AIC. In contrast, when the data generating model is not one of the candidates fit to data, then AIC tends to out-perform BIC. More generally, BIC tries to find the true model, while AIC tries to find the best model for coping with the bias-variance tradeoff. It is routinely true that the best model in terms of out-of-sample prediction is not the true model. This is because in a finite sample of data, it may not be possible to precisely estimate all of the parameters of the true model family. The true model will be poorly estimated, and AIC will tend to select a false model that leads to better predictions. (Of course you already know that your author believes that all models are false *a priori*.) Moreover, it does not appear to be possible to develop a criterion that can do both. We have to choose between seeking truth and seeking predictive accuracy.⁷⁶

Luckily, since both BIC and AIC are easy to compute for a very broad class of statistical models, you can always compute the posteriors implied by both criteria and evaluate how different the inferences are under each. No one is going to force you to swear fealty to one approach or the other. In many cases, the implications of the posterior density implied by AIC is practically identical to that provided by BIC. Some scholars get very excited because AIC and BIC sometimes produce different rankings of model families, even when this is because AIC favors models that include weak effects with little impact on prediction. Once predictions are averaged over the posterior, the different rankings may hardly matter at all. In that case, you can assuage advocates of both approaches by pointing out that model-averaged predictions between the two approaches hardly differ. But even when the two densities are notably different, if you compute the posterior both ways, you'll at least know this and hopefully learn something additional from the data and set of model families.

In the interest of full disclosure, I favor AIC-like metrics, which mostly means in practice AICc and DIC (more on DIC in a later chapter) and some types of cross-validation (which has the advantage of working when neither AICc nor DIC are appropriate). In my own work, it is never plausible that the data-generating model is present in my analysis, because I work on rather complex phenomena for which everyone agrees that many different factors influence the outcome, often in subtle ways. When studying the evolution of a dozen taxa, the model is a phylogenetic tree, and such trees are fictions. The real history of a species is much more complex than what a tree attempts to represent. This is the kind of problem for which AIC-like metrics make sense. I also work on social science problems, in which any of hundreds of influences are plausibly related to an outcome. Indeed, unless one is working on closed physical or computational systems, in which all of the possible influences can be named and measured, I find it hard to imagine a context in which BIC makes sense. Models are never true, so it makes no sense to me to try to find a true model. Perhaps as a result of being an anthropologist, and having done several years of ethnographic fieldwork, I find it difficult to use the word *truth* in any manner which is not partly ironic. I find it easy to give up on truth as an objective, as far as statistical modeling goes, favoring instead the modest goal of improving predictive accuracy.

But there are other reasons to prefer simpler models than those that either AIC or BIC recommends. When one intends to provide a statistical model as a tool for others to use, asking them to employ a dozen variables with marginal impacts on prediction may introduce more error than merely asking them to use the top three, or even just the top one. Decision trees, for example, can encode powerful models in a simple way, but they are much simpler usually than the full statistical models that either BIC or AIC would recommend for the same data.

Whatever the most productive resolution of these issues, these are epistemological questions that statistical methodological cannot directly answer for us. While we can test AIC and BIC in simulation, all this does is verify that each metric performs better than the other, when its assumptions are met. Some other line of argument is needed to resolve the issue.

Regardless of one's position on the epistemology, however, it is possible for everyone to agree upon how BIC and AIC differ in definition. In the remainder of this section, you'll see how both BIC and AIC can be related to Bayes' theorem. This sheds light on the different assumptions about prior probability that are embodied in each measure. If you want

to minimize information divergence, then you want a prior like the one AIC provides. I am going to show you what that prior looks like.

5.7.2. Comparing priors. In Section 5.6.1.4, page 265, you saw how AIC arises from concern with information and the bias-variance trade-off. The deviance (the $-2 \log\text{-likelihood}$) of a model will be a biased estimate of K-L divergence, because of over-fitting. Therefore, in order to correct for this bias, there needs to be a penalty to the deviance. AIC provides the first-order approximate penalty, $2k$, where k is the number of parameters in the model. AICc provides a second-order estimate of the penalty, $2k/(1 - (k + 1)/n)$, where n is the sample size. These estimates stand in contrast to BIC's penalty, derived in a completely different way, $k \log(n)$.

It's not so mysterious why BIC and AIC end up with different penalties—they have different goals. But I keep saying that nearly everything in statistical inference can be related through Bayes' theorem. This implies that any difference between BIC and AIC should be understandable in terms of posterior probabilities of model families. And indeed they can. Here's the heuristic approach.⁷⁷

Typically, BIC is used with uniform prior probability. This implies that the posterior probability of a model family \mathcal{M}_i is approximately:

$$\Pr(\mathcal{M}_i|D, U) \approx \frac{\exp(-\frac{1}{2}\text{BIC}_i)\frac{1}{m}}{\sum_{j=1}^m \exp(-\frac{1}{2}\text{BIC}_j)\frac{1}{m}}.$$

The little $1/m$ factors are the uniform prior probabilities of each model family out of all m families, and I've added the explicit notation that the posterior is conditional on this uniform prior, $\Pr(\mathcal{M}_i|D, U)$. If you think about it, this has always been true. The posterior depends upon both the likelihood, which summarizes the evidence, and the prior.

Now, since the posterior probability of the same family i implied by AICc is:

$$\Pr(\mathcal{M}_i|D, P) \approx \frac{\exp(-\frac{1}{2}\text{AICc}_i)}{\sum_{j=1}^m \exp(-\frac{1}{2}\text{AICc}_j)},$$

there must be an unknown prior here, which I've denoted with P . We can find out what it is by noting that, since BIC is just an estimate of marginal likelihood, AICc must be $\exp(-\frac{1}{2}\text{BIC})$ multiplied by the unknown prior. This implies that, whatever the prior AICc assumes, we can find it by solving:

$$\exp(-\frac{1}{2}\text{BIC}_i)P_i = \exp(-\frac{1}{2}\text{AICc}_i),$$

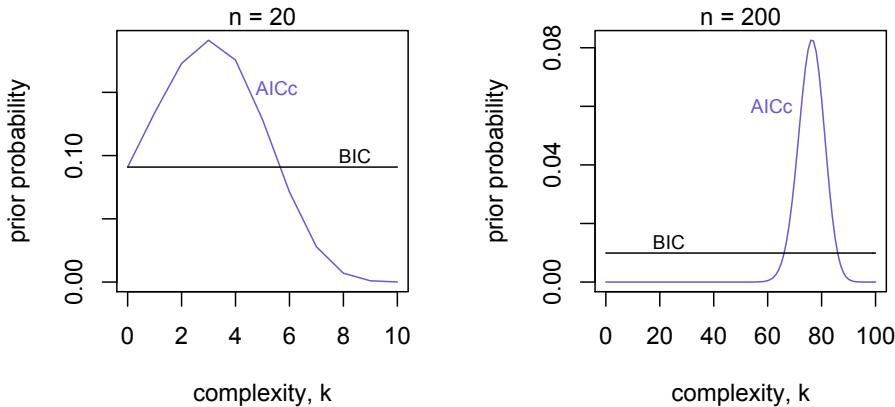


FIGURE 5.7. Comparing the priors implied by AICc and BIC. In both plots, the vertical axis is the prior probability of each model family and the horizontal axis is the number of parameters in a model family, k . The blue curve is the prior density from AICc, and the black line is the uniform prior customary of BIC.

for the prior probability P_i . Skipping over the algebra, the solution is:

$$P_i = \exp(k_i \log(n)/2 - k_i/(1 - (k_i + 1)/n)).$$

The factor P_i above isn't really a probability yet, because it hasn't been normalized so that $\sum_i P_i = 1$, but this is easy enough. The implied prior probability is just $\Pr(\mathcal{M}_i) = P_i / \sum_j P_j$.

This prior turns out to be illuminating, because it makes clear that AIC-like metrics favor models of a certain complexity, before any evidence is considered. That is to say, if all you know is the sample size and the relative complexities of the model families, then in order to minimize expected K-L divergence (maximize predictive accuracy), you should favor models of intermediate complexity. Exactly where “intermediate” lies will depend upon the sample size and the model families. FIGURE 5.7 displays the AICc-implied prior, along with the uniform prior customarily used with BIC. In both plots, the vertical axis is prior probability of a model family. The horizontal axes are model families of varying complexity, k , ordered from simplest to most complex. Each plot shows the priors for both AICc (blue) and BIC (black). The lefthand plot displays the priors when the sample size is modest, at

$n = 20$. The most complex model family here has $k = n/2$. Notice that the model family that maximizes prior probability in this case is $k = 3$. In the righthand plot, sample size is now much larger, at $n = 200$. The most complex model family is again $k = n/2$. Now the model family that maximizes prior probability is $k = 76$.

The prior is maximized at a much more complex family, at the larger sample size. Why is this? This is a consequence of negotiating the bias-variance tradeoff. For any fixed sample size, more complex models yield worse estimates of parameters. This means that the optimal complexity for prediction may well be a model with fewer parameters than even the true data-generating process. This much is just a restatement of the bias-variance tradeoff. Now as sample size increases, the accuracy with which each parameter can be estimated increases, holding the ratio of parameters to sample size constant. This effectively moves the optimal complexity, from the perspective of bias-variance tradeoff, towards more complex models. This is what the prior for AICc is demonstrating. As always, these priors can be overwhelmed by model likelihoods.

Different model families and ways of estimating models will exhibit different precise locations of such an optimal tradeoff. AICc is just an approximation for model families that have reasonably Gaussian likelihood functions. When parameters are near a boundary or likelihood functions are highly skewed, then in principle the prior we need should be different. But saying this is looking towards a future in which we have results that extend AICc to other types of models. We aren't there yet.

But comparing the different prior distributions implied by AICc and BIC can help provide intuitions about why these approaches sometimes provide different answers. It's because they are aiming at different targets. Note that BIC is not trying to minimize K-L divergence, so it has no reason to use the same prior as AICc. So BIC isn't doing anything wrong. It's just doing something different.

5.8. Using BIC and AIC

Both BIC and AIC are easy to compute and allow for estimates of posterior model probabilities. Because they produce estimates of posterior probability, they can be used to then do all of the useful inferential tasks we've already used posterior densities for, like averaging predictions over the uncertainty. In this case, the uncertainty is over model families, rather than just parameter values, but the concept is the same. You might then want to go on to make a decision to accept or reject some or all of the models, based on other criteria. But as always, the inferential step and the decision step should be clearly distinguished.

There are two major tasks people put to BIC and AIC. Both are based upon estimates of posterior probability. The first is model ranking, where models are simply ordered by posterior probability. The second is model averaging, where the predictions and estimates of the model families are averaged using posterior probability. In this section, you'll see how to do both.

5.8.1. Ranking model families. Perhaps the simplest application of a metric like BIC or AICc is to rank model families by posterior probability. In the event that one of the model families has much higher posterior probability than all of the others, then it may not be necessary to do anything more. However, even when this is not true, reporting the posterior probability of each model family provides information about the uncertainty in model structure.

Let's return to the primate milk data that you first encountered in Chapter 4. Load the data table with:

```
library(rethinking)
data(milk)
d <- milk
```

R code
5.10

You'll consider the hypothesis again that neocortex percent predicts the energy content of the milk (`kcal.per.g`), but now in a model comparison framework. There are two hypotheses to model here. First, some scholars think that primates with a lot of neocortex need to produce higher-energy milk. Second, it may be that any relationship between milk energy and neocortex is only apparent after controlling for body mass, which also affects milk energy.

These two hypotheses lead us to consider four different models, which are just four different combinations of the prediction variables `neocortex.perc` and `mass`:

- (1) First, consider the model that ignores both neocortex percent and mass. This is a kind of null model. Its equation for the mean of the normal distribution of `kcal.per.g` is just $\mu_i = \alpha$. In other words, it is constant. Call this model M_1 .
- (2) Second, consider a model with only `neocortex.perc`, in the equation for the mean: $\mu_i = \alpha + \beta_n n_i$. Call this model M_2 .
- (3) Third, consider a model with only the logarithm of body mass, `log(mass)`, in the equation for the mean: $\mu_i = \alpha + \beta_m \log(m_i)$. Call this model M_3 .
- (4) Finally, consider the model with both `neocortex.perc` and `log(mass)` as main effects. This model evaluates the second

hypothesis. Its equation for the mean is: $\mu_i = \alpha + \beta_n n_i + \beta_m \log(m_i)$. Call this model M_4 .

In all four cases, the stochastic node is still just $c_i \sim \text{Normal}(\mu_i, \sigma)$, where c_i is the i -th value of `kcal.per.g`. In the rest of this section, we'll fit these models and see how to rank them by BIC and AICc.

5.8.1.1. Beware dropped rows. The first lesson to internalize here is that one must be very careful when comparing models to ensure that exactly the same outcome values are being predicted by each model. Otherwise, there will be potentially large differences in log-likelihood of the models that stem from only the fact that some were fit to more data than the others. This will make sense in the context of the milk energy example.

Take a look at the `neocortex.perc` column in the data frame `d`. Notice the missing values, marked by `NA`. Now if you use a convenience command like `lm` (or `glm` or `glmer`) to fit a model, R will first drop all of the rows that contain missing values, `NA`'s. Then it will fit the model on the remaining rows only. If one of your models contains a variable, like `neocortex.perc`, that has missing values, then that model will be forced to predict fewer outcome values than the models that do not include `neocortex.perc`. This further means that when you go to compute BIC/AICc, the values of these metrics will suggest that the models with `neocortex.perc` perform better than they actually do, because they dropped rows from the data.

BIC/AICc and similar metrics can be used to compare only models that predict exactly the same number and identity of outcomes. The best way to ensure this is true is to either (1) drop rows with missing values before you fit any models or (2) somehow impute missing values, so you can use all of the rows with all of the models. Imputing missing values is a tricky business, but it is a straightforward application of Bayesian estimation, which you'll see in a later chapter. For now, we'll forge ahead by dropping all of the rows with any missing values. This is a good line of code to keep handy, when you start a model comparison analysis:

```
R code  
5.11 dd <- d[ complete.cases( d ) , ]
```

You now have a new data frame, `dd`, that contains 17 rows instead of 29. It's a shame to waste data like this, but models must be compared fairly, and that means comparing their abilities to predict the same data.

Now you're ready to fit the models, using a convenient function like `lm`. If you use `mle2` instead, you'll have to do the same thing, but `mle2`

won't drop rows for you; instead it'll give you a cryptic error and abort. Here's the simple approach, using `lm`:

```
M1 <- lm( kcal.per.g ~ 1 , data=dd )
M2 <- lm( kcal.per.g ~ neocortex.perc , data=dd )
M3 <- lm( kcal.per.g ~ log(mass) , data=dd )
M4 <- lm( kcal.per.g ~ neocortex.perc + log(mass) , data=dd )
```

R code
5.12

The only trick to note here is the `kcal.per.g ~ 1` in the first model family. This is the null model, so to speak, because it models the implied hypothesis that neither neocortex nor body mass helps us predict milk energy. The "1" after the squiggle (tilde) indicates an *intercept*, the α in the equation for the mean. In the other models, like `kcal.per.g ~ log(mass)`, the intercept is implied automatically. If this notation is new to you, look back in Chapter 4, in the section on OLS.

5.8.1.2. Computing BIC/AICc. Now you are ready to compute BIC/AICc and rank the model families. Most of the time, you can just use the commands `BIC` and `AICc` to compute these metrics from a fit model. So for example to compute a list of AICc values for the models, you could use `sapply` like this:

```
library(bbmle)
AICcs <- sapply( list(M1,M2,M3,M4) , AICc )
AICcs
```

R code
5.13

```
[1] -7.601156 -5.028031 -6.891948 -14.022083
```

Notice that all four AICc's are negative. There is nothing wrong here. AICc's and BIC's too can be negative in value, because of how likelihoods are defined mathematically. This situation is very common when estimates of σ are small, as they are here. Just be aware that negative AICc's and BIC's are just like positive ones: the smaller values indicate better fits. In this case, that means the more negative ones. And so reading the above AICc's, model M_4 has the lowest AICc, followed by model M_1 , and then by M_3 and finally M_2 .

You can also compute posterior probabilities from these values, just using the formula from earlier in the chapter:

```
AICcs <- AICcs - min( AICcs )
post.AICc <- exp( -1/2 * AICcs )/sum( exp( -1/2 * AICcs ) )
post.AICc
```

R code
5.14

```
[1] 0.03735770 0.01031889 0.02620460 0.92611881
```

Again, the model with the best AICc will have the highest posterior probability. In this case, that is model M_4 , by a large amount.

In the code block just above, notice that the first thing I did was subtract the smallest (best) AICc from each model family's AICc. This isn't always necessary, but it has numerical advantages, so I recommend always doing it. This is indeed what pre-packaged functions like **AICctab** (next paragraph) always do, because there is no harm in doing it, but sometimes great harm in not doing it. Why can not doing this subtraction cause problems? Numerical problems can arise when the AICc values are very large numbers, because the result of exponentiating a very small negative number is usually rounded to zero, in the computer. So you might end up asking R to compute the value of zero divided by zero, and that's no good. Since it's only the relative values of AICc (and BIC) that matter, subtracting the smallest AICc from each AICc removes the meaningless absolute value common to all of them.

It's much more convenient to use the commands **AICctab** and **BICtab** provided by the **bbmle** library. These command just automate the computations above and collect the results into a nice table. Here's an example:

R code
5.15

```
library(bbmle)
AICctab(M1,M2,M3,M4, weights=TRUE , nobs=nrow(dd) )
```

	dAICc	df	weight
M4	0.0	4	0.9261
M1	6.4	2	0.0374
M3	7.1	3	0.0262
M2	9.0	3	0.0103

In addition to doing the calculations, **AICctab** will sort the model families from highest posterior probability to lowest. The parameter **weights=TRUE** forces **AICctab** to compute posterior probabilities, the column it calls **weight** in the output. (In the scientific literature, these posterior probabilities are often called *Akaike weights*.) We also have to tell **AICctab** the number of observations in the data, which is passed via **nobs**. Above, the code just took the shortcut of using the number of rows in the data frame, which is equal to the number of observations. In later chapters, with other kinds of models, this will not be true. But it's safe for Gaussian models. For information on the other columns, see the help **?AICctab**.

Go ahead and try the BIC-equivalent, **BICtab**. You should find that the rankings are in the same order, although the posterior probabilities

will be a little different. In particular, BIC likes the complex model family \mathcal{M}_4 a bit more than AICc does.

5.8.1.3. *Interpreting model rankings.* So what does one do with such a table? Let's repeat the ranking table, for ease of reference.

	dAICc	df	weight
M4	0.0	4	0.9261 neocortex.perc + log(mass)
M1	6.4	2	0.0374 1
M3	7.1	3	0.0262 log(mass)
M2	9.0	3	0.0103 neocortex.perc

I've manually annotated each line to show the model structure in each case. There is substantial evidence in favor of \mathcal{M}_4 , the model family including both neocortex percent and body mass. More than 90% of the posterior probability is assigned to this model family. However, experience has taught me that this isn't as clear a victory for \mathcal{M}_4 as you might think it is. In the next section, you'll see that even though the other models have low posterior probabilities, all less than 5%, they can still have noticeable effects on prediction. Ignoring them is not the conservative approach.

What summary can be made of these rankings, then? The clearest inference is that no model containing only `neocortex.perc` or `log(mass)` does nearly as well in expected out-of-sample prediction (remember AICc's objective!) as the model containing them both. In fact, the second-highest ranked model family is just the null model, \mathcal{M}_1 , which includes neither predictor variable. This suggests that it would be better in fact to ignore both than to use either alone.

Be careful not to make inferences about the importance of each predictor variable from a table of this kind. You still need to inspect parameter estimates and make prediction plots, now perhaps averaged over model families, in order to talk about the influence of each predictor on the outcome. That's where we turn next.

5.8.2. Model averaging predictions. Just ranking model families by posterior probability provides information about model uncertainty, just like posterior probability of parameters provides information about parameter uncertainty. A lot can be inferred just from comparing rankings and relative posterior probabilities. But a lot more can be done, as well. Just like the posterior over parameters can be used to quantify the uncertainty in model predictions, so too can the posterior over model families be used to quantify the uncertainty in predictions that arises from uncertainty in model family. In this way, you can calculate approximate confidence intervals that incorporate model family uncertainty, just like

you calculated confidence intervals from the posterior within a model family. This approach is often referred to as *model averaging*, and it is a simple logical extension of posterior probability of parameter values.⁷⁸

We're going to work through three distinct steps in estimating and using the AICc-implied posterior across model families:

- (1) Sample parameter values from the *model averaged* joint posterior of all the model families.
- (2) Use the samples from (1) to produce model-averaged parameter estimates.
- (3) Use the samples from (1) to produce model-averaged predictions.

The second step, averaging parameter estimates, is actually something I'm going to try to warn you off doing. You don't need to do it, to complete step three. But since it pops up in the scientific literature, you should know what it is.

In the examples, I will work with AICc-implied posterior probability estimates. But all of the examples work for BIC exactly like they do for AICc. Just swap out AICc for BIC everywhere in the code.

5.8.2.1. *Sampling from the posterior across model families.* In earlier chapters, you saw how to use samples from the naive posterior of a fit model to average predictions over their uncertainty. This made for an easy and transparent way to produce 95% prediction intervals that incorporate all the covariances among parameters. Now that we have a posterior probability density across model families, we can use it just like we used the naive posterior across parameter values in earlier chapters. This posterior allows us to simultaneously take into account the uncertainty in both model structure and parameter values. Instead of sampling from only the posterior distribution of parameters, we'll first sample a model family, using the posterior probabilities you computed just above. Then, once you have a model family in hand, you sample from its posterior for parameters. Do this process again and again, each time choosing another model family at random from the posterior probabilities of the model families and then sampling at random from the posterior distribution of parameter values. After many rounds of sampling, you'll have a collection of parameter values that represent both the uncertainty about model family and about parameters.

This procedure is not conceptually too difficult—it's just two sequential sampling steps, repeated a large number of times. But the code for undertaking this procedure is a bit opaque, mainly because some

tricks are needed to make the columns of the resulting table of samples line up correctly. The LIBRARY of code that accompanies this book provides a working algorithm for you, though. The same `sample.naive.posterior` command you've been using can also produce samples across model families, if you give it the right inputs. This line of code will do the job:

```
post.avg <- sample.naive.posterior( list(M1,M2,M3,M4) ,
n=40000 , method="AICc" )
```

R code
5.16

I've named this `post.avg` to indicate that it is a posterior sample of parameters, averaged over model families. Instead of passing a single fit model, you give the function a list of all the models you want to use in constructing the posterior. You also have to tell `sample.naive.posterior` what method you want to use in estimating the posterior across model families. Here, you've told it to use AICc. So it computes the AICc (using the `AICc` command) for each model family and builds the posterior just as you did earlier. (The command `sample.naive.posterior` will guess the number of observations for you, but you can also tell it manually that `nobs=17`, just like with `AICctab`.)

Notice that you've told it to produce 40-thousand samples, rather than the default of 10-thousand. When sampling across model families, you may need many more samples in order to adequately sample the tails of the posterior of each model family. There is no hard rule about how many samples are enough—it will always depend upon the exact context and number of model families. But you'll know when you have sampled enough, when the confidence intervals you compute from your samples stop changing. I like to start with 10-thousand samples times the number of model families (hence the 40-thousand used above) and then double that number and see if it changes the computed intervals.

Now for a bit of bookkeeping convenience. Because we fit these models using `lm`, the columns in `post.avg` have rather inconvenient names. They are named after the variables (factors) and not the parameters. We can rename the columns after the parameters with this line:

```
colnames(post.avg) <- c("a","b.n","b.m")
```

R code
5.17

In doing this, I'm suggesting to the reader that you want to be mindful at all times that you are working with estimated parameters, not predictor variables. It is customary for people to talk about parameters as

“the effect of” a variable of interest. In the simplest models, you can get away with such talk. But this kind of thinking can routinely lead to mistakes. When you get to interactions in Chapter 6, you’ll see that the “effect” of changing a predictor variable depends upon more than a single parameter. In those cases, it can be very misleading to call any individual parameter estimate “the effect of” a variable, because the variable is multiplied by more than one parameter, within the model. So I’m encouraging you to adopt my preference here for naming parameters by not calling them after any predictor variable, because it can help you avoid interpretation fallacies down the road.

5.8.2.2. *Computing model averaged parameter estimates.* In the special case of truly linear models like these, with Gaussian stochastic nodes and additive equations for their means, it is possible to average parameter estimates themselves, averaging over model families to produce parameter estimates that incorporate uncertainty about model family. I am not a fan of this kind of averaging. Technically there is nothing wrong with it. In practice, however, it is hard to interpret model-averaged coefficients, partly for reasons that will be transparent once you meet interactions in Chapter 6.

A secondary concern is that many people seem tempted to use the model-averaged standard errors of each parameter to engage in some kind of significance test. Usually this means concluding that the best value of a parameter should be zero, just because the 95% confidence interval of the parameter includes zero. Every time I’ve taught model averaging, someone invents this fallacious practice, even when I warn them off it. Here’s the problem with it. The logic of model averaging is that the estimated posterior probabilities of each model family quantify uncertainty about model structure. The approach has deep information theoretic roots (if you use AIC-like metrics) and does what it advertises: it adjusts model predictions so they perform better out-of-sample. If you then use the fact that the 95% (or any other arbitrary coverage) model-averaged confidence interval of a parameter includes zero to decide that a variable shouldn’t be included in a “best” model family, you are engaging in a schizophrenic exercise. You have already adjusted predictions to account for uncertainty about the best value of a parameter. Doing so again, using an arbitrary criterion that prefers the value zero (remember Chapter 3’s lesson about NHST being functionally equivalent to having a strong prior preference for the null hypothesis), is logically incompatible with using AIC and BIC at all. For now, I do want to explain the process of model averaging parameter estimates, however. But then I’ll

turn to model averaging *predictions* instead, which is a much more general and powerful approach.

Here's the logical path to and method for averaging parameters. In a linear model, the prediction for any case i is an additive model like:

$$\mu_i = \alpha + \beta_1 x_i + \beta_2 z_i.$$

The posterior probabilities reference the *predictions*, and μ_i is the predicted average. In this case, because the equation for μ_i is additive, the average μ_i across samples from the posterior is the same as averaging each parameter—each column in the posterior—and then computing the mean. That is to say that if we take the expectation across models, which is the same as averaging over the posterior probabilities of models, then:

$$E(\mu_i) = E(\alpha + \beta_1 x_i + \beta_2 z_i) = E(\alpha) + E(\beta_1)x_i + E(\beta_2)z_i.$$

Taking the average of the μ_i 's for each model family is equivalent to taking the average of each parameter and then adding the averages together. But this is only true when the equation for the mean is additive, as it is in typical linear Gaussian models. In the non-linear models of later chapters, this won't work, unless you get lucky. As always, getting lucky is not a methodology.

But when we are working with linear nested models like our M_1 through M_4 , there's no necessary harm in averaging parameters. This leads to a way of summarizing averaged linear models, using a table of model-average parameter estimates. Predictions can be generated from such a table of averaged estimates, and those predictions will be the same as those you learn to compute in the next section, in which you'll see how to use the more-general approach.

Since the average value of α from the model-averaged posterior, `post.avg` in this case, is what is needed to compute the model-averaged prediction, we can use this average α and its quantiles to summarize how uncertainty across model families affects inferences about α . To compute the model-averaged value of α , using the samples from the posterior, just use:

```
mean( post.avg$a )
```

R code
5.18

```
[1] -0.9583595
```

And to compute the 95% HPDI of the intercept, averaged over model families, use:

R code
5.19

```
HPDI( post.avg$a )
```

	lower	upper
a	-1.9610761	0.7486974

All of the other parameters can also be averaged in this way. You can build up a table of model-averaged estimates by repeating this procedure for each column in the samples from the posterior. A quick way to do this is to use the very powerful `apply` command:

R code
5.20

```
coef.means <- apply( post.avg , 2 , mean )
coef.hpdi <- apply( post.avg , 2 , HPDI )
```

This just applies some operation—the mean or the quantile, respectively—to each column of the posterior samples. Whereas `sapply` only works on each element of a vector, `apply` can work on entire rows or columns of a matrix. This makes it extremely powerful for all sorts of calculations. Check the help `?apply` for details.

Let's compare the model avaraged parameter estimates to the estimates from only the highest-ranked model, M_4 . First, I'm going to compute model averaged standard errors and then make a quick display table of all of the model averaged estimates, using a trick with the `rbind` command:

R code
5.21

```
coef.se <- apply( post.avg , 2 , sd )
t(rbind( coef.means , coef.se , coef.hpdi ))
```

	coef.means	coef.se	lower	upper
a	-0.95728258	0.67346694	-1.9613398	0.74551574
b.n	0.02589740	0.01057273	0.00000000	0.04113058
b.m	-0.09018515	0.03308953	-0.1366103	0.00000000

That `t()` around the `rbind` command just rotates the resulting table, so that rows are columns and columns are rows. This will let us more easily compare to the `precis` output for model M_4 , the highest-ranked model family:

R code
5.22

```
precis( M4 )
```

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	-1.08525385	0.515280719	-2.09518550	-0.07532220
neocortex.perc	0.02793065	0.008015062	0.01222141	0.04363988
log(mass)	-0.09640223	0.024748549	-0.14490849	-0.04789596

Keep in mind that `(Intercept)` is equivalent to a/α , `neocortex.perc` is the same as $b.n/\beta_n$, and `log(mass)` is $b.m/\beta_m$. Comparing the two tables, you can see that the estimates themselves hardly move at all, as a result of averaging. This is because model family `M4` has more than 90% of the posterior probability. But notice that the standard errors in the first table, the model averaged estimates, are noticeably larger. And the confidence intervals for both `b.n` and `b.m` now extend all the way to zero, after averaging. This is quite typical: model averaging will tend to have more of an effect on confidence intervals than it will on central estimates. This is because confidence intervals, especially wide ones like 95% intervals, are about extremes. Even low-probability model families can have big effects on extreme predictions, because extreme predictions are about low-probability events.

Keep in mind that this procedure is not going to be valid for any non-linear model. The reason will be clearer, once you meet non-linear models in later chapters. But the short explanation is that the posterior probabilities of model families reference the predictions of those model families, on the scale of measurement of the outcome variable. In a linear model, as explained above, there is a straightforward additive relationship between the parameters and the predictions. For a linear model with only main effects, like the ones here, each β -type coefficient can be interpreted as the change in the outcome per unit change in the corresponding predictor variable. But this is never exactly true, in a non-linear model. Indeed, even in completely linear models, once there are *interactions*, it can be hard to understand the influence of a variable on outcomes by just inspecting the corresponding parameter estimate. We'll tackle that problem in the next chapter. In non-linear models, it is essentially true that all of the predictor variables always interact with one another, at least a little, and that is one reason why inference from them is more difficult.

5.8.2.3. Computing model averaged predictions. While model averaging estimates only works for additive models, that doesn't stop us from computing model averaged predictions, for any kind of model. Remember, the posterior probabilities implied by AICc or BIC reference patterns of prediction. And so it can be appropriate to average predictions by these probabilities, even when the parameters in a model are not even remotely additively related to the predictions. Model averaged estimates, in turn, only make sense when they yield the same results as averaging predictions does.

This section shows you how to compute and plot model averaged predictions and confidence intervals that incorporate all of the complex

covariances among parameters. We're going to plot the model-averaged predicted mean milk energy, for each body mass and neocortex percent, as well as the model-averaged 95% confidence intervals of these means. The procedure is really the same as it was in previous chapters. So the code in this section will look very familiar, even though the application is a little more complicated. As always, the computer will do all of the hard work for you.

First, to compute predicted mean in previous cases, all you had to do was use the equation for the mean, μ_i , for the isolated model family at hand. So you could just extract the maximum likelihood estimates and plug them into the equation for μ_i and use a command like `curve` to plot the result. But now we have 4 model families, so we have 4 different equations for μ_i . So we need to average over these individual predictions for the mean. Thankfully, the naive model-averaged posterior in `post.avg` makes this easy. Here's some code that will produce model-averaged predicted means, varying body mass but holding neocortex percent constant at the sample mean:

R code
5.23

```
neocortex.seq <- 50:80
mu.log.mass <- mean( log(dd$mass) )
mu.avg <- sapply( neocortex.seq , function(z) mean(
  post.avg$a + post.avg$b.n*z + post.avg$b.m*mu.log.mass ) )
```

As you've done earlier in the book, first decide upon a range of horizontal axis values (body mass values) to compute the mean across. In this case, we choose 50 through 80, because that's the range of percents in the sample. Next, we're varying neocortex percent here, holding body mass constant, so we need a value for body mass. It's useful to compute the average mass that we'll use in the equation for the mean, so the code stores this value in `mu.log.mass`. Then apply each neocortex percent value to a function that computes the mean μ_i for each sample (row) in the posterior and finally returns the mean of these μ_i 's. That gives us a model averaged prediction for the mean milk energy as a function of neocortex percent, holding mass constant.

Now to compute the confidence interval around that mean. This is just a matter of replacing the `mean` command above with your old friend `HPDI` (or use `PCI` if you want simple percentile intervals):

R code
5.24

```
mu.avg.ci <- sapply( neocortex.seq , function(z)
  HPDI(
    post.avg$a + post.avg$b.n*z + post.avg$b.m*mu.log.mass ) )
```

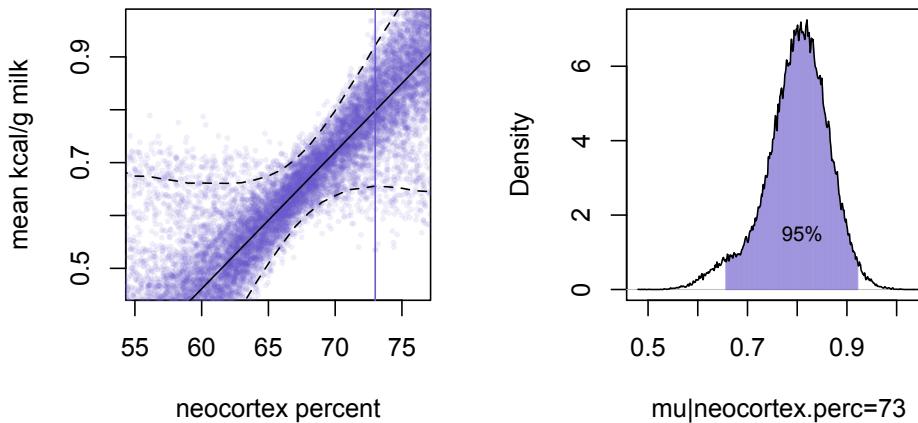


FIGURE 5.8. Samples from the model averaged posterior, based upon AICc-implied posterior probabilities of each model family. The black regression line in the lefthand plot is the mean model-averaged predictions for μ . The black dashed curves are the 95% HPDI's of the mean. The vertical blue line in the lefthand plot shows the location of the posterior slice shown in the righthand plot. On the right, the distribution and 95% HPDI of the model averaged posterior, where `neocortex.perc` equals 73.

You should be ready by this point in the book to plot the above estimates without my help. If you need a hint, look back at examples in the previous chapter. You should also be able, with a little effort, to modify the code above to compute model averaged predictions for the other predictor variable, `log(mass)`.

But before plotting these mean lines and confidence boundaries, as you've done before, it's worth pausing for a moment to look directly at the samples from the posterior. Because we're averaging over models, even though we're using the convenient large-sample assumption that the posterior for each model family is multivariate Gaussian, this does not imply that the model averaged posterior is Gaussian. Indeed, it is very non-Gaussian. FIGURE 5.8 plots samples from the model averaged posterior, `post.avg`, used to calculate μ_i across the full range of `neocortex.perc`. The mean μ_i and HPDI's you just calculated are shown

by the solid black line and dashed curves, respectively. In the lefthand plot, you can make out a distinct ridge along the diagonal. These are the samples derived mainly from \mathcal{M}_4 , the highest ranked model family. But there is also a very long tail of samples from the other models, extending above the ridge at low values of `neocortex.perc` and above the ridge at high values of `neocortex.perc`.

The righthand plot in FIGURE 5.8 shows the cross-section of a slice through the model averaged posterior, taken at the location of the vertical blue line in the lefthand plot, where `neocortex.perc` equals 73. In this profile, it is easy to make out the long tail on the left, resulting from considering the low-rank model families. This long tail has a strong impact on interval calculations, as can be seen by the range of the shaded 95% HPDI. This is why the dashed boundaries flare out so much in the lefthand plot. You can make a plot like this one yourself, sliced anywhere you like, with:

R code
5.25

```
pts <- post.avg$a + post.avg$b.n*73 + post.avg$b.m*mu.log.mass
dens( pts , show.HPDI=0.95 )
```

Just change the 73 in the first line of code to move the slice. Note that the `dens` function is part of the LIBRARY that accompanies this book.

As a result, the confidence intervals for the model averaged predictions are much wider than those for the highest-ranked family alone. FIGURE 5.9 compares the mean μ and intervals you calculated earlier to the same sort of predictions based upon only the highest-ranked model, \mathcal{M}_4 . The black regression lines and dashed curves in each plot are the model averaged predictions and 95% confidence intervals computed just now. The light blue lines and dashed curves are the predictions and 95% intervals for only \mathcal{M}_4 . While model averaging has had little effect on the average predictions—the blue lines and black lines are very similar—it has had a much bigger effect on the confidence intervals. The fact that model family \mathcal{M}_4 has more than 90% of the posterior probability has not prevented the other models from casting considerable uncertainty on the question of how shallow the regression slopes really are. 95% intervals are about extreme events, and so model averaging quite often has consequences of this kind. This isn't a flaw with the procedure. Instead, it's a feature. You have to embrace the uncertainty and propagate it throughout an analysis, in order to do statistics honestly.

5.8.3. Why not just use the best model for prediction? This example gets at a very important and frequent question about model comparison. Whenever I teach this material, someone always asks, “Why not just

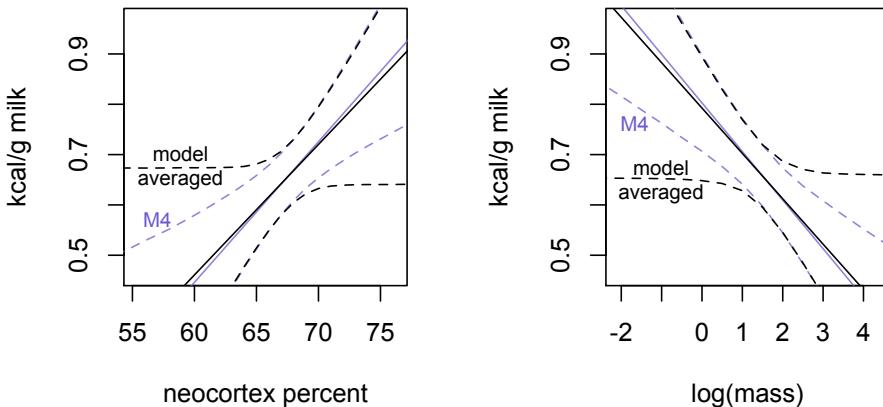


FIGURE 5.9. Model averaged predictions, based upon AICc-implied posterior probabilities. The black regression lines in each plot are the model-averaged mean (μ_i) predictions for the primate milk energy models M_1 through M_4 . The black dashed curves are the 95% intervals of the mean, again model-averaged. The blue lines and dashed curves in each plot are instead the predicted means and 95% intervals for just the highest ranking model family, M_4 . On the left, mean predicted milk energy as a function of neocortex percent, holding body mass constant at the sample mean. On the right, mean predicted milk energy as a function of log body mass, holding neocortex percent constant at the sample mean.

use the best model?” The “best” model in this context means the model family with the highest posterior probability (smallest AICc). I’ve also had reviewers and editors ask me the same question. This is a good question. There are a couple ways to answer it.

First, I hope the example above convinces you that model averaging is a conservative approach that preserves the uncertainty about model family. Using only the best model family throws away that uncertainty. The point of averaging over model families is the same as averaging over parameter values—we wish to honestly represent the uncertainty. Using just the model family with the largest posterior probability would be like using only the maximum likelihood estimates. It is routinely true

that the “best” model still has a posterior probability much less than 1. In that case, if you generate predictions from only the best model family, you will substantially underestimate the variability of the predictions implied by your analysis. Simulation studies—in which we know the “true” data generating model—indicate that model-averaged predictions of this kind can have lower average prediction error.⁷⁹ In do-or-die cases like population viability analysis, taking account of unlikely-but-disastrous events is crucial to preserving populations. The average event isn’t our interest, and so even unlikely models are suddenly very interesting.

Second, sometimes you really do want to select the single best model. But keep in mind that the inference step (producing the posterior) and the decision step are distinct. Posterior probability provides advice on choosing a model, if we must choose one. But many times, there’s nothing extra to be gained from claiming to “select” a best model. Once you’ve reported the estimated posterior probabilities, it’s usually little more than Popperian ritual to pretend that you have “tested” the models and chosen a winner. Our predictions and risk analysis based upon the statistics must still account for uncertainty over model families. When one model family is a clear winner, with say more than 99% of the posterior probability mass, then the model-averaged predictions will be almost entirely a product of a single model family. In this way, averaging predictions automatically accounts for and honestly represents the results of very decisive “tests.” This also protects you against making the mistake of missing the effect low-probability models can have on confidence intervals, as exemplified by FIGURE 5.9.

5.8.4. Do not perform significance tests with BIC/AICc. A related concern to using only the best model lies in using BIC’s or AICc’s to perform some analog of significance testing. The Tyranny of Fisher motivates many students to ask, “How big of a difference in AICc is significant?” Similarly, many students ask if a posterior probability greater than 0.95 means that the top-ranked model is significantly better than the others. When I first started publishing papers with AICc in them, reviewers and editors asked the same questions. I don’t fault anyone for asking these questions, given that professional training has lead most scientists to expect that the goal of statistics is to provide significance tests. But the reader already knows how I’m going to answer.

BIC/AICc should not be used to perform any kind of significance test. Most of what was said in Chapter 3 still applies here. Theory cannot recommend any threshold difference in posterior probability for

choosing one model family over all others. The 5% significance threshold is arbitrary and so would be any other threshold. Given the approximations that go into computing BIC and AICc, it makes no sense to ritualize the acceptance and rejection of model families based upon some superstitious number. Model families with less than 5% posterior probability can also have important consequences for prediction, especially when we are interested in extreme events. And so the level of posterior probability below which a model family can be safely ignored will always depend upon the context and objective.

The same reality applies to ordinary P -values and confidence intervals, of course. But you already know that story.

5.9. Comparing Estimates

While formal methods of comparing model families—AIC, BIC and the like—are powerful, there are some inference tasks that they are poorly suited to. In particular, when all one wishes to show is that the estimated effect of a predictor variable is *robust* to the inclusion of various other predictor variables, then comparing estimates is a simpler approach that will get the job done. In some fields, like economics, this is indeed the dominant way of comparing structurally different models. Nearly every econometric analysis contains a table of estimated coefficients.

There are at least two reasons that putting all of the possible confounds into a single model may not work. First, sometimes there are just too many potential predictor variables, even more than there are rows in the data frame. This circumstance happens all of the time, when working with archival data sources. Consider international comparative work. It's easy to acquire a large number of statistics on individual countries, but since the number of countries in the world is fixed, at some point the number of variables will exceed the number of cases. Long before you reach that limit, the model will become unreliable, if you just include everything at once. Second, many potentially confounding variables are highly correlated with one another. As a result, you cannot easily include them all in the same model, or you will encounter the inflated standard errors and unreliable estimates that come with flirting with multicollinearity.

For both of these reasons, it is sometimes and even usually necessary to consider potentially confounding variables in smaller batches. What you'd like to know is which, if any, of the batches of variables change the inferences about the predictor variables of primary interest.

Everything makes more sense with an example. So here you'll meet a set of data that you'll work with again in the next chapter. Go ahead and load the table:

R code
5.26

```
library(rethinking)
data(rugged)
d <- rugged
```

Each row in this data frame is a country, and the various columns are economic, geographic and historical features.⁸⁰ We'll be interested in predicting economic development as a function of, get ready for it, how rugged the terrain is in a country. The variable `rugged` is a Terrain Ruggedness Index⁸¹ that quantifies the topographic heterogeneity of a landscape.

If you inspect the distribution of `rugged` in the data, you'll see that it has a long right tail, due to a few very rugged countries, like Nepal and Switzerland. An explanatory variable with a skewed distribution like this is usually a good candidate for pre-analysis transformation. The reason is that a predictor variable that flares out on either side is unlikely to be linearly related to an outcome variable. In this case, I'm going to skip this step, both for the sake of simplifying the presentation and because it makes little difference in this case. After working through this section, it would be instructive to come back to this point, use a log or square-root transform on `rugged`, and then repeat the regression analysis. Showing that conclusions are robust to common transformations of the predictor variables is a good check, anyway.

For reasons that will become clear in the next chapter, let's work on only the subset of the data that excludes African countries. The variable `cont_africa` is a dummy variable that indicates when a country is in Africa. So to make a new data frame that excludes African nations:

R code
5.27

```
dd <- d[ d$cont_africa==0 , ]
```

Read this code as *give me all columns of all of the rows in d where cont_africa is equal to zero*. This is the data frame you'll work with for now.

The hypothesis to investigate here is that rugged terrain hampers economic development. The outcome variable will be the logarithm of `rgdppc_2000`, which is the real gross domestic product per capita, measured in the year 2000. So the basic model needed to evaluate the relationship between this measure of economic development and the ruggedness of the terrain in a country is:

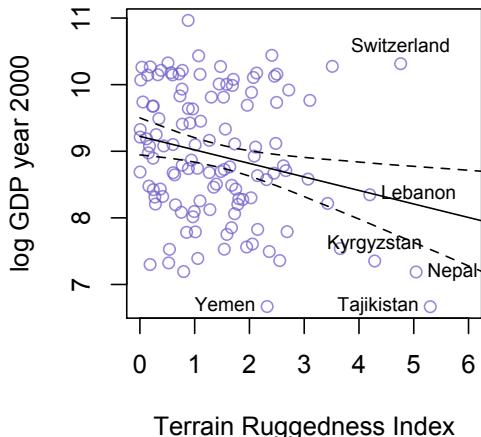


FIGURE 5.10. Real gross domestic product per capita and terrain ruggedness for 121 countries. The solid line is the maximum likelihood estimate of the relationship, and the dashed curves are the 95% confidence interval of the mean relationship.

```
m1 <- lm( log(rgdppc_2000) ~ rugged , data=dd )
precis(m1)
```

R code
5.28

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	9.2232264	0.14146421	8.9459616	9.50049111
rugged	-0.2028571	0.07839806	-0.3565145	-0.04919972

I plot these estimates, with 95% confidence interval of the mean, in FIGURE 5.10. Without consider any confounding factors, it looks like rugged countries do have lower economic development.

But can this relationship survive including other predictor variables? Maybe ruggedness is just correlated with something more fundamental, like total area or soil quality? If so, then including some additional predictor variables might knock the estimate of `rugged` down in magnitude or precision. In fact, let's consider five batches of predictor variables.

- (1) `log(land_area)`: Maybe ruggedness is standing in for small countries, and the real causal variable here is just the size. If so, then including area in the model should knock out `rugged`, or at least diminish the magnitude of its estimate. You'll use the logarithm of area here, which is common.
- (2) `soil`, `desert` and `tropical`: These variables are percents of fertile soil, desert and tropical climate in each country. Perhaps these are more fundamental causes of economic activity than

terrain ruggedness. If so, then including them should again knock out ruggedness.

- (3) `dist_coast`: Maybe ruggedness is standing in for lack of access to the coast. It has long been appreciated that being landlocked is bad for business.
- (4) `colony_gbr`, `colony_fra` and `colony_prt`: It is hard to discuss differences in economic development among nations without taking some account of colonial history. Great Britain, France and Portugal were major colonial powers, so we include dummy variables indicating if a country was ever a colony of one of these powers.
- (5) `cont_europe`, `cont_oceania`, `cont_north_america` and `cont_south_america`: Finally, let's consider that nations on the same continent likely share economic fates, for complex reasons we may not have the variables for. In that case, we fall back on group dummy variables for continent, to try to soak up some of the correlations within continents and again knock out ruggedness, which is also likely correlated between neighbors. We'll let Asia be the aliased category assigned to the intercept.

Again, you could try to include all of these batches in a single model, attempting to control for everything at once. But that won't go well. If you don't believe me, I encourage you to give it a try on your own. Let's forge ahead and fit five more models, in the meantime, one for each batch of control variables. So each of these five models will contain `rugged` as well as a batch of control variables. Here's the code you need:

R code
5.29

```
m2 <- update( m1 , . ~ . + log(land_area) , data=dd )
m3 <- update( m1 , . ~ . + soil + desert + tropical , data=dd )
m4 <- update( m1 , . ~ . + dist_coast , data=dd )
m5 <- update( m1 , . ~ . + colony_gbr + colony_fra + colony_prt ,
    data=dd )
m6 <- update( m1 , . ~ . + cont_europe + cont_oceania
    + cont_north_america + cont_south_america , data=dd )
```

I've slipped a new modeling command in here: `update`. What `update` does is take a previously fit model, like `m1`, and add or subtract factors from it. So in each line of code above, the fit model `m1` is passed to it first. Then a formula like `. ~ . + variable` is used to add one or more variables to the model. The periods on either side of the tilde mean "all." So read the formula `. ~ . + log(land_area)` as *Keep everything*

on the left side and the right side of the formula, and add `log(land_area)` to the right side. The `update` command doesn't work for every sort of fit model, but when it does, it can save you a lot of typing and confusion.

Go ahead and inspect the estimates from each model in turn. You'll see that the estimate for `rugged`'s slope is fairly constant across these models, and its standard error doesn't vary much either. But there's an easier way to compare a set of models here. What you really want is a table with models along one margin and variables along the other margin. Inside the table, there would be estimates. Then you could just scan across models and see how variable an estimate is as model structure changes. This is exactly the sort of table that is common in economics papers.

You could build such a table yourself, by extracting estimates and piecing them together into a spreadsheet or a matrix within R. But the LIBRARY of code that accompanies this book provides a convenient command for this purpose.

```
coeftab( m1 , m2 , m3 , m4 , m5 , m6 )
```

R code
5.30

	m1	m2	m3	m4	m5	m6
(Intercept)	9.2232	9.8418	9.5801	9.3879	9.2815	8.7500
rugged	-0.2029	-0.2082	-0.2605	-0.1927	-0.1915	-0.1674
log(land_area)	NA	-0.0678	NA	NA	NA	NA
soil	NA	NA	0.0003	NA	NA	NA
desert	NA	NA	-0.0121	NA	NA	NA
tropical	NA	NA	-0.0071	NA	NA	NA
dist_coast	NA	NA	NA	-0.6566	NA	NA
colony_gbr	NA	NA	NA	NA	-0.1276	NA
colony_fra	NA	NA	NA	NA	-0.5791	NA
colony_prt	NA	NA	NA	NA	0.1744	NA
cont_europe	NA	NA	NA	NA	NA	1.0150
cont_oceania	NA	NA	NA	NA	NA	0.4205
cont_north_america	NA	NA	NA	NA	NA	0.3946
cont_south_america	NA	NA	NA	NA	NA	0.0290
nobs	121.0000	121.0000	121.0000	121.0000	121.0000	121.0000

Each column in this table is one of the models you passed to `coeftab`. Each row is a variable that appeared in at least one model. In the cells of the table are the estimates for the parameters that correspond to each variable. When a model does not include a variable, the table displays a NA.

This kind of table makes it easier to compare the estimates across models. Looking at the second row in the table above, notice that while

the magnitude of the estimate corresponding to `rugged` does vary between -0.17 and -0.26 , it never gets knocked out completely. You can compare standard errors, as well, using an optional parameter:

R code
5.31

```
coeftab( m1 , m2 , m3 , m4 , m5 , m6 , se=TRUE )
```

After you execute the line above, you'll see another table, this time with standard errors appended to the bottom. Notice that the standard errors for `rugged` are pretty insensitive to which variables are included.

At the very bottom of the table, the number of rows used in each model is shown. These are just the numbers you get if you type `nobs(m1)`, for example. This row is useful, because sometimes including new variables will drop rows from the analysis. You really do want to be aware of when this happens. There are two common problems that arise from dropping incomplete cases. First, if you want to use AICc or another metric to compare the models, then they must all predict the exact same outcomes. So you have to drop all of the incomplete cases before you fit the models, even though some of the models don't include variables with any missing values. Second, if some variable is missing values for half of the rows, then when you include it in a model, the accuracy of all estimates is going to decrease, just because you've lost so much data. This may create the illusion that an estimate or its standard error have meaningfully changed. But what has actually happened is that you unwittingly tossed away half of your cases. If you ever want to turn off these sample sizes, just add the optional `nobs=FALSE` when you invoke `coeftab`.

It is possible to use all of the cases, despite missing values, but you'd have to use some kind of *imputation* (Chapter 12). In fact, the Bayesian approach has key advantages in that area, since it's possible to simultaneously sample from the posterior of a missing value and sample from the posterior of the parameters of the model needed to do the imputation. But we'll leave that for much later in the book.

5.10. Translating Hypotheses into Model Families

Notes on methods of setting up sets of models to compare, as an aid in testing hypotheses.

5.11. Fitting All Possible Models

A strategy that spontaneously occurs to many people, after meeting AIC and BIC, is to fit all possible combinations of a set of predictor variables and then let model comparison sort it out. For example, in the

primate milk energy data, there are six predictor variables available to model `kcal.per.g`. Ignoring interactions for now (these are the topic of the next chapter), that implies $2^6 = 64$ different model families, one for each unique combination of variables, including the intercept-only model.

This approach is not normally a good idea. First, many of these possible combinations will run afoul of multicollinearity, and they aren't worth fitting in the first place. Second, if we try enough combinations and transformations, we might eventually find a model that fits the data well. But this fit will be a peculiar case of over-fitting, unlikely to generalize to new data. Consider by analogy the *Curse of Tippecanoe*. From 1840 until 1960, every United States president who was elected in a year ending in the digit 0 (which happens every 20 years, given 4 year terms) has died in office. William Henry Harrison was the first, being elected in 1840 and dying of pneumonia the next year. John F. Kennedy was the last, being elected in 1960 and being assassinated in 1963. Seven American presidents died in sequence in this pattern. Ronald Reagan was elected in 1980, but managed to live long after his term was up, breaking the "curse." Given enough time and data, a pattern like this can be found for almost any body of data. But without any compelling reason to believe this pattern is meaningful, it is hardly compelling that such patterns exist. Most large sets of data will contain patterns of correlation that are strong and surprising. If we search hard enough, we are bound to find a Curse of Tippecanoe.⁸² There are many other patterns in presidential names and dates, and no doubt new ones are being found and circulated all the time. Data dredging is a great way to find coincidences, but not necessarily a great way to test hypotheses.

However, fitting all possible models isn't always a dumb idea, provided some judgment is exercised in weeding down the list of variables at the start. There are two scenarios in which this strategy seems defensible. First, sometimes all one wants to do is explore a set of data, because there are no clear hypotheses to evaluate. This is rightly labeled pejoratively as *data dredging*, when one does not admit to it.

5.11.0.1. *Data exploration.*

5.11.0.2. *Model averaging.* Defending the claim that a variable of interest does not help, regardless of covariates.

5.12. Common Questions

?? Deal with routine questions like: (1) Can use AIC with non-nested models?; (2) Can use AIC with models with different outcome distributions?; (3) Can use AIC to chose the right P-value?

6 Interactions

This chapter focuses on the specification and interpretation of *interaction effects*, a relationship between predictor variables in which the effect of each variable on the outcome depends upon the values of the other predictor variables. Interactions allow for a much wider range of hypotheses to be modeled. However, interpretation of such effects is usually much harder than is the interpretation of simple main effects. The non-linear, non-Gaussian models in later chapters automatically induce interactions among predictor variables. Therefore an understanding of interactions is necessary before moving on to generalized linear models.

6.1. Double Jeopardy

A frustrating fact about human life is the persistence of discrimination based upon social categories, much of it unconscious. A substantial empirical literature documenting and quantifying this discrimination, much of it race-based, has accumulated over the decades.⁸³ What is rarely asked, however, is what happens when an individual is a member of more than one discriminated category. For example, if both women and blacks are discriminated against, how much worse off is a black woman? Is there double jeopardy, in which the intensities of discrimination add up for such combinations?

In the late 1980's and early 1990's, Ian Ayres and Peter Siegelman coordinated 306 bargaining tests of discrimination at 153 different new car dealerships in the greater Chicago area.⁸⁴ These tests shed light on the double jeopardy question. Potential buyers were actually trained testers with fixed bargaining scripts. Some testers were white, while others were black. Some were female, while others were male. All of the testers were approximately matched for attractiveness, age, and education. They all wore similar "yuppie" attire, and all of them started bargaining by announcing that they would not finance the car. All of them claimed to be employed as a "systems analyst" at a large Chicago

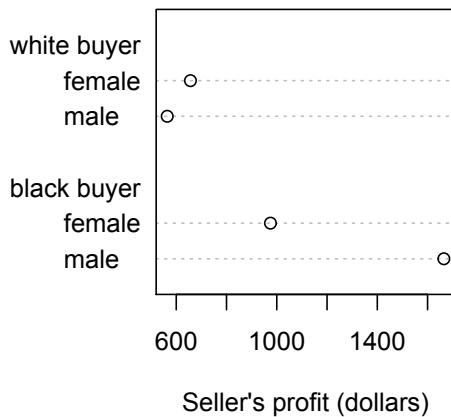


FIGURE 6.1. Average final profits of new car dealers, bargaining with different categories of trained testers. While both females and blacks received worse deals than did white men, the effect of gender on the intensity of discrimination depends hugely upon race. This is an interaction effect. Data from Ayres & Siegelman 1995.

area bank, and they gave their address as a wealthy Chicago neighborhood. All of this was designed to reduce confounds other than race and gender in explaining any differences in the prices dealers quoted to the testers.

The data from this study are actually quite detailed and can be analyzed several different ways. For now, it'll be sufficient to merely consider the final profit for the dealer. This is the amount the dealer would have retained after his or her own costs. So large amounts will indicate a worse deal for the tester. Figure 6.1 shows the average final profits, broken down by four combinations of race and gender. Considering initial dealer offers gives the same impressions. By far the largest effect is that black testers received worse deals, despite all testers' using identical bargaining scripts. If you could only have one explanatory variable here, it should be race. The effect of gender is much more subtle. It is true, as has been seen in other audit studies of discrimination, that white females get worse deals than do white men. The effect is much smaller in absolute magnitude than the effect of being black, but it is appreciable. However, black women actually get much better final deals than do black men, about \$500 better.

The lack of double jeopardy in these data are an example of an *interaction effect*. Interaction effects arise whenever there is a non-additive impact two (or more) variables on an outcome. *Non-additive* is a funny way to talk about this effect, so it'll help if I just show you what *additive* would look like. Figure 6.2 plots the imaginary means for an additive relationship between race and gender. In these imaginary data, both

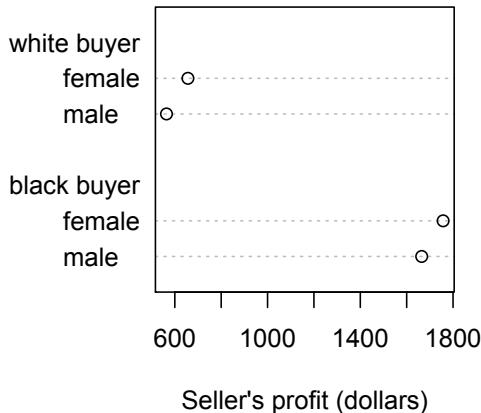


FIGURE 6.2. What the car price discrimination data would look like, if the effects of race and gender were purely additive. Compare to Figure 6.1.

being female and black increase price discrimination by a fixed average amount. In the real data, in Figure 6.1, the relationship is not additive in this way. Instead, in order to understand the effect of the variable *female*, you have to know the individual's value for the variable *race*.

Whenever the impact of one variable depends in this way upon the value of another variable, statisticians tend to call it an interaction effect. This sort of relationship, in which the direction or strength of the impact of one variable changes in response to yet another variable, is common in nature. My favorite example, which has been used in countless introductory statistics courses, is of adding sugar to your coffee. Just adding sugar does little to increase the sweetness of coffee, because the lump of sugar tends to fall to the bottom of the mug and lie there, only slowly dissolving. Likewise, stirring your coffee, in the absence of sugar, doesn't do anything for its sweetness. But if you both add sugar and then stir your coffee, it becomes much sweeter. This is an interaction effect between the variables *adding sugar* and *stirring*, affecting the outcome *sweetness*.

It might seem odd to some readers that this book devotes an entire chapter to interactions. Most of the material in this book is considerably more advanced than the concept of interaction, which is usually met in an introductory course on regression. And typically very little space is given over to the definition and exploration of interactions. But teaching this material has convinced me that most PhD students and even many practicing scientists never had a proper education in these important causal relationships. Just knowing how to specify interactions in

a statistical model is hardly sufficient for being able to understand and interpret the predictions they imply.

I recently (in 2011) reviewed a paper in which a world-famous scientist claimed that an interaction was unimportant, because the size of the estimate was small (with a very tight confidence interval that did not include zero). This was claimed despite the fact that the products this estimate was to be multiplied by in the model were on the order of 10-thousand. If the scientist had just plotted the model's predictions, it would have been clear that the interaction was hugely important. If world-famous scientists can make fundamental errors of this kind, then anyone can. Interactions are hard, and I am not going to pretend otherwise.

Furthermore, the spread of generalized linear models, such as the non-Gaussian models in the next two chapters, have made interactions even more important. In such models, even when one does not explicitly define variables as interacting, they will always interact to some degree. You'll see why in the next chapter. Moreover, every variable can essentially interact with itself, as the impact of change in its value will depend upon its current value. Say goodbye to simple constant slopes of linear regression. Say hello to mercurial impacts that may depend upon the covariation of dozens of predictor variables.

The similarly important spread of mixed/hierarchical/multilevel models—models in which there are more than one stochastic node—induce similar effects. Once you allow slopes to vary within each cluster (person, genus, village, city, galaxy) of the data, interaction effects can become very complex. Now we're not just allowing the impact of a predictor variable to change depending upon some other variable, but we are also estimating aspects of the *distribution* of those changes. It can be very easy to fit models that allow for complex random interactions, but it can be considerably harder to understand these models.

Since generalized linear models and multilevel models are so important, it is essential to appreciate how interaction effects arise, how to interpret them, and how to plot their predictions. Indeed, in general, I don't think it's easy to understand interactions without plotting, unless the models are very simple. Trying to figure out what the model implies, from only a table of estimates, is much harder once you allow interactions. Computing has become very powerful, but the culture of statistics in the sciences has lagged behind the software. Students and scientists are fitting models that they do not understand and reporting parameter estimates that are largely uninterpretable to themselves and their audience.

And so I want to spend this chapter reviewing simple interaction effects, how to specify them, how to interpret them, and how to plot them. The chapter starts in the next section with a case of an interaction between a single categorical (dummy) variable and a single continuous variable. In this context, it is easy to appreciate the sort of hypothesis that an interaction allows for. Then the chapter moves on to show how manipulating the model of the mean itself can help us understand the behavior of an interaction, before moving on to more complex interactions between multiple continuous predictor variables as well as higher-order interactions among more than two variables. The chapter ends by returning to the topic of multicollinearity, which is a common issue that can arise when specifying interactions, especially ones that involve dummy variables. In every section of this chapter, the model predictions are visualized, averaging over uncertainty in parameters and model families.

My hope is that this chapter lays a solid foundation for interpreting generalized linear models and multilevel models in the later chapters.

6.2. Building an Interaction

At the end of the previous chapter, you encountered the statistical relationship between the ruggedness of a nation's terrain and its state of economic development, as measured by per capita GDP in the year 2000. Nations with rugged terrain were worse off economically, on average. In that analysis, I had you remove African nations from the data. The reason for omitting the African nations is that the relationship between ruggedness of terrain and economic development tends to be reversed in Africa.

First, load the relevant data table and remove cases without economic data:

```
library(rethinking)
data(rugged)
d <- rugged
dd <- d[ complete.cases(d$rgdppc_2000) , ]
```

R code
6.1

You already saw the negative relationship between ruggedness and GDP outside Africa. So before seeing how to use an interaction effect, the cheating way to see that the relationship is positive within Africa is to analyze only the Africa cases now. Pull them out into a new data frame, and fit the simple linear regression of economic development on terrain ruggedness, just like in the last chapter:

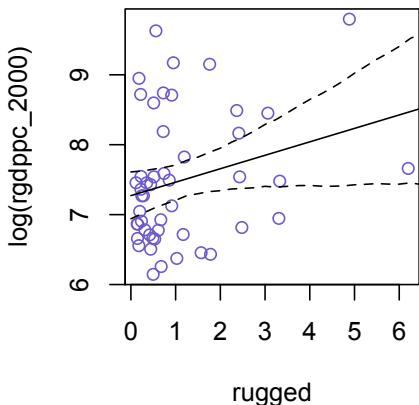


FIGURE 6.3. The relationship between economic development and terrain ruggedness, considering only African nations. Compare to FIGURE 5.10 (page 291).

R code
6.2

```
ddA <- dd[ dd$cont_africa==1 , ]
library(bbmle)
mA <- mle2( log(rgdppc_2000) ~ dnorm(
  mean=a + b.r*rugged , sd=1/tau ) , data=ddA ,
  start=list( a=mean(log(ddA$rgdppc_2000)) ,
  b.r=0 , tau=1/sd(log(ddA$rgdppc_2000)) ) )
precis(mA)
```

	Estimate	Std. Error	2.5%	97.5%
a	7.2751838	0.1696851	6.942607092	7.6077606
b.r	0.1905374	0.1007779	-0.006983668	0.3880586
tau	1.1194392	0.1130804	0.897805710	1.3410728

The slope is now positive, with a 95% confidence interval that barely includes zero. I plot this relationship in FIGURE 6.3. So it seems like the relationship between these two variables may really be reversed, for a subset of the cases.

One problem with analyzing this kind of reversal in the manner above is that you don't get any confidence information about the way you are splitting the data. Think of it this way. What I had you do just now was split the data into two tables: African nations and non-African nations. There are two major reasons that this is a bad idea, for anything except data exploration. First, there are usually some parameters, such as τ ($1/\sigma$) in this case, that the model says do not depend in any way upon an African identity for each case i . By splitting the data table, you are hurting the accuracy of the estimates for these parameters, because

you are essentially making two less-accurate estimates instead of pooling all of the evidence into one estimate. In effect, you have accidentally assumed that variance differs between African and non-African nations. Now, there's nothing wrong with that sort of hypothesis, and indeed in a later chapter (Chapter 10) this kind of hypothesis will be an important concern. But you want to avoid accidental assumptions.

Second, in order to acquire probability statements about the variable you used to split the data, `cont_africa` in this case, you need to include it in the model. Otherwise, you have only the weakest sort of statistical argument. Isn't there uncertainty about the predictive value of distinguishing between African and non-African nations? Of course there is. Unless you analyze all of the data in a single model, you can't easily quantify that uncertainty. If you just let the posterior density do the work for you, you'll have a useful measure of that uncertainty.

So let's see how to do it.

6.2.1. Adding a dummy variable doesn't work. The first thing to realize is that just including the categorical variable (dummy variable) `cont_africa` won't reveal the reversed slope. It's worth fitting this model to prove it to yourself, though. I'm going to walk through this as a simple model comparison exercise, just so you begin to get some applied examples of concepts you've accumulated from earlier chapters.

The question is to what extent singling out African nations changes predictions. There are two models to fit, to start. The first is just the simple linear regression of log-GDP on ruggedness, but now for the entire data set:

```
m1 <- mle2( log(rgdppc_2000) ~ dnorm(
  mean=a + b.r*rugged , sd=1/tau ) , data=dd ,
  start=list( a=mean(log(dd$rgdppc_2000)) , b.r=0 ,
  tau=1/sd(log(dd$rgdppc_2000)) ) )
```

R code
6.3

The second is the model that includes a dummy variable for African nations:

```
m2 <- mle2( log(rgdppc_2000) ~ dnorm(
  mean=a + a.A*cont_africa + b.r*rugged ,
  sd=1/tau ) , data=dd ,
  start=list( a=mean(log(dd$rgdppc_2000)) , a.A=0 , b.r=0 ,
  tau=1/sd(log(dd$rgdppc_2000)) ) )
```

R code
6.4

Now to compare these models, using AICc:

R code
6.5

```
AICctab( m1 , m2 , weights=TRUE , nobs=nrow(dd) )
```

```
dAICc df weight
m2  0    4   1
m1 64   3  <0.001
```

This is a case in which it will be safe to ignore the lower ranked model. Model averaging here will end up taking very few, if any, samples from `m1`. So it's safe to just go forward with `m2`, although harmless to use model averaging as well. To sample from the posterior and compute the predicted means and HPDI's for both African and non-African nations:

R code
6.6

```
post <- sample.naive.posterior( m2 )
rugged.seq <- seq(from=0,to=8,by=0.25)
mu.NotAfrica <- sapply( rugged.seq , function(z)
  mean( post$a + post$a.A*0 + post$b.r*z ) )
mu.Africa <- sapply( rugged.seq , function(z)
  mean( post$a + post$a.A*1 + post$b.r*z ) )
mu.ci.NotAfrica <- sapply( rugged.seq , function(z)
  HPDI( post$a + post$a.A*0 + post$b.r*z ) )
mu.ci.Africa <- sapply( rugged.seq , function(z)
  HPDI( post$a + post$a.A*1 + post$b.r*z ) )
```

I show these predictions in FIGURE 6.4. African nations are shown in blue, while nations outside Africa are shown in gray. What you've ended up with here is a rather weak negative relationship between economic development and ruggedness. The African nations do have lower overall economic development, and so the blue regression line is below, but parallel to, the black line. All including a dummy variable for African nations has done is allow the model to predict a lower mean for African nations. It can't do anything to the slope of the line. The fact that AICc tells you that the model with the dummy variable is hugely better only indicates that African nations on average do have lower GDP.

6.2.2. Adding a linear interaction does work. How can you recover the change in slope you saw at the start of this section? You need a proper interaction effect. The model you just plotted, in maths form, is:

$$\begin{aligned}y_i &\sim \text{Normal}(\mu_i, \tau), \\ \mu_i &= \alpha + \alpha_A A_i + \beta_r r_i,\end{aligned}$$

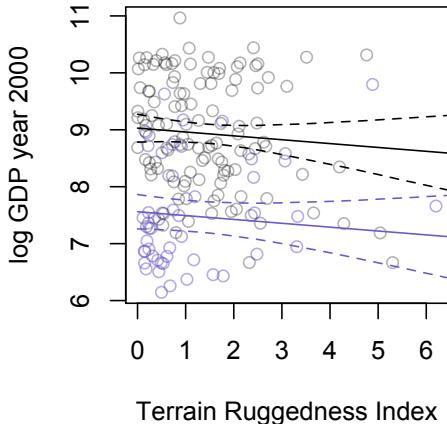


FIGURE 6.4. Including a dummy variable for African nations has no effect on the slope. African nations are shown in blue. Non-African nations are shown in gray. Regression lines for each subset of cases are shown in corresponding colors, along with 95% HPDI's.

where y is `log(rgdppc_2000)`, A is `cont_africa`, and r is `rugged`. As you've done since Chapter 2, the multivariate linear model is built by replacing the parameter μ in the top line, the stochastic node, with a linear equation that is a function of data and new parameters.

Now you want to allow the relationship, or slope β_r , between y and r to vary as a function of A . Following the same strategy of replacing parameters with linear models, the most straightforward way to do this is just to define the slope β_r as a linear model itself, one that includes A . This approach results in this model:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \tau), \\ \mu_i &= \alpha + \alpha_A A_i + \gamma_r r_i, \\ \gamma_r &= \beta_r + \beta_{Ar} A_i. \end{aligned}$$

This is the first model with three expressions, but its structure is the same as every Gaussian model you've already fit in this book. It might help to format the model definition this way:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \tau), \\ \mu_i &= \alpha + \alpha_A A_i + \gamma_r r_i, \\ \gamma_r &= \beta_r + \beta_{Ar} A_i. \end{aligned}$$

Hopefully you can see more easily now that each successive line just defines one element of the previous line.

You don't need to learn any new tricks for estimating this model. The tricks lie entirely in interpreting it. The first line above is the same

Gaussian stochastic node you've been using since Chapter 4. The second line is the same kind of additive definition of μ_i that you've seen many times.

The third line is the new bit. The new symbol γ_{ri} is just a placeholder for the linear function that defines the slope between GDP and ruggedness. We use “gamma” (γ) here, because it follows “beta” (β) in the Greek alphabet. The equation for γ_{ri} defines the interaction between ruggedness and African nations. It is a *linear interaction effect*, because the equation γ_{ri} is the equation for a line.

By defining the relationship between GDP and ruggedness in this way, you are explicitly modeling the hypothesis that the slope between GDP and ruggedness depends upon whether or not a nation is in Africa. The parameter β_{Ar} defines the strength of this dependency. If you set $\beta_{Ar} = 0$, then you get the previous model back. If instead $\beta_{Ar} > 0$, then African nations have a more positive slope between GDP and ruggedness. If $\beta_{Ar} < 0$, African nations have a more negative slope. For any nation not in Africa, $A_i = 0$ and so the interaction parameter β_{Ar} has no effect on prediction for that nation. Of course you are going to estimate β_{Ar} from the data. But once you have the estimate, it is only understanding where the parameter fits into your model that will allow you interpret the estimate.

So how do you fit this new model? You can just use `mle2` as before. I'll show you how to do this with the simpler interface of `lm` later. But it's very instructive right now to avoid convenient functions like `lm`, because those functions hide the parameters from you. As always, you really want to know what the exact model is. Otherwise, interpreting and plotting the fit model will range from clumsy to impossible.

To use `mle2`, first you'll need to fold γ_{ri} into the equation for μ_i . This just means replace γ_{ri} with the expression for it, inside μ_i :

$$\begin{aligned}\mu_i &= \alpha + \alpha_A A_i + \gamma_{ri} r_i, \\ &= \alpha + \alpha_A A_i + \underbrace{(\beta_r + \beta_{Ar} A_i)}_{\gamma_{ri}} r_i.\end{aligned}$$

And you can enter that into the `mean` parameter of `dnorm` as well as any other equation.

So let's actually do it. Here's the code to fit the model that includes an interaction between ruggedness and being in Africa:

R code
6.7

```
m3 <- mle2( log(rgdppc_2000) ~ dnorm(
  mean=a + a.A*cont_africa + (b.r + b.Ar*cont_africa)*rugged ,
  sd=1/tau ) , data=dd ,
```

```
start=list( a=mean(log(dd$rgdppc_2000)) , a.A=0 , b.r=0 ,
b.Ar=0 , tau=1/sd(log(dd$rgdppc_2000)) ) )
```

Before moving on to interpret the estimates and plotting the predictions, let's use AICc to compare this new model to the previous two:

```
AICctab( m1 , m2 , m3 , weights=TRUE , nobs=nrow(dd) )
```

R code
6.8

	dAICc	df	weight
m3	0.0	5	0.9675
m2	6.8	4	0.0325
m1	70.8	3	<0.001

Model family m3 has about 97% of the AICc-estimated posterior probability. That's very strong support for including the interaction effect.

6.2.3. Interpreting an interaction estimate. Interpreting interaction estimates is tricky. The parameter estimate for β_{Ar} (`b.Ar`) should tell us whether or not African nations have higher or lower GDP with increasing ruggedness. And it does. But what you really want to know is how changing the value of either predictor variable implicated in the interaction will change the outcome. And that's not always easy. In this subsection, I'll try to provide a sense for interpreting interaction estimates and combining them with main effect estimates. In the next section, I'll introduce the important concept of *centering*, which will make this job a little easier. But in general, plotting the change in outcome as a function of changes in predictors is the most transparent and effective way to understand what the model estimates mean. And so after chatting about estimates themselves, I'll turn to showing you how to use them to produce implied predictions of the model.

Here are the estimates:

```
precis( m3 )
```

R code
6.9

	Estimate	Std. Error	2.5%	97.5%
a	9.2232344	0.13799694	8.9527654	9.4937034
a.A	-1.9480705	0.22456475	-2.3882093	-1.5079317
b.r	-0.2028494	0.07647626	-0.3527401	-0.0529587
b.Ar	0.3933755	0.13007433	0.1384345	0.6483165
tau	1.0721959	0.05814753	0.9582288	1.1861629

First, consider the so-called *main effects*, `a.A` (α_A) and `b.r` (β_r). The estimate for `a.A` is about -1.9 . This is the estimated *difference* between

the intercept for African nations and that for non-African nations. So the model predicts an intercept for African nations of about $9.22 - 1.9 = 7.32$. African nations have lower average GDP than nations outside Africa. The quadratic estimate confidence interval for α_A suggests that the estimate is very reliably on the negative side of zero.

The estimate for b_r is about -0.2 . The 95% confidence interval doesn't include zero, but it does get fairly close to it. This estimate says that the slope between log-GDP and ruggedness *outside of Africa* is negative. More rugged nations do worse, by about 0.2 log-GDP per unit of ruggedness, on average.

Now we turn to the interaction estimate. The maximum likelihood estimate of b_{Ar} is about 0.39. As with the other two estimates, the quadratic approximation to the naive posterior tells you that the parameter is reliably on one side of zero. This is a positive interaction effect. What is the predicted slope between log-GDP and ruggedness? Just use the equation for γ_{ri} to find out:

$$\gamma_{ri} = \beta_r + \beta_{Ar} A_i \approx -0.2 + 0.39 A_i.$$

Therefore an African nation (with $A_i = 1$) has an average predicted slope of $-0.2 + 0.39 \times 1 = 0.19$. Any non-African nation (with $A_i = 0$) still has a predicted slope of $-0.2 + 0.39 \times 0 = -0.2$. The interaction estimate says that African nations have almost exactly the opposite average slope as non-African nations.

Combining what you've learned about the intercept and slope for African nations, the predicted regression mean for Africa is about $7.32 + 0.19 r_i$, whereas it is $9.22 - 0.2 r_i$ for non-African nations. But those estimates are just the maximum likelihood estimates. We'd also like a way to quantify uncertainty around these estimated slopes. But since the slope for African nations depends upon two estimates, β_r and β_{Ar} , and these two estimates are correlated in some way, you can't just use the posterior of any single parameter. This is a typical issue with interpreting interactions: the actual effect of interest, the change in the outcome as a result of changing a predictor variable, depends upon more than one estimate. As a result, it isn't simply a matter of reading a single parameter estimate. This has always been true of predicting regression trends, to some extent, because of the correlation between intercept parameters, α , and slopes, β . But now with interactions the obstacle is correlations among intercepts and two slopes that multiply the same predictor.

Shortly, I'll illustrate how to plot the estimated interaction and its confidence interval, against the data. But if you can plot it, then it's possible to quantify the uncertainty in the total slope against rugged, γ_{ri} ,

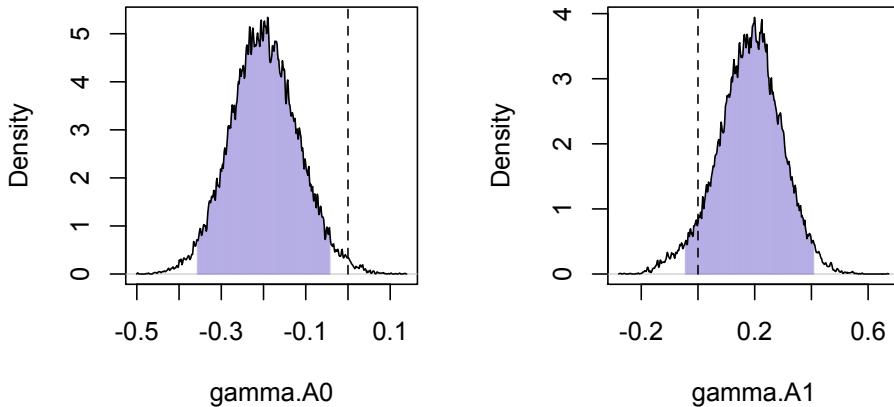


FIGURE 6.5. The naive posterior densities for γ_{ri} outside Africa (gamma.A0, left) and inside Africa (gamma.A1, right). The slope between GDP and ruggedness depends upon two parameter estimates, but these can be combined, using samples from the posterior, to produce a measure of uncertainty in the total slope. The blue shaded regions are the 95% HPDI's. The vertical dashed line shows the location of zero.

as well. Here's the recipe, which will be familiar to you. First, sample from the naive posterior of the fit models. We'll do this, averaging with AICc. Second, use the samples from the posterior to compute the posterior density of γ_{ri} , both for African and non-African nations. Finally, compute intervals from the posterior or just plot them. Here are these steps, in code form:

```
post <- sample.naive.posterior( list(m1,m2,m3) , n=30000 ,
  nobs=nrow(dd) )
gamma.A0 <- post$b.r + post$b.Ar*0
gamma.A1 <- post$b.r + post$b.Ar*1
dens( data.frame( gamma.A0 , gamma.A1 ) )
```

R code
6.10

The resulting density plots are shown in FIGURE 6.5. The left plot is the posterior density of the slope between GDP and ruggedness, γ_{ri} , outside of Africa. This posterior is the same as that of β_r ($post$b.r$) alone,

because the interaction coefficient cancels outside of Africa. On the right is the posterior density for γ_{ri} inside of Africa. Now the interaction estimate matters. This density is a bit wider than that on the left, because it is built from two parameter estimates. This leads the HPDI (blue shaded region) of γ_{ri} within Africa to overlap zero, even though the HPDI of neither parameter comprising it, β_r nor β_{Ar} , does.

Careful not to conclude that this is evidence against an interaction! AICc already told you that there is overwhelming reason to prefer the model that includes it. As I've warned before in Chapter 5, if you accept the logic of comparing model families with AICc (or BIC), then you shouldn't accept the logic of significance tests, when choosing the values of parameters. Something like a significance test, applied to the actual predictive distribution of the model family (not just the null model!), might be useful for rejecting the model entirely, once you've estimated its parameters. But choosing parameter values with NHST both performs worse and is logically incompatible with using posterior probabilities and AICc/BIC. If you feel your resolve against using significance tests wavering, then please go back and read Chapter 3 again.

This kind of calculation—combining multiple estimates into a single density of actual importance—is a pain, analytically, because you'd have to properly account for the correlation between the two estimates, when computing the density of their sum. It isn't rocket surgery, but it's certainly a bit beyond the skills of most scientists. However, the samples from the posterior make it trivial, provided you actually know the equation for the model of the mean, μ_i . Again, sampling from the posterior transforms a problem in mathematics into a problem in summarizing data.

6.2.4. Plotting the interaction. Interaction effects are notoriously difficult to interpret, from parameter estimates alone. There are some techniques, like *centering*, that can help. I'll discuss centering in a bit. But even then, most scientists have a hard time intuiting the magnitude of the effect, from parameter estimates alone. The primary obstacle, as you've just seen above, is that the rate of change in the outcome, within an interaction model, depends upon more than one parameter. For this reason, using the full posterior to compute uncertainty in the combined slope (as above) and plotting the implied predictions of the model are almost always a superior way to understand the model yourself and to communicate its implications to an audience. Now we'll focus on plotting the interaction, to make sure you really understand the model and how to extract predictions from it.

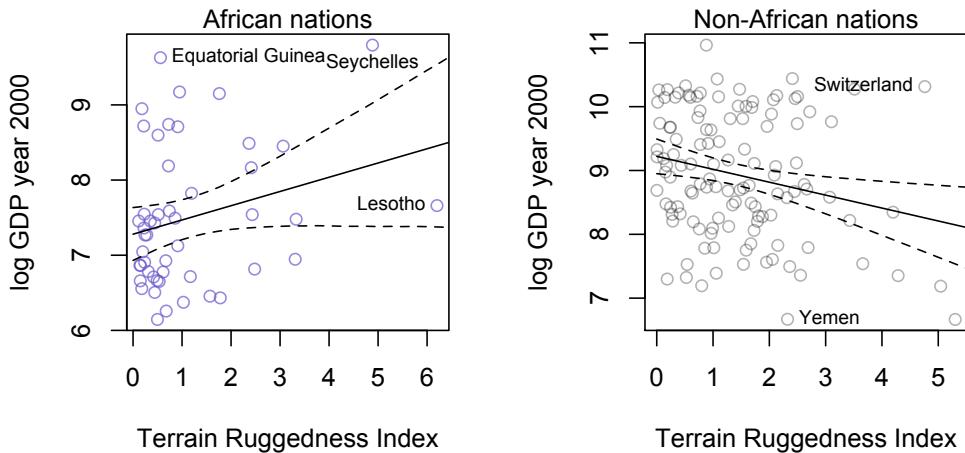


FIGURE 6.6. Including an interaction between African nations and ruggedness changes the slope. On the left, for African nations only, the predicted slope is positive. On the right, for non-African nations, the predicted slope is negative.

Just like above, we want to incorporate model family uncertainty into the estimates, via model averaging. So let's plot the model averaged naive posterior predictions for model families m_1 , m_2 , and m_3 . You have already seen the tools needed to do this. But inspecting the code here will help cement in your mind how the interaction estimate enters into the model.

```
post <- sample.naive.posterior( list(m1,m2,m3) , n=30000 ,
  method="AICc" , nobs=nrow(dd) )
rugged.seq <- seq(from=0,to=8,by=0.25)
mu.NotAfrica <- sapply( rugged.seq , function(z)
  mean( post$a + post$a.A*0 + (post$b.r + post$b.Ar*0)*z ) )
mu.Africa <- sapply( rugged.seq , function(z)
  mean( post$a + post$a.A*1 + (post$b.r + post$b.Ar*1)*z ) )
mu.ci.NotAfrica <- sapply( rugged.seq , function(z)
  HPDI( post$a + post$a.A*0 + (post$b.r + post$b.Ar*0)*z ) )
mu.ci.Africa <- sapply( rugged.seq , function(z)
  HPDI( post$a + post$a.A*1 + (post$b.r + post$b.Ar*1)*z ) )
```

R code
6.11

FIGURE 6.6 displays these calculations, in two separate plots. In each plot, the line shows the mean relationship between log-GDP and ruggedness, while the dashed boundaries show the 95% HPDI for the relationship. On the left, the estimated relationship between log-GDP and ruggedness is shown for only African nations. The slope is positive. On the right, the same relationship but now for only those nations outside of Africa. The slope here is negative. This change in the direction of the slope is a result of adding the interaction effect. The slope depends upon whether or not the nation is in Africa. You might compare the slopes of the dashed confidence intervals in FIGURE 6.6 to the boundaries of the blue shaded HPDI in FIGURE 6.5. They do correspond, because they come from the same calculations.

6.3. Anatomy of the Linear Interaction

What you've seen above is a very common and useful way to incorporate a hypothesis like *the influence of ruggedness on log-GDP depends upon whether or not a nation is in Africa*. You used a linear interaction in which you replaced the simple regression slope parameter β_r with a linear model γ_n that includes a dummy variable for being in Africa. This allowed the predicted slope to depend upon continent. After fitting the interaction model to the data, you discovered that the relationship does strongly depend upon continent.

A natural question at this point might be: Isn't it also possible that the different intercept for African nations, $a \cdot A (\alpha_A)$, depends upon ruggedness? There's nothing logically wrong with this hypothesis. If the influence of ruggedness on GDP can depend upon continent, than why can't the influence of continent depend upon ruggedness?

The snag here is that as long as the hypothetical interactions are modeled as linear ones, using the approach introduced in the previous section, then it isn't possible to tell the difference between:

- (1) The influence of ruggedness depends upon continent, and
- (2) The influence of continent depends upon ruggedness.

As a result, whenever you model one of these using a linear interaction, you simultaneously model the other. So while these two hypotheses might seem causally distinct when expressed in words, within the additive model they are the same.

Linear interactions are incredibly useful modeling tools. But in order to interpret them correctly, you have to appreciate their inherent ambiguity. In this section, I'm going to try to expose their ambiguity, or rather bidirectionality, by showing you how to manipulate the equation for the mean μ_i . The mathematics here are very basic, nothing more

than simple factoring. One reason to do this is to reveal to you why functions like `lm` force you to specify interactions as products of predictor variables. Another important reason is that, as you get used to manipulating these models, you gain confidence in building and interpreting them. So the simple algebraic manipulations in this section will hopefully plant a small seed in your brain that will eventually grow into the insight that all statistical models can be explored in similar ways.

I can appreciate that most scientists did not get into science to build models, and most scientists do not think of themselves as “modelers.” But if you want to do statistics in the 21st century, you’re going to have to do a little bit of modeling, even if all that means is translating your verbal hypotheses into regression models.

6.3.1. Linear interactions are bidirectional. The first lesson is that whenever you include a linear interaction for a predictor variable, you simultaneously include a linear interaction for the other predictor variable. Recall the linear regression model that includes a linear interaction, whereby the influence of ruggedness on log-GDP depends upon continent:

$$\begin{aligned}y_i &\sim \text{Normal}(\mu_i, \tau), \\ \mu_i &= \alpha + \alpha_A A_i + \gamma_{ri} r_i, \\ \gamma_{ri} &= \beta_r + \beta_{Ar} A_i.\end{aligned}$$

This clearly reads as if it is the influence of r_i that depends upon A_i , not the other way around. But you’re going to show that the opposite interpretation is just as valid.

Begin by substituting in the equation for the interaction, γ_{ri} :

$$\mu_i = \alpha + \alpha_A A_i + (\beta_r + \beta_{Ar} A_i) r_i.$$

Now factor to group together all of the terms multiplied by A_i :

$$\mu_i = \alpha + (\alpha_A + \beta_{Ar} r_i) A_i + \beta_r r_i.$$

Now it looks like it is the influence of the dummy variable A_i that depends upon ruggedness r_i , rather than the other way around. The parameter β_{Ar} is equally interpretable as either direction of dependency.

This fact leads most analysts most of the time to just express linear interactions as separate terms in the equation for μ_i . Taking the equation above and expanding it:

$$\mu_i = \alpha + \alpha_A A_i + \beta_r r_i + \beta_{Ar} r_i A_i.$$

This is the same equation, but now with the interaction parameter in a term at the end. The product of the two predictor variables A_i and r_i

creates the linear interaction. But its leading coefficient, β_{Ar} , is equally interpretable as the degree to which (1) the influence of r_i depends upon A_i or (2) the influence of A_i depends upon r_i .

6.3.2. Linear interactions are doubly bidirectional. You arrive at the same destination, if you set out to explicitly include *both* interactions as linear models. Suppose you take the parameter α_A from earlier and make it too a linear model:

$$\begin{aligned}y_i &\sim \text{Normal}(\mu_i, \tau), \\ \mu_i &= \alpha + \gamma_{Ai}A_i + \gamma_{ri}r_i, \\ \gamma_{ri} &= \beta_r + \beta_{Ar}A_i, \\ \gamma_{Ai} &= \alpha_A + \beta_{rA}r_i.\end{aligned}$$

Now you have a mean μ_i that depends upon two embedded linear interactions, one for each of the predictor variables. The parameter β_{Ar} might be thought of as measuring A 's influence on the slope of r , while β_{rA} might be thought of as r 's influence on the intercept A .

Substituting both of these “gamma” linear models into μ_i and simplifying:

$$\begin{aligned}\mu_i &= \alpha + \gamma_{Ai}A_i + \gamma_{ri}r_i, \\ &= \alpha + (\alpha_A + \beta_{rA}r_i)A_i + (\beta_r + \beta_{Ar}A_i)r_i, \\ &= \alpha + \alpha_AA_i + \beta_rr_i + (\beta_{rA} + \beta_{Ar})A_ir_i.\end{aligned}$$

There's that product of the predictor variables again, at the end. But now both interaction parameters β_{rA} and β_{Ar} are multiplied by the same simple product A_ir_i . As a result, it is their sum that is estimated. This is exactly the sort of problem that causes *unidentifiable* parameters. If the maximum likelihood estimate of the sum $\beta_{rA} + \beta_{Ar}$ is called β_{A+r} , then there are an infinite number of combinations of two values β_{rA} and β_{Ar} that will result in the same value of β_{A+r} . Since you can't estimate β_{rA} and β_{Ar} separately, you might as well replace their sum with a single parameter that you can estimate:

$$\mu_i = \alpha + \alpha_AA_i + \beta_rr_i + \beta_{A+r}A_ir_i.$$

And again we arrive at an equation for μ_i with an interaction term containing the product of the two predictor variables.

6.3.3. Specifying linear interactions as products. Most books actually teach interaction effects as if they were simply this product of the predictor variables. The virtue of starting with the product specification is that it automatically reveals the bidirectionality of interpretation. The

curse of starting with the product specification is that it conceals the linear model nature of the interaction. These simple linear interactions are built by embedding linear models within linear models. They are a natural extension of the gambit that gets you to writing a model for μ that depends upon predictor variables. You can replace any parameter of a stochastic node with a model that depends upon data and parameters. Likewise, you can replace any parameter of a model with another model, also depending upon data and parameters. So far, you've just embedded deterministic expressions inside other deterministic expressions. In later chapters, a similar strategy gets us conceptually to multilevel regression models, which actually embed stochastic nodes within other stochastic nodes. At that point you'll encounter interactions again, in the form of *random effects*.

But that can wait. For now, you'd like to know how to specify interaction effects in R's built-in functions like `lm`. Here's how to fit the log-GDP model with `lm`:

```
m3lm <- lm( log(rgdppc_2000) ~ cont_africa * rugged , data=dd )
precis( m3lm )
```

R code
6.12

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	9.2232264	0.13965252	8.9495125	9.49694027
cont_africa	-1.9480480	0.22725775	-2.3934650	-1.50263100
rugged	-0.2028571	0.07739404	-0.3545466	-0.05116756
cont_africa:rugged	0.3933938	0.13163438	0.1353952	0.65139245

The row in the estimates table beginning with `cont_africa:rugged` is the interaction coefficient, β_{Ar} . Compare these estimates to those from `m3`, the model you fit with `mle2`. What `lm` does when it encounters input like `cont_africa*rugged` is automatically include main effects terms for both variables, as well as an interaction term. As a result, you could fit the exact same model with:

```
m3lm <- lm( log(rgdppc_2000) ~ cont_africa + rugged
+ cont_africa * rugged , data=dd )
```

R code
6.13

6.4. Continuous Interactions and the Value of Centering

The main point I want to convince the reader of is that interaction effects are difficult to interpret, using only coefficients. One reason for this difficulty is that interaction models make the change in the outcome depend upon more than one parameter. Another reason is that

the magnitude of the interaction parameter bears an uncertain relationship to the magnitude of the change in prediction. Very small interaction parameter estimates can imply huge changes in outcomes, because the “data” they are multiplied by is potentially a very large number, because it is the product of two potentially large numbers. This is just as higher-order terms in a polynomial regression can be important, even if the corresponding coefficient is very small, just because the coefficient is multiplied by something like x^4 , which can be a very large number even when x is not particularly large itself. For example, if $x = 10$, then $x^4 = 10000$. Even a tiny coefficient, once multiplied by 10-thousand, can exert strong influence on the trend.

A third reason to be wary of using only tables of coefficients to interpret interactions is that interactions among continuous variables are especially opaque. It’s one thing to make a slope depend upon a *category*, as in the previous example of ruggedness and being in Africa. In such a context, the model reduces to estimating a different slope for each category. But it’s quite a lot harder to understand that a slope varies linearly with a continuous value, even though the mathematics of the model are essentially the same as in the categorical case.

In pursuit of clarifying the construction and interpretation of *continuous* interactions among two or more continuous predictor variables, in this section I develop a simple regression example and show you a way to plot the two-way interaction between two continuous variables. The method I present for plotting this interaction is a *triptych* plot, a panel of three complementary figures that comprise a whole picture of the regression results. There’s nothing magic about having three figures—in other cases you might want more or less. Instead, the utility lies in making multiple figures that allow one to see how the interaction alters a slope, across changes in a chosen variable.

This data example will also allow me to illustrate two benefits of *centering* prediction variables. When a prediction variable is “centered,” it is just rescaled so that its mean is zero. Why would you ever do that to the data? There are two common benefits. First, centering the prediction variables can make it much easier to lean on the coefficients alone in understanding the model, especially when you want to compare the estimates from models with and without an interaction. Second, sometimes maximum likelihood search has a hard time with uncentered variables. Centering the data before fitting the model can help you achieve a faster and more reliable set of estimates.

There is another important benefit of centering, since it can be used to reduce the risk of collinearity. But I'll have to illustrate that benefit with another example, later in this chapter.

6.4.1. The data. The data in this example are sizes of blooms from beds of tulips grown in greenhouses, under different soil and light conditions. Load the data with:

```
library(rethinking)
data(tulips)
d <- tulips
str(d)
```

R code
6.14

```
'data.frame': 27 obs. of 4 variables:
 $ bed   : Factor w/ 3 levels "a","b","c": 1 1 1 1 1 1 1 1 1 2 ...
 $ water : int  1 1 1 2 2 2 3 3 3 1 ...
 $ shade : int  1 2 3 1 2 3 1 2 3 1 ...
 $ blooms: num  0 0 111 183.5 59.2 ...
```

The `blooms` column will be our outcome, what we wish to predict. We'll treat it as Gaussian, although as you'll see in the next chapter, there are reasons to prefer other densities. The `water` and `shade` columns will be our predictor variables. `water` indicates one of three ordered levels of soil moisture, from low (1) to high (3). `shade` indicates one of three ordered levels of light exposure, from high (1) to low (3). The last column, `bed`, indicates a cluster of plants from the same section of the greenhouse.

Since both light and water help plants grow and produce blooms, it stands to reason that the independent effect of each will be to produce bigger blooms. But we'll also be interested in the interaction between these two variables. In the absence of light, for example, it's hard to see how water will help a plant—photosynthesis depends upon both light and water. Likewise, in the absence of water, sunlight does a plant little good. One way to model such an interdependency is to use an interaction effect. In the absence of good mechanistic model of the interaction, one that uses a theory about the plant's physiology to hypothesize the functional relationship between light and water, then a simple linear two-way interaction is a good start.

6.4.2. The un-centered models. While a complete model averaging analysis is possible here, I'm going simplify the story by focusing on just two models: (1) the model with both `water` and `shade` but no interaction and (2) the model with both main effects and the interaction of `water` with `shade`. I do so just for the sake of brevity. You can fit

the missing models, like those with only one of the two predictor variables, and demonstrate for yourself that the conclusions don't substantially change.

The main effect model is:

$$\mathcal{M}_1 : B_i \sim \text{Normal}(\mu_i, \sigma), \\ \mu_i = \alpha + \beta_w w_i + \beta_s s_i.$$

And the full interaction model is:

$$\mathcal{M}_2 : B_i \sim \text{Normal}(\mu_i, \sigma), \\ \mu_i = \alpha + \beta_w w_i + \beta_s s_i + \beta_{ws} w_i s_i,$$

where B_i is the value of `bloom` on row i , w_i is the value of `water`, and s_i is the value of `shade`. I'm leaving the categorical variable `bed` out of this analysis, but I actually think a sincere analysis requires it. The points I wish to make don't depend upon it, however.

I'll fit the above models with plain `lm` right now. Later, I'll show you how `mle2` treats these data.

R code
6.15

```
m1 <- lm( blooms ~ water + shade , data=d )
m2 <- lm( blooms ~ water * shade , data=d )
coeftab(m1,m2)
```

	m1	m2
(Intercept)	60.5959	-150.8107
water	75.8017	181.5050
shade	-41.6028	64.1006
water:shade	NA	-52.8517
nobs	27.0000	27.0000

Now consider these estimates and try to figure out what the models are telling us about the influence of water and shade on the blooms. First, consider the intercepts, labeled `(Intercept)` in the output and corresponding to the parameter α in the models. The estimate of the intercept changes a lot from one model to the next. What do these values mean? Remember, the intercept is the expected value of the outcome when *all* of the predictor variables take the value zero. In this case, neither of the predictor variables ever takes the value zero within the data. As a result, these intercept estimates are very hard to interpret. This is a very common issue, and most of the regression examples so far in this book have suffered from it.

Now, consider the slope parameters. In the main-effect-only model, `m1`, the MLE for the main effect of `water` is positive and the main effect for `shade` is negative. Take a look at the standard errors and confidence

intervals in `precis(m1)` to verify that both estimates are reliably on one side of zero. You might infer that these estimates suggest that water increases blooms while shade reduces them. For every additional level of soil moisture, blooms increase by 76, on average. For every addition unit of shade, blooms decrease by 42, on average. Those sound reasonable.

But the analogous estimates from the interaction model, `m2`, are quite different. First, assure yourself that the interaction model is indeed a much better model:

```
compare(m1,m2,nobs=27)
```

R code
6.16

	k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC
<code>m2</code>	5	295.3212	298.9432	0.992476	0.991453	0.000000	0.000000
<code>m1</code>	4	305.0853	308.4505	0.007524	0.008547	9.764124	9.507248

This is a slam dunk for `m2`, whether you prefer AICc or BIC. So let's consider the estimates from `m2`. Now both main effects are positive, but the new interaction estimate is negative. Are you to conclude now that the main effect of shade is to help the tulips? And the negative interaction itself implies that as shade increases, water has a reduced impact on blooms. But reduced by how much?

Sampling from the posterior now and plotting the model's predictions would help immensely with interpretation. And that's the course I want to encourage and provide code for. In general, I don't think it's safe to interpret interactions without plotting them. But for the moment, let's instead look at the value of *centering* the predictor variables and re-estimating the models.

6.4.3. Center and re-estimate. To *center* a variable means to create a new variable that contains the same information as the original, but has a new mean of zero. For example, to make centered versions of `shade` and `water`, just subtract the mean of the original from each value:

```
d$shade.c <- d$shade - mean(d$shade)
d$water.c <- d$water - mean(d$water)
```

R code
6.17

The new centered variable has the same variance as the original, but now has a mean of zero.

Why do this? There a couple of common benefits of centered predictor variables. I'll illustrate the first, now. Let's re-estimate the two regression models, but now using the new centered variables `shade.c` and `water.c`:

R code
6.18

```
m1c <- lm( blooms ~ water.c + shade.c , data=d )
m2c <- lm( blooms ~ water.c * shade.c , data=d )
coeftab(m1c,m2c)
```

	m1c	m2c
(Intercept)	128.9937	128.9937
water.c	75.8017	75.8017
shade.c	-41.6028	-41.6028
water.c:shade.c	NA	-52.8517
nobs	27.0000	27.0000

Now when we compare the estimates across the two models, the main effects are the same. Unlike before, the direction of the estimate for `shade` has not changed. More water appears to directly increase blooms, while more shade directly decreases them. Meanwhile, the interaction estimate has remained the same as it was in the non-centered model.

Why did centering the predictor variables result in the main effect estimates remaining the same across the models with and without the interaction? In the un-centered models, the interaction effect is applied to every case. This is because neither of the predictors in those models, `shade` and `water`, are ever zero. As a result the interaction term always factors into generating a prediction. Consider for example a tulip at the average moisture and shade levels, 2 in each case. The expected blooms for such a tulip is:

$$\mu_i|_{s_i=2,w_i=2} = \alpha + \beta_w(2) + \beta_s(2) + \beta_{ws}(2 \times 2).$$

So to figure out the effect of increasing water by 1 unit, you have to use all of the β estimates. Plugging in the MLE's for the un-centered interaction model, `m2`, we get:

$$\mu_i|_{s_i=2,w_i=2} = -150.8 + 181.5(2) + 64.1(2) - 52.9 \times 2 \times 2.$$

You can compute the prediction in R:

R code
6.19

```
k <- coef(m2)
k[1] + k[2]*2 + k[3]*2 + k[4]*2*2
```

```
(Intercept)
128.9937
```

And by no coincidence at all, this is the same prediction we get from the centered interaction model, `m2c`, also using the mean values of both predictors:

```
k <- coef(m2c)
k[1] + k[2]*0 + k[3]*0 + k[4]*0*0
```

R code
6.20

```
(Intercept)
128.9937
```

In this case, however, you can arrive at this fact straight away, because the mean value of each predictor is zero, and so all that remains is the intercept:

$$\begin{aligned}\mu_i|_{s_i=0, w_i=0} &= \alpha + \beta_w(0) + \beta_s(0) + \beta_{ws}(0 \times 0), \\ &= \alpha.\end{aligned}$$

The intercept actually means something, when you center the predictors. It becomes the grand mean of the outcome variable, `mean(d$blooms)`. This ease of interpretation alone is a good reason to center predictor variables.

And now in the centered model, when both predictors are at the mean, and you increase one of them, the interaction effect does not apply. This forces the main effect for that variable to be the same across models with and without the interaction. In the un-centered model, the interaction always applied to each case, and so the estimates of the main effects had to adjust to account for that fact. The predictions of the model are the same, whether you center or not. But the centered model is much easier to interpret, if you want to lean on tables of coefficients.

As the reader knows already, I'm not keen on encouraging people to lean on tables of coefficients. So if you are going to follow my advice and always plot predictions and check posteriors against data anyway, do you need to center the predictors? Yes, there are still good reasons to do so. But before getting to the second benefit of centering, let's look at how to plot this continuous interaction, so you can appreciate how big of an effect it has on prediction.

6.4.4. The visualization. In previous chapters, there were no interactions. As a result, when plotting model predictions for any one predictor, you could hold the other predictors constant at any value you like. I encouraged the use of the mean or median, but all other values would do is shift the regression trend up or down the vertical axis. So the choice of which values to set the un-viewed predictor variables to wasn't sensitive.

Now that'll be different. Once there are interactions in a model, the effect of changing a predictor depends upon the values of the other predictors. Maybe the simplest way to go about plotting such interdependency is to make a frame of multiple bivariate plots. In each plot,

you choose different values for the un-viewed variables. Then by comparing the plots to one another, you can see how big of a difference the changes make.

Here's how you might accomplish this visualization, for the tulip data. I'm going to make three plots in a single panel. Such a panel of three plots that are meant to be viewed together is a *triptych*, and triptych plots are very handy for understanding the impact of interactions.

First, just draw samples from the naive posterior as usual. I'm also going to rename the columns in the resulting posterior matrix, `post`, so that they are parameter names instead of `lm`'s variable naming scheme:

R code
6.21

```
post <- sample.naive.posterior(m2c)
colnames(post) <- c("a", "bw", "bs", "bws")
```

Now for the plotting. Here's the strategy. I want each plot to show the bivariate relationship between shade and blooms, as predicted by the model. Each plot will plot predictions for a different value of water. For this example, it is easy to pick which values of water to use, because there are only three values: -1 , 0 , and 1 (this variable was centered, recall). So the first plot will show the predicted relationship between blooms and shade, holding water constant at -1 . The second plot will show the same relationship, but now holding water constant at 0 . The final plot will hold water constant at 1 . In addition, in each plot, I'll show only the raw data that has the water value appropriate in each case.

You already know how to produce each plot, using the same kind of `sapply` code from previous chapters. I'm going to wrap that kind of code in a loop now, and iterate over the three values of `water.c`. Here's the code:

R code
6.22

```
par(mfrow=c(1,3))
for ( w in -1:1 ) {
  dt <- d[d$water.c==w,]
  plot( blooms ~ shade.c , data=dt , col="slateblue" ,
        main=paste("water =",w) , xaxp=c(-1,1,2) , ylim=c(0,362) ,
        xlab="shade (centered)" )
  x.seq <- -1:1
  mu <- sapply( x.seq , function(z)
    mean(post$a + post$bw*w + post$bs*z + post$bws*w*z) )
  mu.ci <- sapply( x.seq , function(z)
    HPDI(post$a + post$bw*w + post$bs*z + post$bws*w*z) )
  lines( x.seq , mu )
  lines( x.seq , mu.ci[1,] , lty=2 )
```

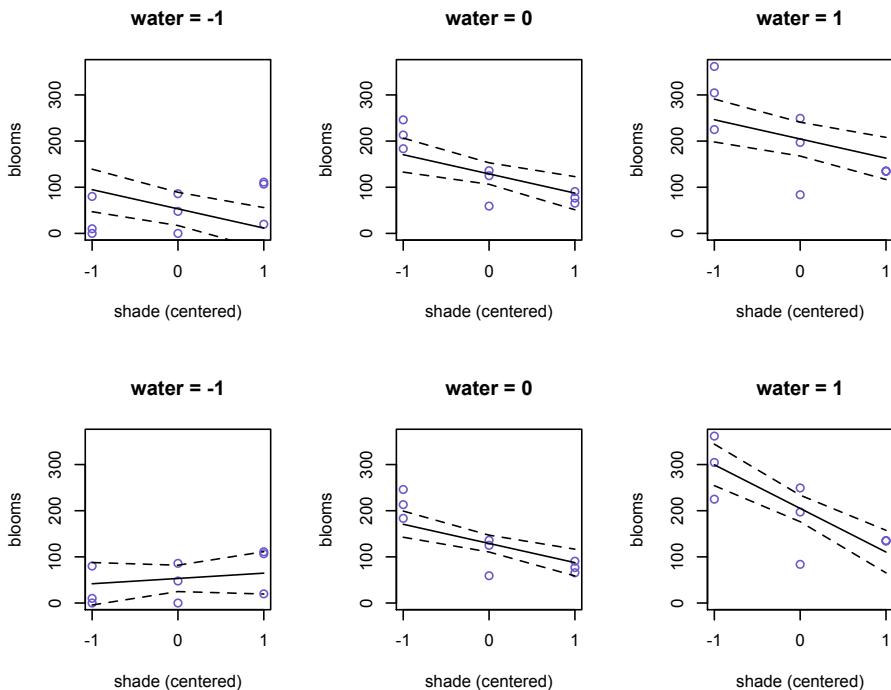


FIGURE 6.7. Triptych plot of predicted blooms across water treatments, without (top row) and with (bottom row) an interaction effect. Blue points in each plot are data. The solid line is the MLE mean and the dashed lines give 95% HPDI of the mean. Top row: Without the interaction, model `m1c`. Each of the three plots of blooms against shade level is for a different water level. The slope of the regression line in each case is exactly the same, because there is no interaction in this model. Bottom row: With the interaction, model `m2c`. Now the slope of blooms against shade has a different value in each plot.

```

  lines( x.seq , mu.ci[2,] , lty=2 )
}
```

The first line uses `par`, which manipulates graphical *parameter* settings, tells R's plot window to divide itself up into one row and three columns. This prepares the panel to be a triptych. Then the `for` statement defines a loop that will assign values -1 , 0 , and 1 to `w` on each pass. The rest of

the code merely extracts the raw data with the right `water.c` value for each pass through the loop and then computes and plots the predicted mean and 95% confidence interval of the mean.

FIGURE 6.7 shows the results, for both the model without the interaction (top row) and the model with the interaction (bottom row). In each plot, the horizontal axis is shade level, from low to high. The vertical axis is `blooms`. The blue points in each plot are those individual cases that match the water value displayed at the top of each plot. The solid lines are the predicted mean in each case, and the dashed lines are the 95% confidence intervals of those means. Note that the non-interaction model in the top row displays exactly the same slope in each plot within the triptych. The height of that line does change quite a lot across water values. But it always slopes downward at the same rate, in each plot. The interaction model on the bottom row, in contrast, shows a different predicted slope for each value of water. When water is at its lowest, on the left, shade has little effect at all, being nearly flat across all values of shade. There is a very weak positive trend, but substantial uncertainty about it. At the mean water value in the middle plot, increasing shade clearly reduces the size of blooms. Going from the least (-1) to the most shade, blooms are approximately halved in size. At the highest water value, on the right, the slope becomes even more negative, and shade is an even stronger predictor of smaller blooms, as shade increases. Going from the least to the most shade, bloom size declines by about two-thirds.

What is going on here? The likely explanation for these results is that tulips need both water and light to produce blooms. At low water levels, shade can't have much of an effect, because the tulips don't have enough water to produce blooms anyway. At higher water levels, shade can matter more, because the tulips have enough water to produce some blooms. At very high water levels, water is no longer limiting the blooms very much, and so shade can have a much more dramatic impact on the outcome. The same explanation works symmetrically for shade. If there isn't enough light, then more water hardly helps. You could remake FIGURE 6.7 with water on the horizontal axes and shade level varied from left to right, if you'd like to visualize the model predictions that way.

6.4.5. Centering for numerical reasons. The first reason you might center predictor variables is to aid in interpreting parameter estimates. However, predictions for centered and un-centered models should always be the same. So if you are prepared to plot predictions as a path to interpretation, centering isn't always much help. It can even hurt,

because centered variables are no longer on their natural measurement scale. As a result, they can actually be harder to interpret, sometimes. In the tulip example, it was no problem to remember that a water value of zero (0) is the mean. But suppose you are using years of age in a model. Then centered age is equal to zero for the average age in the data. So at least one extra step in inference is needed, to translate plotted predictions back to the scale you measured age on originally.

Quite another reason to center variables, however, is that it can improve estimation. The tulips data will serve to illustrate this problem, as well. Now we're going to fit the interaction model again, using `mle2` this time. Here's the code to do it, using the un-centered shade and water variables:

```
m2.mle2 <- mle2( blooms ~ dnorm(
  mean=a + bw*water + bs*shade + bws*water*shade , sd=sigma ) ,
  data=d ,
  start=list(a=mean(d$blooms),bw=0,bs=0,bws=0,sigma=sd(d$blooms)) )
```

R code
6.23

Everything will seem to go as planned, until you inspect the estimates this model arrives at.

```
precis( m2.mle2 )
```

R code
6.24

```
Caution, model may not have converged.
Code 1: Maximum iterations reached.

      Estimate Std. Error    2.5%    97.5%
a     -35.04912  71.674485 -175.52853 105.430288
bw    129.05355  33.016032   64.34331 193.763782
bs    13.74203  32.729930  -50.40745  77.891518
bws   -31.36487  14.909497  -60.58694  -2.142789
sigma 46.64502  6.935002   33.05266  60.237373
```

The `precis` output warns you about estimates, because “maximum iterations” were reached. Huh?

To understand what's happened here, first let's confirm that the new fit is indeed different than the previous one. Let's compare the MLE's here to those from the same model, fit with `lm` before:

```
coef(m2)
coef(m2.mle2)
```

R code
6.25

(Intercept)	water	shade	water:shade
-150.81074	181.50500	64.10056	-52.85167

a	bw	bs	bws	sigma
-35.04912	129.05355	13.74203	-31.36487	46.64502

These new estimates are quite different from the ones `lm` found. Which set of estimates is correct? The set produced by `lm` is correct. You can get an indication of this by looking at the log-likelihood of each model:

R code
6.26

```
logLik(m2)
logLik(m2.mle2)
```

```
'log Lik.' -141.232 (df=5)
'log Lik.' -143.2222 (df=5)
```

The log-likelihood closer to zero for `m2` indicates a better fit. It looks like `mle2` has failed us here. It didn't actually find the maximum likelihood estimates.

What has gone wrong? The problem is that, unlike the OLS method of `lm`, maximum likelihood has to search for the estimates. By default, `mle2` will only search so long before giving up. With the uncentered predictor variables, the MLE for the intercept is a very long way indeed from the average blooms value, about 129. As a result, `mle2` spends a lot of time searching. It will eventually get there, but inefficiently. R will actually tell you that it is worried, because it gave up, if you look at its default display information:

R code
6.27

```
show( m2.mle2 )
```

```
Call:
mle2(minuslogl = blooms ~ dnorm(mean = a + bw * water + bs *
    shade + bws * water * shade, sd = sigma), start = list(a = mean(d$blooms),
    bw = 0, bs = 0, bws = 0, sigma = sd(d$blooms)), data = d)
```

Coefficients:

a	bw	bs	bws	sigma
-35.04912	129.05355	13.74203	-31.36487	46.64502

Log-likelihood: -143.22

Warning: optimization did not converge (code 1:)

Look at the bottom line of the output. When the optimization does not "converge," this usually means R gave up before it found the maximum likelihood estimates.

There are several ways to fix this issue. One path is to toy around with different optimization algorithms in `mle2`. You could tell `mle2` to search longer or tell it to use a different search strategy. For example, if you add the parameter `method="Nelder-Mead"` to the call to `mle2`, it'll find the MLE this time.⁸⁵

Another path is to try centering the predictor variables, first. Why might this help? When you center the predictors, recall, the intercept parameter α should have its MLE at the grand mean of the outcome, as explained previously in this section. This means `mle2` won't have to search nearly as much to find the MLE for all of the parameters. So let's fit the model again, using the centered variables this time:

```
m2.mle2.c <- mle2( blooms ~ dnorm(
  mean=a + bw*water.c + bs*shade.c + bws*water.c*shade.c ,
  sd=sigma ) ,
  data=d ,
  start=list(a=mean(d$blooms),bw=0,bs=0,bws=0,
  sigma=sd(d$blooms)) )
coef(m2c)
coef(m2.mle2.c)
```

R code
6.28

	(Intercept)	water.c	shade.c	water.c:shade.c	
	128.99370	75.80167	-41.60278	-52.85167	
	a	bw	bs	bws	sigma
	128.99370	75.80132	-41.60259	-52.84823	45.22457

Now the estimates are the same, whether you use `lm` or `mle2` to fit the model. Centering has estimation advantages.

You might think that all this example demonstrates is that you'd be better off always using `lm` instead of `mle2`. First, there are other, more subtle numerical advantages to centering that apply to both OLS and MLE algorithms. Second, in the context of purely Gaussian models for which you aren't interested in the posterior density of σ , that may be so. But in later chapters, you'll meet non-Gaussian regressions that cannot be fit using the OLS shortcut of `lm`. In those cases, you have to use some kind of estimation procedure that will be sensitive to starting conditions and the distance of the MLE from them. At that point, you'll need to worry more about checking for convergence. By centering the variables, you can make the estimation process a lot more reliable.

6.5. Higher-order interactions

Still looking for a plausible three-way interaction in a real analysis. Might just fall back on boring age/sex/race examples, predicting income or education.

6.6. Multicollinearity returns

Dangers of multicollinearity from higher order interactions, esp. involving dummy variables. Centering to help reduce collinearity among interactions of continuous variables.

7 Generalized Linear Models I: Meet the Family

This chapter introduces an approach to multivariate statistical modeling known as *generalized linear modeling*. Generalized linear models (GLM's) allow for a wide variety of non-Gaussian outcome variables to be modeled as functions of multiple predictor variables. These approaches are very powerful, especially compared to coercing data to be Gaussian or falling back on “non-parametric” procedures. However, GLM's also introduce some new obstacles in model design and estimation. This chapter describes the general approach and uses simple exponential and gamma regressions, often called *survival analyses*, to introduce the structure of GLM's.

7.1. The Tyranny of Gauss

The Gaussian distribution is common and useful. Up to this point, it's been the only outcome distribution—stochastic node—that we have seriously modeled. And you can get quite far with it, because so many measures in the world converge towards a Gaussian shape.

But there are just as many measures that either never converge to a Gaussian shape or do so only slowly. Distances, durations, counts, and ranks are the most common kinds of measurements that can frustrate a Gaussian model. What all of these measurements have in common is that they are *bounded*, either on one or both sides. Distances, durations, and counts cannot be less than zero. Ranks—relative positions of items in a list, usually—are bounded on both sides, because they have definitive minimums and maximums.

Why does this matter? It sounds like mathematical fuss. It matters because a Gaussian model will happily predict impossible outcomes. This means that the likelihood of all values will be at least somewhat off, because the Gaussian density will assign some of its mass to impossible values either below zero or above to maximum. In cases in which much of the data is near a boundary, the likelihoods will be off by a huge margin, resulting in misleading estimates and nonsensical predictions. The

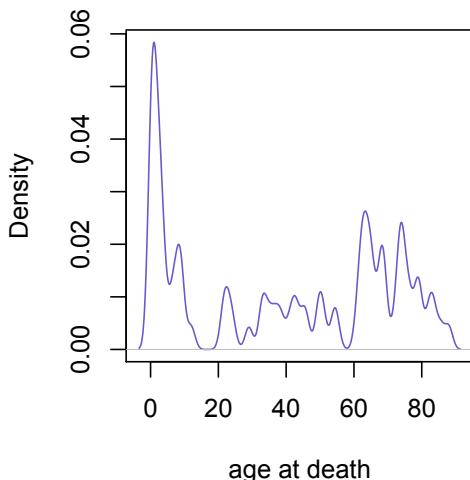


FIGURE 7.1. Distribution of ages at death, for 83 !Kung San individuals who lived in the middle of the 20th century. Many of the deaths happen before age 10, and many others after age 60. These data are not approximately Gaussian, and no matter how much sampling we do, they never will be.

nonsense can cascade through the chain of inference, leading approximations like BIC and AIC to perform badly.

Don't take my word for it, though. Let's try an example. The data contained in `data(Howell2)` are demographic data for the Dobe area !Kung San, compiled from interviews conducted by Nancy Howell in the late 1960's.⁸⁶ This is the same population you encountered in Chapter 4. Now we'll have some more variables to work with.

R code
7.1

```
library(rethinking)
data(Howell2)
d <- Howell2
```

This data frame has six columns. We're only going to be concerned for now with the first: `age.at.death`, which is just an individual's age at time of death. There are many missing values in this column, because most individual's in the census did not die during the study period. We'll return to this issue later in the chapter.

For now, take a look at the distribution of the ages at death.

R code
7.2

```
dens( d$age.at.death )
```

I reproduce this plot in FIGURE 7.1. Like nearly all non-industrial human populations, the 83 deaths represented here are clustered first at very young ages. A third of all 83 deaths happen before age 10. Then fewer

people die between 10 and 20 years old, again typical of non-industrial populations. The remaining ages at death are spread along ages above 20, rising as time goes on.

These measures—the ages at death—are events that happened in the world. They are the result of complex biological and ecological and political processes. But they are not Gaussian measures. While the sample here is quite limited, you could get your hands on a huge sample of deaths for many world populations, and none of them would be approximately Gaussian. Only in the most pampered urban populations with extensive investments in end-of-life extension do data of this type start to look Gaussian in shape, with symmetrical variance far from the hard boundary at zero. But even in those cases, the right side tends to be sharply truncated, as everything starts breaking rapidly, imposing a harsh upper limit on lifespan.

So what would happen, if you tried to fit a Gaussian outcome model to these data? Well, the posterior density itself won't tell you the model is broken. But you will get absurd predictions. Let's walk through it. First, fit a regular Gaussian model:

```
library(bbmle)
aad <- d$age.at.death[ complete.cases( d$age.at.death ) ]
m.bad <- mle2( aad ~ dnorm( mean=mu , sd=1/tau ) ,
  data=list(aad=aad) ,
  start=list(mu=mean(aad),tau=1/sd(aad)) )
```

R code
7.3

Note that you have to reduce down to complete cases, first. Otherwise, `mle2` is going to give you a mysterious error message. The value `NA` has no likelihood, so a vector of data contained even one `NA` will have no likelihood. A caveat to this is that sometimes you actually do want to model missing values, and indeed later in this chapter you will actually compute the likelihood of `NA` in this vector, in a sense.

Okay, what's happened here. Take a look at the estimates, to see that this approach does give a fine estimate of the mean age at death:

```
precis( m.bad )
```

R code
7.4

	Estimate	Std. Error	2.5%	97.5%
mu	39.47385542	3.335486046	32.9364229	46.01128794
tau	0.03290799	0.002553967	0.0279023	0.03791367

You can compare that estimate, 39.47, to the result of `mean(aad)`. It's a very good estimate, because estimating the mean of single distribution is just about the easiest task you can put to maximum likelihood. But when

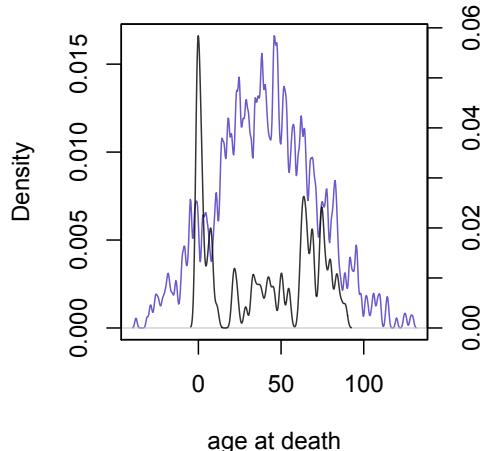


FIGURE 7.2. Simulated ages at death, shown in blue, produced from the naive posterior derived from the Dobe !Kung census data, shown here in black. Because the model assumed the ages are Gaussian, the simulated ages are Gaussian, even though this is clearly absurd.

we turn to making this model *predict* ages at death, is it a comic disaster. To generate predictions, sample from the naive posterior and generate a Gaussian age-at-death for each sample. Then you'll have predictions that incorporate all of the uncertainty in the estimates. Here's the code:

R code
7.5

```
post <- sample.naive.posterior( m.bad )
post$sigma <- 1/post$tau
pred.aad <- rnorm( 1000 , mean=post$mu , sd=post$sigma )
```

Note the trick of using `tau` to produce samples for `sigma`, detailed back in Chapter 4 (page 227). You can plot the density of these simulated ages at death to see what they look like. I produce their density, compared to the original data that produced the estimates, in FIGURE 7.2. The simulated ages at death are shown in blue, while the original data are shown in black. Not only does the Gaussian model concentrate predicted deaths at the mean, which is sparsely populated in the actual data, but it also happily predicts negative ages at death and lifespans far beyond the maximum observed. It does a poor job of prediction in almost every sense, aside from getting the mean right.

This is, by the way, a powerful example of why some kind of model check, outside of estimating posterior probability, will always be necessary. The posterior didn't tell us that this model is a bad one. Instead, checking its implied predictions against the data did. That doesn't mean that a significance test is needed nor even particularly helpful. But it

can mean comparing the implied predictions, or sampling distribution, of the fit model against the observations, like in calculating a P -value, can be helpful.

If this model can't begin to predict the distribution of ages at death, then it isn't a good model. Now, readers who know about *survival analysis* might object that no one would try to fit a Gaussian linear regression to data of this type. Well, one might hope not. But it does happen, just like scientists routinely fit Gaussian models to all manner of count and proportion and distance and duration data. The sin I committed above is much rarer than that of modeling counts as Gaussian variables, thankfully. Sometimes, one is lucky and can get away relatively unscarred, because interest lies only in comparing means between groups, and little of the data lies near zero. But getting lucky in this way is hardly a defensible methodology. And if one's attention ever turns to actual prediction or stronger inferences from the posterior, then it will be much harder to get away unscarred.

And while it's true that all models are false, only useful approximations, that isn't license to ignore how useful they are. If you never attempt the non-Gaussian model, then you won't know if inferences change. And if you never attempt a model that can make reasonable predictions on the scale of measurement, then you probably won't think seriously about prediction.

Luckily, you don't have to subscribe to the Gaussian outcome model in all cases. In the past, when computers were slow (or unavailable) and software rather incapable (by modern standards), there were basically two options. Either you assumed a Gaussian outcome or you fell back on "non-parametric" procedures with low power and no ability to model complex relationships. It was a necessary compromise, in most cases. But now computers are very fast indeed. Most readers probably own "smart" phones that are vastly more powerful than the computers that sent humans to the moon. And statistical software is now very capable, largely as a result of research on numerical algorithms that began in earnest only in the second half of the 20th century.

Now what used to be a necessary compromise is a tyrannical historical inertia, the Tyranny of Gauss. Under this indifferent, impersonal regime, young scientists are still being taught the old compromise. Researchers continue either to overuse the Gaussian outcome model or to fall back on underpowered non-Gaussian procedures that provide little

more than null hypothesis significance tests. These are the two syndromes of the Tyranny of Gauss. I call the first *coercion*, because it involves unconsciously coercing non-Gaussian data into a Gaussian outcome model. I call the second *surrender*, because it entails giving up on serious inference, as soon as an outcome variable is realized to be non-Gaussian.

I'll say a little more about each of these syndromes, before moving on to the bulk of the chapter, in which I show you how to justify and model natural non-Gaussian distributions. Keep in mind that, as always, blame for the Tyranny of Gauss is hard to pin on individual scientists. We certainly can't pin it on Gauss.⁸⁷ The historical and cultural forces that have lead to the creation and inertia of these practices are complex. I detail the syndromes here to raise consciousness of them, not to point fingers.

7.1.1. Coercion. In some sub-disciplines, Gaussian coercion is the norm. For example, a very common practice among some zooarchaeologists, mainly those following Grayson's book *Quantitative Zooarchaeology*,⁸⁸ is to conduct bivariate Gaussian regressions in which one count variable is regressed on another. Often the counts are log-transformed first. Sometimes, these analyses are followed up by comparing the estimated slopes (β coefficients) from distinct bivariate regressions, using *t*-tests. This sort of thing has been going on for more than two decades now. A recent example of this syndrome appeared in a 2008 issue of *Journal of Human Evolution*.⁸⁹ As always, it's a shame to see smart people suffering under the Tyranny of Gauss. These folks are honest victims of a specialized statistical culture that has been cut off from most of the inferential advances of the second half of the 20th century.

What is wrong with the practice of fitting Gaussian regression models to count data? First, counts in these contexts are frequently near zero. This means a Gaussian likelihood is hugely misleading, because it assigns high probability to impossible events, like negative values and non-integers, and too low probability to counts above the mean. But even when counts are not small, the Gaussian model cannot properly account for sample size. A count of "6" is treated as a single observation in a Gaussian model. In truth, it may be 6 (or more) observations. This failure to correctly deal with sample size is ironic, because much of Grayson's book is aimed at correcting counts for sampling intensity and overall density of remains found. But regressing one measure of sample size against another will not correctly adjust likelihoods for sampling design. Instead, we must use an actual count model.

Taking the logarithm of the counts first doesn't really fix the problem, either. If there are any zeros in the data, then you have to add a constant before taking the log, and the choice of constant is not harmless. But even when there are no exact zeros, the Gaussian model will still predict impossible values, both because it will predict negatives and non-integers. On a more basic level, all counts demonstrate correlations between the mean count and the variance in counts. A Gaussian regression will assume, unless you strive to make it behave otherwise, that the mean and variance are completely unrelated. Unless you are quite lucky, you could make poor inferences, whether the counts are transformed or not. And if you want to check model predictions against data, as you always should, it will be very difficult to do, if the raw data are counts and the model predicts Gaussian outcomes, in which the variance is unrelated to the mean.

Frequently, scientists will also regress proportions—the count of one kind of thing over the total count of all kinds of things—against explanatory variables. This procedure is even worse, because proportions are not data. Proportions divide away all sample size, because $1/2$ and $10/20$ are the same proportion, but reflect very different likelihoods. The same holds for proportion-like constructed variables like evenness and diversity indexes. These measures include no information about sample size, and so they should not be used to make probability inferences, if you can avoid it. Likewise, these measures are bounded hard at zero and one, and so they easily diverge from approximate Gaussian shapes.

In the 1980's, binomial and multinomial regressions were certainly rare, so I'm inclined to view Grayson's 1984 book as an honest attempt to deal with sampling and taphonomic issues. It came at a very awkward time, and the empirical subject matter is not easy. It's the book's legacy that is more problematic. In population biology, much work has gone into designing species area models that are based on explicit counts and really accomplish much of what the zooarchaeologists need.⁹⁰ But since analyses in the Grayson tradition continue to be published in top zooarchaeology journals, and people continue to cite Grayson as a guide on how to analyze faunal remains, the Tyranny is obviously in full force in at least some circles of that sub-discipline.

Cultural inertia can be strong indeed, and that inertia is what maintains the Tyranny, in many cases. If statistical practice across disciplines were more integrated, it would be easier to cancel such inertia. Whatever the causes, it isn't my intention to single out these victims of the Tyranny of Gauss for ridicule. They are merely doing the best they can

within a particular scientific tradition that suffers more than most under the oppressive regime. Indeed, operating under the Tyranny, people like Grayson and his students are actually thinking hard about how to get sample size information into an analysis. That is commendable. The Tyranny of Gauss walls off the path to proper solutions. The tradition is to blame, a small group of specialized scientists reinforcing the Tyranny of Gauss generation after generation, most shy about quantitative methods and eager to please their supervisors. Archaeology is a small field, and so it is probably more vulnerable than most to personalities with idiosyncratic views.

The next chapter focuses exclusively on count models, because they are so important and so often overlooked. If you doubt such models are often overlooked, perhaps because you had a better foundation than others, I can assure you that I've advised many senior colleagues and PhD students over the last decade who believed there are no complex regression models for counts. One senior colleague—a rather prestigious ecologist at my university, which is one of the most prestigious places in the world to do ecology—actually tried to argue that belief with me, when I suggested a humble Poisson regression model. He was honestly sure that you had to use either Gaussian models or give up on multivariate inference. Such a position is thankfully pretty rare in ecology these days, but it's very common in other fields that I frequently intersect with, including archaeology (as mentioned above), animal behavior and portions of psychology. Again, these are victims of a lagged curriculum and culture of statistical practice. Don't blame the victim.

7.1.2. Surrender. The ironic thing about count models is that many disciplines and sub-disciplines have long realized that Gaussian models are hazardous when data are frequencies. In animal behavior, for example, it's rare to find anyone modeling counts as a Gaussian variable. And in fairness, many archaeologists (including Grayson) use “non-parametric” correlation to analyze some counts, as well as chi-square contingency tests. But these are just examples of the other syndrome of the Tyranny of Gauss, falling back on relatively powerless procedures, instead of using powerful likelihood-based count models. This is the next syndrome, the path of surrender.

So-called “non-parametric” procedures have two major limitations. First, they do not estimate any quantity that can be used to make a prediction. Instead, they merely check a pattern against a null hypothesis such as “entirely random.” In short, they merely aim to provide P -values, often without any estimate of effect size. Second, they do not usually admit multivariate relationships to be modeled. This means that at best

a bivariate relationship can be studied. But since even then there is no estimate of effect size that can be used to make predictions—no distributional assumptions means no likelihoods means no sampling—this isn’t much help in studying anything outside the most highly controlled experiments and simplest hypotheses. Ironically, non-parametric procedures are most common in fields in which samples are chronically small and so model-based Bayesian inference would provide the greatest marginal benefits.

Perhaps the worst procedure to participate in the syndrome of surrender is the Pearson chi-square (χ^2) contingency test, and tests like it.⁹¹ A contingency table is a table of categorical data, in which rows and columns represent different dimensions of categorization. The chi-square procedure advises whether the observed counts of these categories differs from some reference distribution, usually “chance.” There are cases in which even your author is not hostile to this procedure, such as in well-calibrated model checks, once a posterior has been estimated and predictions are to be compared to observations.

Otherwise, there are now much better tools on everyone’s desktop. This testing procedure is doubly frustrating, because not only does it not permit any strong inferences beyond “not identical to expectation,” but it also depends upon large-sample approximations of count (multinomial) likelihood to Gaussian likelihood. A typical “significant” result from a chi-square test only advises that at least one cell in the contingency table is different from expectation, at the arbitrary 95% level. As always, with sufficient sample size, you are guaranteed a significant result. But in small samples, a problem arises because the procedure assumes that the probabilities of each observed count are normally distributed. This leads to the common concern about small cell counts. Many scientists were taught not to use a chi-square contingency test, when any cell has a count less than 5. This is an arbitrary rule, but it arises from a sensible concern with the Gaussian approximation of the counts. So what we end up with is a procedure that performs best when it is needed least. And when it does perform, it tells us only that some cell, someplace, is arbitrarily unlikely under a model that is certainly false to begin with, because it is a point hypothesis that can always be rejected with enough evidence. It estimates nothing and provides no posterior density, because it was not designed to.

Given the widespread availability of Poisson, binomial and multinomial models (see next chapter), all of which allow for multivariate prediction equations, the Pearson chi-square contingency test needs to be buried, at least as far as its use as a fallback from regression. There

is no circumstance in which this procedure will provide more information or stronger inferences than the appropriate non-Gaussian regression model. This isn't to say that Pearson's proposal didn't have its time and place. When Pearson published his procedure in 1900, it was not practical to imagine estimating a Poisson or multinomial regression with many predictor variables. At that point in the chronology of statistics, Fisher had yet to illuminate maximum likelihood, and Bayesian inference had been sidelined for decades already, due to brilliant but obstinate opponents like John Venn at Cambridge.⁹² There were fewer tools available then, and the computational tools we take for granted now did not yet exist. And so the inferential power of Bayesian approaches could not be simply demonstrated.

We really can reliably fit non-Gaussian multivariate models, now. Most readers of this book likely own laptop computers and even mobile phones that are powerful enough to estimate complex multilevel non-linear models in a fraction of the time a desktop computer needed half of a decade ago. Certainly care is needed in getting the fit right, but that is hardly a good reason to fall back on a weak, specialized procedure like the Pearson chi-square test. There are a bunch of other "non-parametric" procedures that similarly deserve to be replaced by non-Gaussian parametric models: Wilcoxon, Spearman, Kruskal-Wallis, Mann-Whitney U, and many others. None of these procedures allow for strong multivariate inference, and none of them produce useful estimates.

It isn't that these procedures didn't have their use, once upon a time. I also believe, like with most "testing" procedures, that these tools are still useful in well-calibrated contexts. It's just that for general inference they've been superseded by much more powerful approaches. One family of the supplanting approaches is generalized linear models. And the rest of this chapter and indeed the book focuses on their construction, estimation, and interpretation.

7.2. Generalized Linear Models

The most accessible way to resist the Tyranny of Gauss is to learn to construct, estimate and interpret *generalized linear models* (GLM's). This label is unfortunate, because it references mathematical definition rather than purpose, and it is easy to confuse with *general* linear models, which are quite different. Sadly, GLM is the standard terminology, so you have to know it.

Better to think of generalized linear models as non-Gaussian regression models. If you have an outcome you wish to model as a result of a multivariate process, but the outcome is a distance, duration, count,

rank or other non-Gaussian measure, then GLM's provide a practical and flexible approach. These models assume non-Gaussian stochastic nodes, but they use the same strategy of replacing parameters of the stochastic node with additive models that incorporate predictor variables. So most of what you've already learned about building and interpreting Gaussian regression models remains relevant for non-Gaussian regression. There are some new wrinkles, as you'll see. But the additional power and predictive precision are worth it.

In this book, I focus mainly on GLM's that use a family of outcome distributions known as the *exponential family*. Indeed, many people actually define GLM's as those regressions that use exponential family distributions. The normal distribution you've come to know and love is the most famous member of the family, but there are others which are just as useful and natural. These distributions are scientifically important and mathematically convenient, partly because they arise naturally from many real-world processes, just like the normal distribution. The naturalness of these distributions helps us appreciate why, for example, the *exponential distribution* is a fundamental distribution of distances and durations.

But the overall strategy used in building GLM's is not constrained to the exponential family. I'll provide a couple of important applied examples, in the form of rank and ordered regression models. These are not exactly exponential family distributions, although they do build on top of them. Still, by learning how these distributions are built up from more basic considerations of probability, it'll help open your mind to the broader approach of seriously considering how to predict outcomes, even when they are measured on idiosyncratic scales. The goal is to build a model that addresses the measurement design of the data.

Here's the plan. Much of the remainder of this chapter will teach you the basics of modeling distances and durations, kinds of measures I like to think of as *displacements*. Displacements are continuous measures that contain only information about distance from a point of reference. They are always positive (or zero, in special cases). In the next chapter, I focus on count distributions and models. And then in the next, I discuss special distributions for awkward measures like ordered categories and ranks, as well as ways to mix exponential family distributions together to begin to model variability in underlying processes. All of this work is on a path towards multilevel models, in Chapter 10.

But before settling into the serious work of developing these GLM's, it will pay first to meet the most prominent members of the family. The important point to make, before moving on, is that these distributions

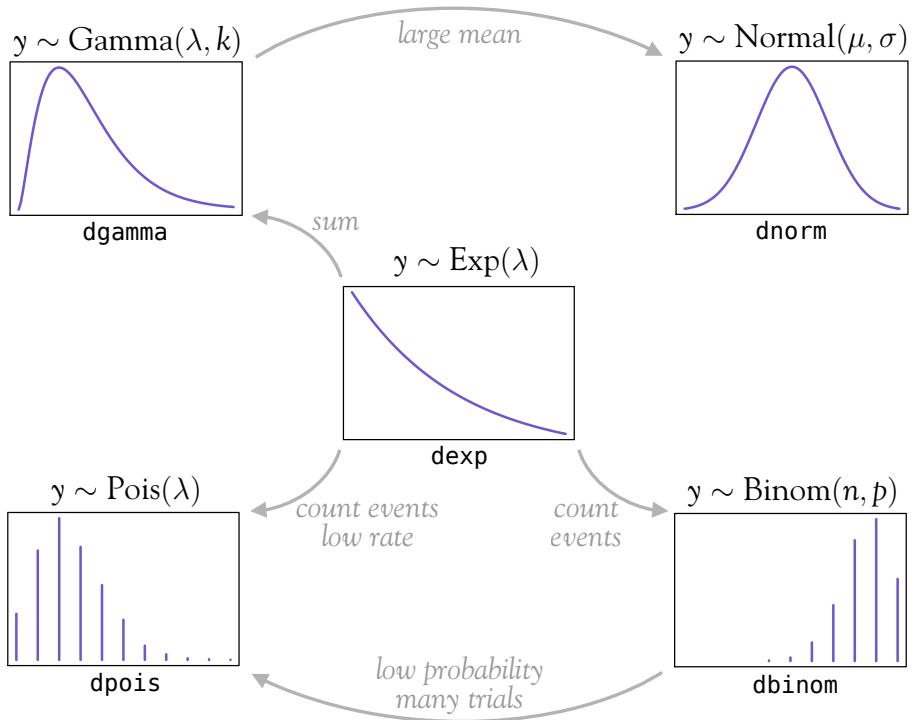


FIGURE 7.3. Some of the exponential family distributions, their stochastic nodes, and some of their relationships. Center: exponential distribution. Clockwise, from top-left: gamma, normal (Gaussian), binomial and Poisson distributions.

are not arbitrary choices. Rather, they relate naturally to the kinds of measures we wish to model. Then I want to foreground some of the additional issues in fitting and interpretation that come with GLM's. You'll meet these issues in detail as you develop the applied cases in the chapters to follow. But it'll help now to outline these issues for you, so you can understand how most of them arise from the same basic characteristic of generalized linear models: the variance of most non-Gaussian outcomes is not independent of the mean.

7.2.1. Meet the family. FIGURE 7.3 illustrates the representative shapes of the densities for the most common exponential family distributions used in GLM's. The horizontal axis in each plot represents values of a variable, and the vertical axis represents density. For each distribution, the figure also provides the notation for its stochastic node (above

each density plot), and the name of R’s corresponding built-in density function (below each density plot).

While there are other important distributions both in the family and outside it, you can usefully think of these distributions as the most common building blocks of probability modeling. Like the normal distribution (see Chapter 4), each of these exponential family distributions arises (approximately or exactly) from natural processes. Not only do they arise from natural processes, but they are also related to one another through natural processes and the transformations that happen during measurement. These are not *ad hoc* mathematical conveniences. Instead they are natural distributions that arise through many processes in both nature and the laboratory. Just like Gaussian distributions populate the world, so too do exponential, gamma, Poisson and others. Or the way I prefer to think of it, these distributions have strong informational “gravity” that attracts collections of measures towards them. Just like a collection of sums from (almost) any distribution tends towards a Gaussian distribution (Chapter 4), other ways of combining or transforming values lead to other distributions.

Later, both in this chapter and the next, I’ll provide heuristic explanations of some processes that give rise to these distributions, as well as how they relate to one another. For now, note the gray arrows in FIGURE 7.3. Each arrow is labeled with a process, whether it occurs naturally or through human measurement, that leads each distribution to generate another, either approximately or exactly. For example, as I’ll explain in more detail later in this chapter, summing values sampled from an exponential distribution (center) leads directly to the gamma distribution (top-left). This isn’t the only way to produce a gamma distribution, but it is a natural and common one. Furthermore, if you add enough exponential values together, the gamma converges to the normal. Beginning again with the exponential in the center, the count distributions at the bottom of the figure can arise by counting the number of exponentially-distributed events that happen in a finite interval of time. There are other relationships among these distributions, and further relationships between these distributions and information theory. But much of those details are beyond the scope of this book. I just want you to be aware of the wider context these distributions live in.

I’m going to split the introduction of these distributions across this chapter and the next. In this chapter, I introduce and show you how to build models based upon the *exponential* (FIGURE 7.3, center) and *gamma* (top-left). These are natural distributions of durations and distances. Both durations and distances are measures of displacement, absolute

length from some point of reference. Therefore, unlike truly continuous measures, durations and distances are strictly positive or zero. They are never negative. A direct consequence of this fact is that the pattern of variation around the mean always varies with the mean. This is quite unlike the Gaussian distribution, in which the mean and variance are independent (unless modeled otherwise).

The same general relationship between the mean and the variance is true of count distributions, as well. In the next chapter, I focus on the count distributions at the bottom of FIGURE 7.3, the *Poisson* (bottom-left) and *binomial* (bottom-right) distributions. Counts are discrete positive integers—like 1, 2 and 3—or zero. Again, they are never negative. And again, a consequence of this fact is that the variance of a count distribution is closely related to its mean. Both the Poisson and binomial can be derived from the exponential distribution, as indicated in FIGURE 7.3. And the Poisson can be derived from the binomial, by choosing special values for its parameters.

It isn't important in this book to grasp the mathematical details of the relationships among these distributions, for the most part. Instead, I describe them in order to help the reader appreciate when each distribution is perhaps most helpful, natural and interpretable.

7.2.2. Linking models and distributions. The first component of the GLM strategy is to open up many natural probability distributions to apply to our outcome measurements. But just like with ordinary Gaussian models, we have to work with the parameters of these distributions in order to build models that include predictor variables, other measurements. The choice of how to associate the basic parameters of the outcome distribution with the predictor variables is called the *link*.

To remind you, in the case of an ordinary Gaussian regression model with a single predictor variable, the linking strategy is to replace the parameter μ , the mean of the Gaussian distribution, with a *linear model* that contains the predictor variable. This leads to the now-familiar model definition:

$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \sigma), \\ \mu_i &= \alpha + \beta x_i. \end{aligned}$$

The mean for each case i is linked to an additive model that includes x_i and the two parameters, α and β .

If you choose a different outcome distribution—a non-Gaussian stochastic node—then the fundamental parameters of that distribution will not be μ and σ . Importantly, most distributions other than the Gaussian do not have means that are independent of their standard deviations.

So in most cases it isn't possible to just replace the parameter for the mean with an additive model. Some other link is needed. And many fundamental parameters are not continuous like μ . Instead, they must be positive (like σ) or bounded between zero and one. These facts can make establishing a useful link a little more complicated.

As this book introduces each outcome distribution, it will also introduce common linking strategies for associating distribution parameters and predictor variables. The important principle to remember is that there is no single right way to accomplish this goal. Your hypotheses tell you how you want the predictor variables to relate the aspects of the outcome, like its mean. The link is a mathematical strategy for modeling that hypothesis. Like all models, all you need to hope for is that it be useful.

But while there is no one right way to build a link, there are well practiced and useful ways to accomplish it. Depending upon the distribution chosen—exponential, binomial or another—there are usually common links that provide a time-tested way to embed an additive model. Still, it's worth being critical of link choices. Just like with other aspects of a model, such as the prior and the likelihood function, there is always some subjective choice involved in the choice of a link. While scientists tend to be much more suspicious of priors than links, both can alter inference in powerful ways.

7.2.3. Living with the family. Moving from Gaussian models to models based upon other distributions does have some costs. We need GLM's, because there are many important types of data that are not even approximately Gaussian and cannot be usefully coerced into Gaussian form through transformation. But these non-Gaussian distributions all share the inconvenient feature of having patterns of variation that are related to the mean. For example, if the average count of some event is close to zero, then the variation around that mean must extend further above the mean than below it. This is just because counts cannot go below zero. The distribution will be skewed. As the mean increases, the pattern of variation will change, usually becoming more symmetrical.

Such relationships between the mean and variation in a collection of measures is natural. Distances and counts really cannot be negative, and their means and variances really do covary. It's how nature actually works. But in the practice of statistics, it presents some new problems for us.

7.2.3.1. *Needle in a non-linear haystack.* Search is harder.

7.2.3.2. *Ceilings, floors and interpreting parameters.* Everything interacts.

7.2.3.3. *The weakest link.* Link functions matter and even smuggle in assumptions.

7.3. The Exponential Distribution

A curious thing about machines and organisms alike is that they tend to fail predictably. I don't mean that it is easy to predict when any particular machine will break or organism will die. Rather I mean that the distribution of these failures tends to take a characteristic form, or one very close to it.

Suppose there is a machine with a number of components necessary for its function. When any one of these components fails, the machine fails. Now, also suppose each component will fail sometime between today and one year from now. Each component however has equal probability of failing on any day of the year, and each component fails independently of the others.

What will the distribution of times to failure look like? The answer depends critically on how many components there are in the machine (FIGURE 7.4). For one component (FIGURE 7.4, left), the distribution is naturally just uniform—any day between tomorrow and one year from now is equally likely. You can simulate the distribution for yourself with this one line of R code:

R code
7.6

```
y <- replicate( 10000 , min( runif(n=1,min=0,max=365) ) )
```

As you add components to the machine, increase $n=1$ to higher integers, the distribution quickly changes. For three components (FIGURE 7.4, middle), the distribution already piles up at low times to failure, with fewer and fewer machines surviving anywhere close to a full year. By the time there are 10 components to the machine (FIGURE 7.4, right), the distribution of failure times is nearly identical to another important ideal exponential family distribution, the *exponential*. The black curve superimposed in the third plot in FIGURE 7.4 is an idealized exponential distribution with the same mean as the empirical distribution shown in blue. Failure times have converged to an exponential density, in which the probability of failing at time T is:

$$\Pr(T|\lambda) = \lambda \exp(-\lambda T).$$

The parameter λ governs the rate of failure, and in these examples it is approximately $\lambda \approx n/365$, where n is the number of components in the

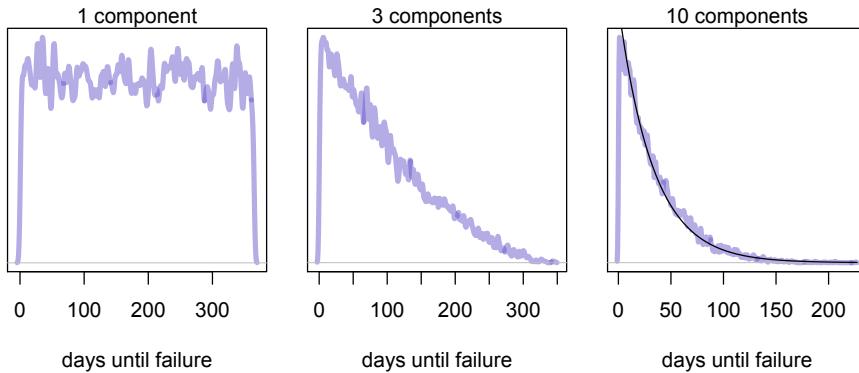


FIGURE 7.4. Times to failure for a machine (or organism) in which failure of any one component leads to total failure. In each plot, time to failure is on the horizontal axis and density is on the vertical. Assume that each component of the machine has an equal chance of failure on any day from now until a year from now. Time to failure is the time it took for at least one component to fail. Left: A machine with only one component shows the uniform distribution of the single component's failure probability. Middle: A machine with 3 components usually fails sooner rather than later. Right: A machine with 10 components exhibits a nearly perfect exponential failure density. Comparison to perfect exponential in black.

machine. As with many non-Gaussian densities, the exponential does not have separate parameters that independently determine its mean and variation. Instead, there is a single parameter, λ , that governs both.

As with the Gaussian distribution, lots of processes converge to an approximately exponential density. Perhaps the most fundamental type of process that will produce an exponential density is constant and proportional decay. If you begin with a 1000 machines and each machine has a constant, unchanging probability of failing through time, then the number of machines remaining after any fixed amount of time follows the exponential. The expected proportion remaining at time T will be:

$$\exp(-\lambda T).$$

The number of failures is highest at the start, because the most machines are still around to fail. After a while, there are very few machines left,

and so fewer fail per unit time. Still, they are failing at the same rate. Substituting other events for “failure” leads one to notice exponential processes in many natural systems, from atoms to ecosystems.

As with the Gaussian density in Chapter 4, there is a maximum entropy interpretation of the exponential density as well. If all we know about a collection of measures is their average *displacement* from some point, then the least surprising (maximum entropy) distribution for the measures will be an exponential distribution.⁹³

7.3.1. Survival analysis. As a fundamental distribution of displacements—distances in space or durations in time—the exponential is very handy for modeling. In this section, I provide a very brief introduction to its use in making GLM’s. This is a good place to start, because the exponential density has only one parameter, a rate, that determines both its mean and variance. And so it will be easier to first encounter some aspects of GLM’s in the context of exponential distributions.

On the other hand, the broader subfield that exponential GLM’s are part of, often called *survival analysis* (in the biological sciences) or *event history analysis* (in the social sciences), can be quite complicated. It is the topic of entire books.⁹⁴ And so here all I’m going to attempt is a simple application to illustrate how to get a linear model into the exponential density.

Add tenure data example

7.4. Gamma

Built up from exponential — add together exponential waiting times and get a gamma.

8

Generalized Linear Models II: Counts

8.1. Fingers and Toes

Summarize Spelke experiments on counting? Goal is to provide intuition for why uncertainty around a count scales with the count.

8.2. Binomial

A filtered exponential

The binomial density is, like you saw early in this book:

$$y \sim \text{Binomial}(p, n),$$

where y is a count (zero or a positive whole number), p is the probability any particular “trial” is a success, and n is the number of trials.

8.2.1. The importance of log-odds. How do we convert the parameters of the binomial distribution into an additive function of parameters and predictor variables? What is a good *link*? Several different strategies are available, but the best isn’t to just replace, for example, the probability p with an additive model. Instead, we define the log-odds of the event as an additive model. The log-odds of any event E is just:

$$\log \frac{p}{1-p}.$$

This is simply the logarithm of the odds. Why do you want to define the density through the log-odds? Why not the plain odds, or even just the probability p itself? A very basic answer, but one that is unsatisfying for most, is that log-odds are a *natural parameter* of the binomial distribution, much like τ but not σ , is a natural parameter of the normal. But so what? What does using log-odds give us that plain probability or regular odds do not?

Log-odds have several pragmatic advantages. First, unlike the probability p , the log-odds are *continuous*. They can take any value between $-\infty$ and $+\infty$. A probability like p is in contrast bounded between zero

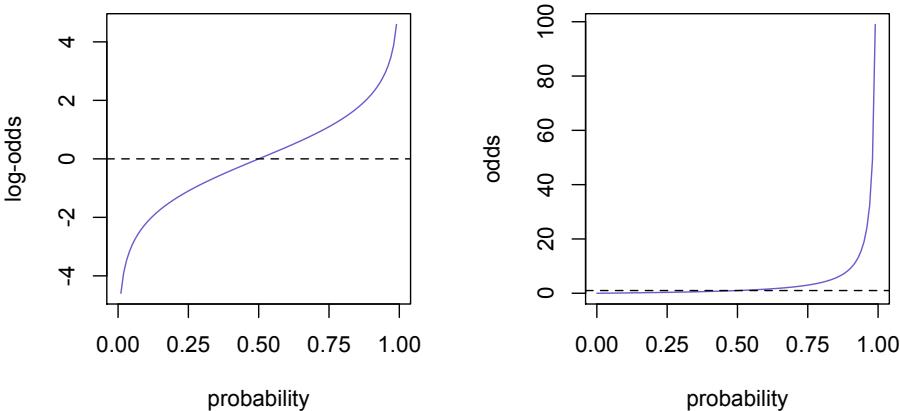


FIGURE 8.1. Log-odds versus ordinary odds. On the left, the log-odds of an event are plotted against the raw probability. The horizontal dashed line shows the log-odds value at which the event has probability one-half. Notice that the log-odds curve is perfectly symmetrical around this line. On the right, ordinary odds are plotted against probability. Again, the horizontal dashed line shows the value of the odds at which the probability is one-half. Now however, the curve is far from symmetrical around this line. Odds are not invariant to the choice of which event to focus on, while log-odds are invariant.

and one. Having a continuous scale to build our models upon will make them much easier to work with, as you'll see.

Second, unlike the plain non-log odds, $p/(1 - p)$, the log-odds are *symmetrical*: They accelerate at the same rate in both directions, as p increases above one-half or as it decreases below one-half. The same is absolutely not true of the regular odds. This will be easier to appreciate with a graph. In FIGURE 8.1, I show the contrast in symmetry between log-odds and ordinary odds. On the left, log-odds of an event are plotted against the probability of the event. The horizontal dashed line shows the log-odds value at which the event is equally likely to happen as not, $p = 0.5$. Notice that this occurs where the log-odds are equal to zero. Furthermore, the shape of the log-odds curve is perfectly symmetrical on both sides of zero, as if reflected in a mirror. In contrast, the plot

on the right shows that ordinary odds are highly asymmetrical. Again, the horizontal dashed line shows the value on the vertical axis at which the event has probability one-half. This is where the odds are equal to one. But now the curve is far from symmetrical on both sides of this line. Instead it accelerates towards infinity as the probability increases above one-half. Why is the symmetry of the log-odds superior to the asymmetry of the odds? We want a scale of measurement that isn't influenced by arbitrary measurement choices. The horizontal probability axis in both plots could just as easily go the other direction, measuring the probability of the event not happening. The log-odds plot would be unaffected by this decision, while the odds plot would change dramatically. Since the choice of which event to label on the horizontal axis is largely arbitrary—a matter of our convenience—it is important to have a scale of measurement that is invariant to such arbitrary decisions. Log-odds provide such a scale.

Finally, a reasonable model of plain probabilities and odds must be fundamentally *multiplicative*, because otherwise the model would allow these values to dip below zero, which should be impossible. In contrast, multiplicative relations, once logged, become additive. You've seen this already, when you learned that joint likelihood is the product of each individual likelihood, but the joint log-likelihood is the sum of each log-likelihood. And so like log-likelihood is additive, log-odds are fundamentally additive, and therefore lend themselves more easily to the generalized linear modeling framework.

8.2.2. The logit link. So we'll adopt the strategy of modeling the log-odds, instead of the probability p directly. More precisely, what this means is that instead of defining the binomial model this way:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n), \\ p_i &= \alpha + \beta x_i, \end{aligned}$$

we define the log-odds instead:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n), \\ \log \frac{p_i}{1 - p_i} &= \alpha + \beta x_i. \end{aligned}$$

So it's still true that we are using the GLM approach to model a unique probability for each case i . But the additive model defines the log-odds for case i , not the probability. If we made p_i itself a linear model, then it would be easy to find parameter values that would make p_i less than zero or greater than one. Since a probability should be between zero and one, that approach could cause problems. But the log-odds extend

symmetrically around zero to both $-\infty$ and $+\infty$, so making the log-odds into a linear model is a lot safer.

Okay, so what does this imply about the probability itself, p_i ? We have to care about the answer to this question, because in order to fit the model to the data, we have to calculate likelihoods. So we want to solve for p_i . Suppose for example that the additive model of the log-odds for case i is just called ϕ_i . In math speak, this implies:

$$\log \frac{p_i}{1 - p_i} = \phi_i. \quad (8.1)$$

Now we solve for the probability itself. Do this by taking (8.1) and solving for p_i . After a little algebra, you get:

$$p_i = \frac{\exp(\phi_i)}{1 + \exp(\phi_i)}.$$

You might recognize this probability as the *logistic*, with the log-odds sometimes called the *logit*.⁹⁵ The logistic function pops up all over science. I first learned it in biology, as the result of natural selection on the frequency of a favorable allele. But it has emerged here just as a consequence of the desire to be able to make our model about log-odds.

What we've done so far is establish a *logit link* between the additive model and the probability of an event. Now you need to see how to code this link.

8.2.3. Example: Prosocial chimpanzees. The data for this example come from an experiment aimed at evaluating the prosocial tendencies of chimpanzees.⁹⁶ Each chimpanzee was placed in a room with two levers, one on the left and one on the right. Across from the focal animal was a transparent divider, on the other side of which another chimpanzee would sit, in one of the treatment conditions. On each side of the divider, each lever was attached to a plate. When the focal animal pulled the left lever, the plates on the left side would move to be close enough to each animal for him or her to retrieve anything on it. As a result, if the focal animal pulled the left lever, whatever rested on the left plates became available to each animal. If he or she instead pulled the right lever, whatever was on the right plates became available.

There were two treatment variables to be concerned with for now. First, one of the two sets of plates, left or right, always had a “prosocial” option that had food on both plates, while the other side had food only on the side close to the focal animal. Think of the prosocial option as being 1/1, indicating food on both sides. The nonsocial option, in contrast, was always 1/0, indicating food only on the focal side. If the focal

animal pulled the lever on the 1/1 side, both animals received food. If he or she instead pulled the 1/0 lever, only the puller received food. The side of the 1/1 option was balanced across trials, to control for handedness preferences. So the first question to ask of these data will be: do the chimpanzees prefer the 1/1 option?

The second key treatment variable is whether or not another chimpanzee sat on the other side of transparent divider. This is the `condition` dummy variable, with 1 indicating there was another chimpanzee present. So the second question to ask of the data will be: do chimpanzees prefer the 1/1 option more when another animal is present? Or are they instead indifferent to the presence of the conspecific?

To address these questions, we will try to predict which side the focal animal pulls, conditional on which side had the 1/1 option and whether or not another animal was present. Since the outcome is binary (left or right lever), we'll use a binomial stochastic node and build a GLM with a logit link. Load the data from the `rethinking` package:

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
```

R code
8.1

Take a look at the built-in help, `?chimpanzees`, for details on all of the available variables. We're going to focus on `pulled.left` as the outcome, with `prosoc.left` and `condition` as predictor variables. There were seven different focal individuals in this experiment, and so it is useful to view the data for them individually, as in FIGURE 8.2. The vertical axes in both plots represent proportions of pulls. Each cluster of joined points are from the same focal individual. The open symbols correspond to trials in which another animal was absent, while filled symbols are those trials in which another animal was present. The circles indicate that the 1/1 ("prosocial") option was on the left side, while the triangles indicate it was on the right side.

The top plot in FIGURE 8.2 shows the data from the perspective of proportion of pulls on the left side. Averaging across trials in this way, you can see that most individuals presented a mix of left and right pulls. But some individuals, notably numbers 2 and 7, demonstrated strong side preferences. Since chimpanzees show handedness preferences much as humans do, perhaps this shouldn't be too surprising. Still, we'll have to cope with it when we estimate the models.

The bottom plot in FIGURE 8.2 shows the same data, now from the perspective of the "prosocial" 1/1 option. The side preferences aren't

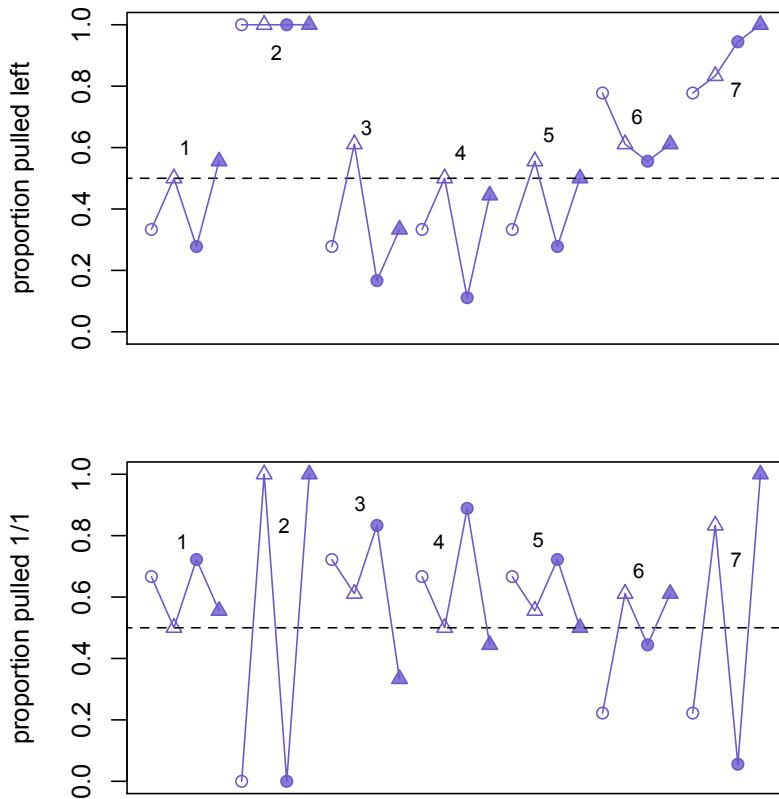


FIGURE 8.2. Proportion of trials in which an individual pulled the lefthand lever (top plot) and the 1/1 (prosocial) lever (bottom plot). Each group of four points are the data from a single individual chimpanzee, with the numbers corresponding to actor labels in the data. Circles represent pulls when the 1/0 option was on the left. Triangles represent pulls when the 1/1 (prosocial) option was on the left. Filled symbols represent pulls when another individual was present (social condition).

as obvious here, but now you can see that most of the averages (proportions) are above the 50-50 dashed line, indicating that overall, the chimpanzees did show at least a weak preference for the 1/1 option.

To get any further, we'll have to start modeling. First model, just the intercept:

$$\mathcal{M}_1 : \quad L_i \sim \text{Binomial}(p_i, 1),$$

$$\log \frac{p_i}{1 - p_i} = \alpha,$$

where L is a 1/0 variable indicating a pull of the lever on the left. To fit this model:

```
m0 <- mle2( pulled.left ~ dbinom( prob=logistic( a ) ) , size=1 ,
  data=d , start=list(a=0) )
precis(m0)
```

R code
8.2

Estimate	Std. Error	2.5%	97.5%	
a	0.320166	0.09022966	0.1433192	0.4970129

This implies an MLE probability of pulling the left lever of $\text{logistic}(0.32) \approx 0.58$, with a 95% quadratic estimate interval of 0.54 to 0.62. So we have to note at the start that the chimpanzees, on average, exhibit a small preference for the lefthand lever.

Now fit the next two models, which allow us to investigate the hypothesis. We don't fit a model with only `condition`, because we don't think there is any reason to suspect that chimpanzees pull the lefthand lever more or less when another chimpanzee is present. In contrast, we have reason to suspect that they pull the lefthand lever more when the lefthand lever is both 1/1 option and another chimpanzee is present. This means we want to estimate these additional two models:

$$\mathcal{M}_2 : \quad L_i \sim \text{Binomial}(p_i, 1),$$

$$\log \frac{p_i}{1 - p_i} = \alpha + \beta_P P_i.$$

$$\mathcal{M}_3 : \quad L_i \sim \text{Binomial}(p_i, 1),$$

$$\log \frac{p_i}{1 - p_i} = \alpha + \beta_P P_i + \beta_{PC} P_i C_i,$$

where P is `prosoc.left`, a 1/0 variable indicating when the 1/1 option was on the left side. C is `condition`, a 1/0 variable indicating when another animal was present. Note that model \mathcal{M}_3 has a two-way interaction, but it doesn't consider one of the main effects. This is because we are assuming that there is no direct effect of condition on pulling left versus right. You may want to fit a model that considers the main

effect of condition, at the end of this section, so you can see if inferences change any.

To fit these models, just build the linear model of log-odds:

R code
8.3

```
m1 <- mle2( pulled.left ~ dbinom(
  prob=logistic( a + bp*prosoc.left ) , size=1 ) ,
  data=d , start=list(a=0,bp=0) )
m2 <- mle2( pulled.left ~ dbinom(
  prob=logistic( a + bp*prosoc.left + bpC*prosoc.left*condition ) ,
  size=1 ) , data=d , start=list(a=0,bp=0,bpC=0) )
```

And compare the three models:

R code
8.4

```
compare(m0,m1,m2,nobs=nrow(d))
```

k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC	
m1	2	680.5197	688.9409	0.7055	0.8014	0.000000	0.000000
m2	3	682.3874	695.0071	0.2773	0.0386	1.867687	6.066216
m0	1	687.9480	692.1627	0.0172	0.1600	7.428364	3.221772

The model that includes `condition` doesn't do too well, but does get more than 25% of the AICc-implied posterior probability. So let's look at the estimates:

R code
8.5

```
precis(m2)
```

	Estimate	Std. Error	2.5%	97.5%
a	0.04762788	0.1258796	-0.1990916	0.2943474
bp	0.61001213	0.2261956	0.1666769	1.0533473
bpC	-0.10425442	0.2635813	-0.6208642	0.4123554

The estimated effect is negative, with a rather wide posterior on both sides of zero. Doesn't look like the chimpanzees cared much about the other animal's presence. But they do prefer to pull the 1/1 option.

Let's look at the model-averaged predictions, now. Keep in mind that these non-Gaussian models imply a non-additive relationship between the linear model and the predictions. So you must be careful to average the predictions, not the parameters. The kind of code you've learned already will take care of that concern, actually. Here you're going to compose the additive model of the log-odds and then translate it to the probability scale of prediction for averaging, using the `logistic` function.

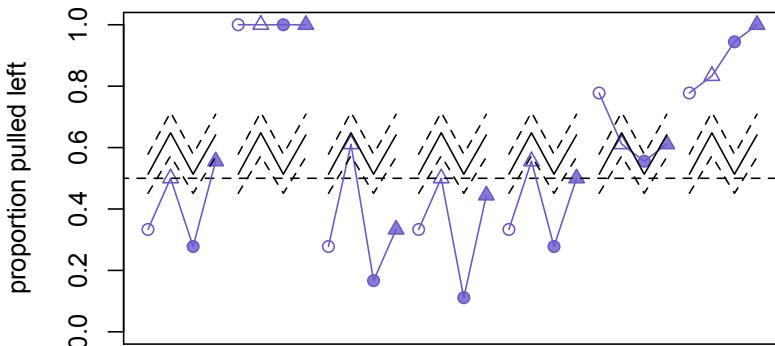


FIGURE 8.3. Model averaged predictions for the chimpanzee data. The blue symbols and lines are the actual data. As in FIGURE 8.2, circles represent pulls when the 1/0 option was on the left, triangles represent pulls when the 1/1 (prosocial) option was on the left, and filled symbols represent pulls when another individual was present (social condition). Chimpanzees did appear to prefer the 1/1 (prosocial) option, but were insensitive to whether another individual was there to benefit from it.

```
post <- sample.naive.posterior( list(m0,m1,m2) , n=30000 ,
  nobs=nrow(d) )
prosoc <- rep( c(0,1,0,1) , 7 )
condit <- rep( c(0,0,1,1) , 7 )
pred.propleft <- sapply( 1:length(prosoc) , function(i)
  mean(logistic(
    post$a + post$bp*prosoc[i] + post$bpC*prosoc[i]*condit[i] )))
pred.propleft.ci <- sapply( 1:length(prosoc) , function(i)
  HPDI(logistic(
    post$a + post$bp*prosoc[i] + post$bpC*prosoc[i]*condit[i] )))
p <- by( d$pulled.left ,
  list(d$prosoc.left,d$condition,d$actor) , mean )
p <- as.vector(p)
plot( p , xlab="" , ylab="proportion pulled left" , ylim=c(0,1) ,
  col="slateblue" , pch=rep(c(1,2,21,24),7) , xaxt="n" ,
```

R code
8.6

```

bg=col.alpha("slateblue",0.8) )
lines( c(-1,length(p)+1) , c(0.5,0.5) , lty=2 )
for ( i in 1:7 ) {
  r <- (i-1)*4 + 1
  x <- r:(r+3)
  y <- p[x]
  lines( x , y , col="slateblue" )
  lines( x , pred.propleft[x] )
  lines( x , pred.propleft.ci[1,x] , lty=2 )
  lines( x , pred.propleft.ci[2,x] , lty=2 )
}

```

FIGURE 8.3. Those are some bad predictions! Nevertheless, no signs that these chimpanzees are responding much to the partner condition.

Would be better to use a multilevel model on these data, because different individuals respond differently and have different preferences for left/right levers. Will leave this important issue until the chapter of multilevel models. For now, let's look at another binomial regression problem, with some similar structure, but in which the data are coded differently.

8.2.4. Example: Graduate school admissions.

UCB admissions data; show how to specify size and calculate nobs.

The data:

R code
8.7

```

library(rethinking)
data(UCBadmit)
d <- UCBadmit

```

These data are famous for demonstrating a phenomenon known in statistics as *Simpson's paradox*. We are going to try to model the influence of an applicant's gender on probability of admission. What these data are also going to allow us to see is how to fit binomial models in which the data are coded with totals instead of 1/0 binary outcomes. In other words, there will be a data column for n now.

First model, with just intercept:

$$n_{\text{admit},i} \sim \text{Binomial}(p_i, n_i),$$

$$\log \frac{p_i}{1 - p_i} = \alpha.$$

```
m0 <- mle2( admit ~ dbinom( prob=logistic(a) , size=applications ) ,
  data=d , start=list(a=0) )
```

R code
8.8

Second model, add gender, m_i :

$$n_{\text{admit},i} \sim \text{Binomial}(p_i, n_i),$$

$$\log \frac{p_i}{1 - p_i} = \alpha + \beta_m m_i.$$

```
d$male <- ifelse( d$applicant.gender=="male" , 1 , 0 )
m1 <- mle2( admit ~ dbinom( prob=logistic(a + bm*male) ,
  size=applications ) ,
  data=d , start=list(a=0,bm=0) )
```

R code
8.9

Compare these two, noting that number of observations relevant for computing AICc is sum of applications column:

```
compare(m0,m1,nobs=sum(d$applications))
```

R code
8.10

k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC	
m1 2	856.5493	857.5165		1	1	0.00000	0.0000
m0 1	947.9970	948.4810	<2e-16	<2e-16	91.44764	90.9645	

Really seems like gender matters! No need to average predictions here, because AICc weights are slam dunk for m1. Look at estimates for m1:

```
precis(m1)
```

R code
8.11

	Estimate	Std. Error	2.5%	97.5%
a	-0.8304821	0.05077166	-0.9299928	-0.7309715
bm	0.6103556	0.06389229	0.4851290	0.7355822

Seems like being male is an advantage. $\exp(0.61) \approx 1.84$ factor change in odds of admission. i.e. male applicant 184% as likely to be admitted as female applicant.

Plot predictions in FIGURE 8.4:

```
post <- sample.naive.posterior(m1)
pred.pr.admit <- sapply( d$male , function(z)
  mean( logistic(post$a + post$bm*z) ) )
pred.pr.admit.ci <- sapply( d$male , function(z)
```

R code
8.12

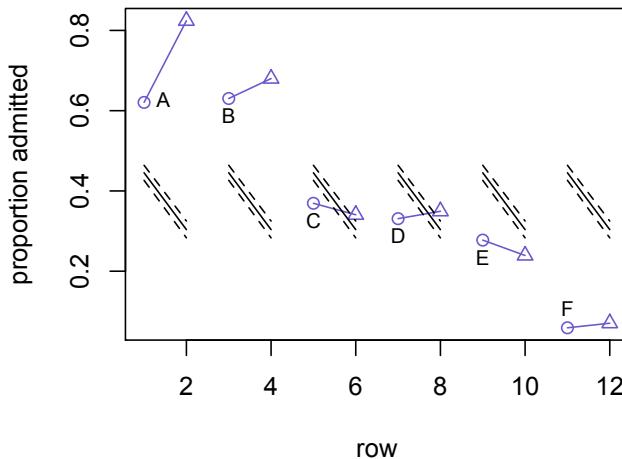


FIGURE 8.4. Predictions for model m1. Blue points and lines are the data, converted to proportions admitted. Lines connect female (circles) and male (triangles) data from the same academic department. Black lines are the predictions arising from m1's naive posterior, with 95% HPDI's shown by the dashed lines.

```

HPDI( logistic(post$a + post$bm*z) )
d$prop.admit <- d$admit/d$applications
plot( d$prop.admit , xlab="row" , ylab="proportion males admitted" ,
      col="slateblue" , pch=ifelse(d$male==1,1,2) )
for( i in 1:6 ) {
  r <- (i-1)*2 + 1
  lines( c(r,r+1) , c(d$prop.admit[r],d$prop.admit[r+1]) ,
         col="slateblue" )
}
lines( 1:12 , pred.pr.admit )
lines( 1:12 , pred.pr.admit.ci[1,] , lty=2 )
lines( 1:12 , pred.pr.admit.ci[2,] , lty=2 )

```

Wow, pretty terrible predictions! There are only two departments in which females had a lower rate of admission than males, and yet the

model says that females should expect to have a 14% lower chance of admission. What has gone wrong here?

Hierarchical data structure. Really want a multilevel model, but that will wait a few more chapters. Will use dummy variable (fixed effects) approach for now, assigning dummies to each department, in order to account for different rates of overall admission among depts.

```
d$deptB <- ifelse( d$dept=="B" , 1 , 0 )
d$deptC <- ifelse( d$dept=="C" , 1 , 0 )
d$deptD <- ifelse( d$dept=="D" , 1 , 0 )
d$deptE <- ifelse( d$dept=="E" , 1 , 0 )
d$deptF <- ifelse( d$dept=="F" , 1 , 0 )
m2 <- mle2( admit ~ dbinom( prob=logistic(
  a + bdB*deptB + bdC*deptC + bdD*deptD + bdE*deptE + bdF*deptF ) ,
  size=applications ) , data=d ,
  start=list(a=0,bdB=0,bdC=0,bdD=0,bdE=0,bdF=0) )
m3 <- mle2( admit ~ dbinom( prob=logistic(
  a + bdB*deptB + bdC*deptC + bdD*deptD + bdE*deptE + bdF*deptF +
  bm*male ) , size=applications ) , data=d ,
  start=list(a=0,bm=0,bdB=0,bdC=0,bdD=0,bdE=0,bdF=0) )
compare(m0,m1,m2,m3,nobs=sum(d$applications))
```

k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC	
m2	6	102.6938	105.5846	0.5591	0.617	0.0000000	0.0000000
m3	7	103.1687	106.5383	0.4409	0.383	0.4749701	0.9536752
m1	2	856.5493	857.5165	<2e-16	<2e-16	753.8555642	751.9318727
m0	1	947.9970	948.4810	<2e-16	<2e-16	845.3032024	842.8963732

Still modest support for some effect of gender, even though best model lacks it. Look at estimates for m3:

```
precis(m3)
```

	Estimate	Std. Error	2.5%	97.5%
a	0.68192242	0.09911021	0.4876700	0.87617486
bm	-0.09987131	0.08084497	-0.2583245	0.05858192
bdB	-0.04339597	0.10983619	-0.2586709	0.17187899
bdC	-1.26259738	0.10663145	-1.4715912	-1.05360358
bdD	-1.29460643	0.10582234	-1.5020144	-1.08719846
bdE	-1.73931087	0.12611231	-1.9864865	-1.49213527
bdF	-3.30648006	0.16998111	-3.6396369	-2.97332320

Effect goes the opposite direction now. $\exp(-0.1) \approx 0.9$, so male odds of admission about 90% of a female applicant's.

Now plot predictions, using model averaging, in FIGURE 8.5:

R code
8.13

R code
8.14

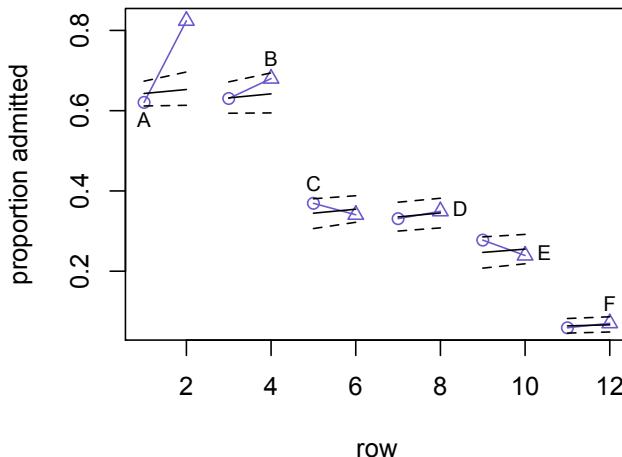


FIGURE 8.5. Model averaged predictions for UCB admissions data.

R code
8.15

```
post <- sample.naive.posterior( list(m0,m1,m2,m3) ,
n=40000 , nobs=sum(d$applications) )
pred.pr.admit <- sapply( 1:nrow(d) , function(i)
mean( logistic(
post$a + post$bm*d$male[i] + post$bdB*d$deptB[i] +
post$bdC*d$deptC[i] + post$bdD*d$deptD[i] +
post$bdE*d$deptE[i] + post$bdF*d$deptF[i] ) ) )
pred.pr.admit.ci <- sapply( 1:nrow(d) , function(i)
HPDI( logistic(
post$a + post$bm*d$male[i] + post$bdB*d$deptB[i] +
post$bdC*d$deptC[i] + post$bdD*d$deptD[i] +
post$bdE*d$deptE[i] + post$bdF*d$deptF[i] ) ) )
plot( d$prop.admit , xlab="row" , ylab="proportion admitted" ,
col="slateblue" , pch=ifelse(d$male==1,1,2) )
for( i in 1:6 ) {
r <- (i-1)*2 + 1
lines( c(r,r+1) , c(d$prop.admit[r],d$prop.admit[r+1]) ,
col="slateblue" )
lines( c(r,r+1) , c(pred.pr.admit[r],pred.pr.admit[r+1]) )
lines( c(r,r+1) ,
```

```

    c(pred.pr.admit.ci[1,r],pred.pr.admit.ci[1,r+1]) , lty=2 )
lines( c(r,r+1) ,
      c(pred.pr.admit.ci[2,r],pred.pr.admit.ci[2,r+1]) , lty=2 )
}

```

This effect arises from heterogeneity in admissions rate by department, along with correlation between gender and department.

8.2.5. Fitting binomial models with `glm`. Gives same results as the `mle2` approach in previous section:

```

m0g <- glm( cbind( admit , reject ) ~ 1 , data=d , family=binomial )
R code
8.16
m1g <- glm( cbind( admit , reject ) ~ male , data=d , family=binomial )
m2g <- glm( cbind( admit , reject ) ~ dept , data=d , family=binomial )
m3g <- glm( cbind( admit , reject ) ~ male + dept , data=d ,
            family=binomial )

```

When outcome coded as 0/1, looks like a linear regression formula:

```

data(chimpanzees)
d2 <- chimpanzees
m <- glm( pulled.left ~ as.factor(actor) + prosoc.left * condition ,
          data=d2 , family=binomial )
R code
8.17

```

8.3. Poisson

Think of it as a filtered exponential with low rate, or just a binomial with low rate.

$$y \sim \text{Poisson}(\lambda).$$

Like the binomial, the Poisson distribution has a “natural” parameter that provides a link function for us:

$$\begin{aligned} y_i &\sim \text{Poisson}(\lambda_i), \\ \log \lambda_i &= \alpha + \beta x_i. \end{aligned}$$

Should I include an explanation of why this is “natural”? Easiest derivation something like:

$$\Pr(y|\lambda) = \frac{\lambda^y \exp(-\lambda)}{y!}.$$

Take the logarithm of the density function, since any exponential family model can be pulled apart this way:

$$\log \Pr(y|\lambda) = y \log(\lambda) - \lambda - \log(y!).$$

To find the “natural” parameter of the distribution, find the coefficient that multiplies the outcome, y . In this case, that is:

$$\log(\lambda).$$

So the logarithm of the mean is the natural parameter of the Poisson. Could just put this derivation in an endnote, instead.

8.3.1. Example: Polynesian tool complexity. Data from Kline and Boyd 2010.⁹⁷ Hypotheses: (1) number of tools increases with population size, (2) number of tools increases with contact rate. Very small data set, so consider hazards of assuming posterior is normal. Will fit again later in book using MCMC and then compare. Also possibly a good RJMCMC example, because will compute reasonably quickly (only 10 cases!).

R code
8.18

```
library(rethinking)
data(islands)
d <- islands
d
```

	Culture	Population	Contact	Total.Tools	Mean.TU
1	Malekula	1100	low	13	3.2
2	Tikopia	1500	low	22	4.7
3	Santa Cruz	3600	low	24	4.0
4	Yap	4791	high	43	5.0
5	Lau Fiji	7400	high	33	5.0
6	Trobriand	8000	high	19	4.0
7	Chuuk	9200	high	40	3.8
8	Manus	13000	low	28	6.6
9	Tonga	17500	high	55	5.4
10	Hawaii	275000	low	71	6.6

And that's the entire data set! You can see the location of these islands in the Pacific Ocean in FIGURE 8.6.

First, we make some new columns with the log of population and a dummy variable for high contact:

R code
8.19

```
d$log.pop <- log(d$Population)
d$contact.high <- ifelse( d$Contact=="high" , 1 , 0 )
```



FIGURE 8.6. Locations of populations in `islands` data. Map reproduced from Kline and Boyd 2010, supplemental.

We'll consider a series of four model families that predict `Total.Tools`: (1) a Poisson model with a constant mean, (2) a model with a mean that depends upon log-population, (3) a model with both log-population and contact rate, and (4) a model that interacts log-population and contact rate.

The most basic Poisson model contains no prediction variables, just a constant mean:

$$n_i \sim \text{Poisson}(\lambda),$$

where n_i is the value of `Total.Tools` on row i and λ is the mean count.

Before fitting this model, though, let's establish the link function to use. We don't technically need a link right now, but it'll be easier to move on to the later models, if we establish the link now. Also, it will help with estimation. It's most common to use a log-link with a Poisson model, such that the mean λ is modeled:

$$\begin{aligned} n_i &\sim \text{Poisson}(\lambda_i), \\ \log \lambda_i &= \alpha. \end{aligned}$$

This log-link is the reason that Poisson models are often called *log-linear models*. The linear model part of the GLM is the logarithm of the observed mean. This actually implies, as you'll see in a bit, that on the real count scale the posited relationship will not be linear.

If you solve the second line above for λ_i , you find that this log-link implies that $\lambda_i = \exp(\alpha)$. We really do need a link here, otherwise the mean will be allowed to go below zero. If the linear model is exponentiated, as it is here, this guarantees that λ will be positive, even if the linear model itself is negative. Go ahead and try it, if you don't immediately intuit this truth.

Now here's the code to fit the model:

R code
8.20

```
m0 <- mle2( Total.Tools ~ dpois( lambda=exp(a) ) , data=d ,
  start=list(a=log(mean(d$Total.Tools))) )
precis(m0)
```

	Estimate	Std. Error	2.5%	97.5%
a	3.549617	0.05360563	3.444552	3.654682

The first thing to do is remember the link function. The predicted mean is $\exp(\alpha)$, so:

R code
8.21

```
exp( coef(m0)[1] )
```

a	34.8
---	------

And by no small coincidence, this is the same value you get by just empirically averaging the `Total.Tools` column.

Adding a predictor is as you'd expect by now, just add terms to the second line. For example, here is the second model we'd like to fit, using log-population as a predictor:

$$n_i \sim \text{Poisson}(\lambda_i), \\ \log \lambda_i = \alpha + \beta_P \log P_i.$$

And the code to fit it:

R code
8.22

```
m1 <- mle2( Total.Tools ~ dpois( lambda=exp(a + bp*log.pop) ) ,
  data=d , start=list(a=log(mean(d$Total.Tools)),bp=0) )
```

And similarly, the final two models:

R code
8.23

```
m2 <- mle2( Total.Tools ~ dpois(
  lambda=exp(a + bp*log.pop + bc*contact.high) ) ,
  data=d , start=list(a=log(mean(d$Total.Tools)),bp=0,bc=0) )
m3 <- mle2( Total.Tools ~ dpois(
  lambda=exp(a + bp*log.pop + bc*contact.high
  + bcp*log.pop*contact.high) ) ,
  data=d ,
  start=list(a=log(mean(d$Total.Tools)),bp=0,bc=0,bcp=0) )
```

Now to compare the models. First, here are the rankings and weights:

```
compare(m0,m1,m2,m3,nobs=nrow(d))
```

R code
8.24

k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC
m2 3	81.04939	77.95715	0.76328	0.69574	0.000000	0.000000
m1 2	83.77255	82.66344	0.19559	0.06614	2.723162	4.706291
m3 4	86.89127	80.10161	0.04113	0.23811	5.841877	2.144462
m0 1	133.91642	133.71900	2.528e-12	5.419e-13	52.867026	55.761856

Keeping in mind that both AICc and BIC are large-sample approximations, these rankings suggest that at least log-population is a useful predictor, and possibly contact rate as well. BIC in particular likes the interaction model, m3, a lot more than AICc does. In a sample as small as this, AICc is much more nervous about overfitting.

First, you can inspect the estimates and see if the coefficients are in the predicted directions. Comparing estimates across models:

```
coeftab(m0,m1,m2,m3)
```

R code
8.25

	m0	m1	m2	m3
a	3.5496	1.3290	0.9243	0.9535
bp	NA	0.2397	0.2660	0.2632
bc	NA	NA	0.2984	-0.3249
bcp	NA	NA	NA	0.0681
nobs	10.0000	10.0000	10.0000	10.0000

In models m1 and m2, the main effects of both log-population and contact rate are consistent with the hypotheses: they are both positive, being associated with more tools. Interpreting m3 is harder, because we didn't center the predictors before fitting (remember centering, from Chapter 6). But the interaction estimate, at least, can be interpreted. It is positive, as it should be if high contact rate and population size act synergistically. But the size of the estimate is very small, and if you inspect the standard error on it, you'll see it's very uncertain. This is partially why the model comparison using AICc or BIC does not much like m3.

To get a sense for what's being predicted, let's plot the model predictions, as usual. There are no new coding tricks here, aside from the detail of how to convert the linear model into the prediction scale. To illustrate that, I'm going to produce two plots. In the first, I'll draw the mean and 95% confidence interval of the mean, using the usual lines. In the second, I'll plot Poisson random samples from the posterior, so you can see how to simulate observations from a fit Poisson model.

To calculate predicted mean counts, you just need the usual `sapply` code, now with the inverse link function embedded. This is exactly

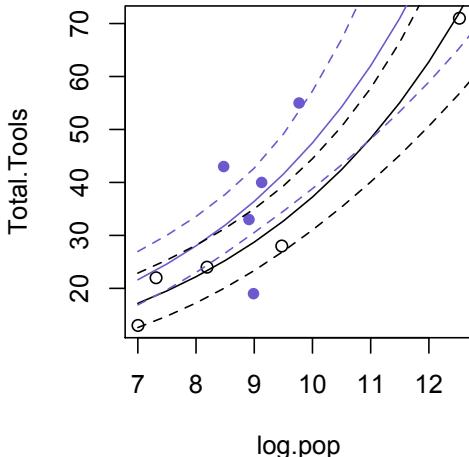


FIGURE 8.7. Predicted mean tool counts and 95% confidence intervals of the mean, for both low contact rate islands (open circles and black lines) and high contact rate islands (filled blue circles and blue lines).

what you did when you plotted the binomial models. But now the link function isn't the logit, but rather the logarithm. So this code will compute the predicted mean and 95% confidence interval of the mean, for low contact and high contact islands, respectively:

R code
8.26

```
post <- sample.naive.posterior( list(m1,m2,m3) , n=30000 ,
                                nobs=nrow(d) , method="AICc" )
pop.seq <- seq(from=7,to=13,by=0.5)
mu.low <- sapply( pop.seq , function(z)
                  mean( exp( post$a + post$bp*z + post$bc*0 + post$bcp*z*0 ) ) )
mu.hi <- sapply( pop.seq , function(z)
                  mean( exp( post$a + post$bp*z + post$bc*1 + post$bcp*z*1 ) ) )
mu.ci.low <- sapply( pop.seq , function(z)
                  PCI( exp( post$a + post$bp*z + post$bc*0 + post$bcp*z*0 ) ) )
mu.ci.hi <- sapply( pop.seq , function(z)
                  PCI( exp( post$a + post$bp*z + post$bc*1 + post$bcp*z*1 ) ) )
```

Since the mean of the Poisson distribution is just λ , then to compute the mean, you only need to exponentiate the linear model. When there is log-link, the inverse is exponential.

Plotting these computations with the usual `lines` commands, the result is shown in FIGURE 8.7. The filled points are islands with high contact rate, while the empty points are islands with low contact rate. The black lines show predictions for low-contact islands, while the black

lines show predictions for the high contact islands. While the trend with increasing population size is quite strong, there is a lot of overlap between the high and low contact rate predictions. So while model comparison supports using contact rate to improve prediction, it isn't half as important as log-population size.

Now for simulating predictions from the model. The function you want now is `rpois`, which produces random Poisson-distributed values. We'll embed this function in `sapply`, just as you embedded `rnorm` in earlier chapters. Here's the code to produce 500 random predictions for both low and high contact rate islands:

```
npts <- 500
xpts <- runif( npts , min=min(d$log.pop) , max=max(d$log.pop) )
pred.TT0 <- rpois( npts ,
  lambda=exp( post$a + post$bp*xpts + post$bc*0 + post$bcp*xpts*0 ) )
pred.TT1 <- rpois( npts ,
  lambda=exp( post$a + post$bp*xpts + post$bc*1 + post$bcp*xpts*1 ) )
```

R code
8.27

The function `rpois` also needs the exponentiated linear model, because it wants the value of λ . Each simulated prediction here is a tool count for an imaginary island, as expected from the posterior.

In FIGURE 8.8, I show these simulated island tool counts. In both plots in this figure, the open black circles are predicted counts for low contact rate, while the filled blue circles are for high contact rate. The plot on the left represents predicted counts for just model `m2`, the model with only main effects of both log-pop and contact rate. This model shows a visible separation between open and filled predictions: high contact islands tend to have higher counts, although there is still considerable overlap. On the right, the model-averaged predictions show practically no visible separation at all. Indeed, there is less visible separation in these actual count predictions than there was in FIGURE 8.7, which showed only predicted means, not actual counts.

8.4. Multinomial

Example: Time allocation data? DG/UG/PGG data?

$$y_i \sim \text{Multinomial}(n, p_{1i}, p_{2i}, \dots, p_{mi})$$

Basic notion: Each category gets a score, and that score is used to generate an unordered probability distribution, through standardization. So if we have m categories in the outcome distribution, then probability

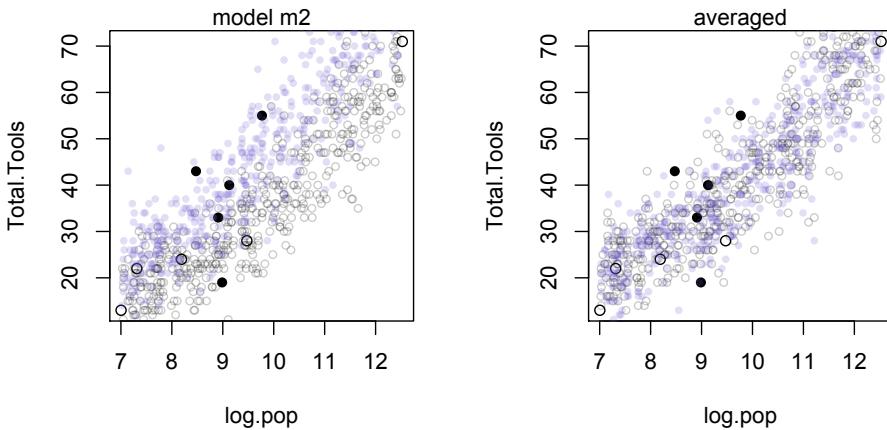


FIGURE 8.8. Predicted tool counts, with low contact rate predictions shown by open black points and high contact rate shown by filled blue points. The actual island data is shown by the larger points. On the left, predictions for only model $m2$ show a clear separation between high and low contact rate islands. On the right, the model averaged predictions are less confident about the importance of contact rate, and little separation appears between the filled and open points.

of observing category $y_i = k$ is:

$$\Pr(y_i = k) = \frac{\exp(\phi_{ki})}{\sum_j \exp(\phi_{kj})}$$

9 Generalized Linear Models III: Monsters and Mixtures

9.1. The Golem

Intro idea: Statistical models are like the golem of folklore: they are human constructions, built from pieces with no life of their own, to form dynamic, useful, and often unpredictable tools. Like the golem, complex models can perform tasks that humans cannot. Also like the golem, they are sometimes foolish, often misunderstood, and dangerous when mistreated. This chapter is about building more complex statistical models, by piecing together the simpler tools of previous chapters. The tension is always between the power we derive from the models and the dangers we face when we misunderstand or misapply them. Inference becomes simultaneously harder, riskier, and potentially more powerful.

Monsters are models that are pieced together from simpler, often exponential family, pieces to form specialized outcome distributions. Ordered categories and ranks are very common kinds of data that don't really suit any of the previous distributions. These kinds of data need their own, somewhat monstrous, probability densities.

Mixtures are models that comprise multiple stochastic processes, usually embedded within one another. Measurement error and other stochastic processes can be blended with the outcome distributions you've already seen to produce models that can estimate and cope with the outcomes that depend upon more than one stochastic process.

9.2. Ordered Categorical Outcomes

Describe why we need a density for ordered categorical outcomes. Show examples. Very common in social sciences. In natural sciences, appears in ecological field data, for example.

9.2.1. An ordered probability density. The probability distribution we seek here will be born of stating how we'd like to be able to interpret it. The main criterion is that we'd like to operate with *log-odds*. Recall from

the last chapter, when you met binomial GLM's, that the log-odds of any event E is just:

$$\log \frac{\Pr(E)}{1 - \Pr(E)}.$$

This is simply the logarithm of the odds. Back there (page 347), I explained the advantages of modeling the log-odds as an additive combination of parameters and variables. All of those reasons matter here, in the same ways.

So we want to work with log-odds, again. But there's also another desirable property of log-odds, which will help us now that we have, in effect, a multinomial distribution with more than two mutually exclusive outcomes: We should focus on the *cumulative* log-odds of each possible value. Let me explain why. The probability density we are after has a number of discrete, ordered observable values. Our goal is to get a formula—a probability density—that tells us the likelihood of each value. This function will be like the `dnorm` and `dbinom` and `dpois` of earlier GLM's. It will help us in getting to this goal if we start with *cumulative* probability, though, instead of the discrete probability of each observable value. Cumulative probability is just the probability of any value or *smaller value*. Why go cumulative at the start here? Because cumulative probability naturally constrains itself to never exceeding a total probability of one. And because this is an ordered density, we know that the cumulative log-odds of the largest observable value must be $+\infty$, which is the same as cumulative probability of one. This anchors the distribution and standardizes it at the same time. If you start instead with discrete individual probabilities of each outcome, then you'd have to later *standardize* these probabilities to ensure they sum to exactly one. It turns out to be easier to just start with the cumulative probability and then work backwards to the individual probabilities. I know, this seems weird. But I'll walk you through it.⁹⁸

What we want is for the *cumulative* log-odds of an observed value y_i to be equal-to-or-less-than some possible value k to be:

$$\log \frac{\Pr(y_i \leq k)}{1 - \Pr(y_i \leq k)} = \phi_k, \quad (9.1)$$

where ϕ_k is a continuous value, different for each observable value k . We'll make this value into a linear model, in a bit. For now, it's just a placeholder. The above function is just a direct embodiment of the log-odds and cumulative density objectives we've stated so far. It actually says nothing else at all. Now we solve for the cumulative probability

density itself. Do this by taking (9.1) and solving for $\Pr(y_i \leq k)$. After a little algebra, you get:

$$\Pr(y_i \leq k) = \frac{\exp(\phi_k)}{1 + \exp(\phi_k)}.$$

You might recognize this probability as the *logistic*, same as in the last chapter. It arose in the same way, establishing the logistic function as the inverse link for the binomial model. But now we have a *cumulative logistic*, since the probability $\Pr(y_i \leq k)$ is cumulative.

But we still need likelihoods, which are not cumulative. So how do you use this thing? Well, it's a probability density, so you can use it to define the likelihood of any observation y_i . By definition, in a discrete probability density, the likelihood of any observation $y_i = k$ must be:

$$\Pr(y_i = k) = \Pr(y_i \leq k) - \Pr(y_i \leq k - 1). \quad (9.2)$$

This just says that since the logistic is cumulative, we can compute the discrete probability of exactly $y_i = k$ by subtracting the cumulative probability of one observable value lower than k .

9.2.2. Putting the GLM in the ϕ . We're almost ready to walk through actually writing the code to fit models to this distribution. But before we get to coding, there is one more conceptual step. To build additive models of the log-odds values ϕ_k , we'll need to distinguish the intercept at each value k from the rest of the additive model.

In the simplest case, there are no predictor variables, and so each observable value k has a unique log-odds value ϕ_k that just translates between the cumulative probability scale and the log-odds scale of the model. For example, consider a case in which the observable values are the numbers 1, 2 and 3. Now the maximum likelihood estimate of the cumulative probability of observing a "1" is just going to be the observed proportion of 1's in the data. The estimate of the cumulative probability of a 2-or-less is going to be the proportion of the observed values that are 2 or 1. The cumulative probability of a 3 must be exactly 1, because it's the maximum value. So if there are, say, 100 observations, and 31 of them are "1" and 49 of them are "2", then the estimated cumulative probabilities of 1, 2 and 3 are: 31/100, (31+49)/100, 1. Crunched into proportions, those values are : 0.31, 0.8, and 1. Now the ϕ value corresponding to each is just the cumulative log-odds of each. So you can compute them directly from these proportions, just by using the formula for log-odds. For example, the log-odds for "2" must be:

$$\phi_2 = \log \frac{0.8}{1 - 0.8} \approx 1.39.$$

That is what I mean when I say that each ϕ_k value translates between the probability and log-odds scales. If you know one, you can compute the other.

But we need a way to allow the distribution to change, as a result of predictor variables. This is where the generalized linear model comes back. Think of the example just above as a GLM in which there are only *intercepts*, no predictor variables. This is akin to a linear regression with only a single parameter, α , in the model of μ_i . In this case, however, there has to be a unique intercept for each observable value (1, 2 or 3, in the example). Otherwise, the distribution wouldn't be sufficiently flexible. Recall, we're resorting to this ordered categorical density, because we don't have any basis to say that the "distance" between each observable value is the same. All these unique intercepts do is take an arbitrary cumulative probability density and translate it to cumulative log-odds.

So how do we get predictor variables and β coefficients back in here? Just add them to ϕ_k . But while there still needs to be a unique intercept for each observable value k , the rest of the additive model will be common to all values ϕ_k . In effect, this means pulling out a unique intercept parameter for each observable value, and then adding a common additive model to each. So now the cumulative log-odds of any value k will be defined as:

$$\log \frac{\Pr(y_i \leq k)}{1 - \Pr(y_i \leq k)} = \alpha_k + \phi_i.$$

In turn, ϕ_i is an additive model that may contain slope parameters and predictor variables. It's functionally just like μ_i , from all of those Gaussian models you fit in earlier chapters.

What these assumptions do is allow the distribution of observable values to vary as a consequence of the predictor variables inside ϕ_i . This will make more sense, once you fit one of these models and plot its implied predictions. I'll walk you through that. But first, we need to actually code the density function that will make estimating the naive posterior possible.

9.2.3. Coding the ordered logistic density. We have our target density now, and we have a way to get an additive model into it. How do we make R compute the likelihoods we need, in order to estimate the naive posterior? Here, we'll write our first completely custom density function. A density function is a block of code that takes observed values and parameters as inputs and returns likelihood (or log-likelihood)

corresponding to those inputs. There's nothing stopping you from writing your own density functions. Then you can use them in `mle2` or any other context, such as the MCMC algorithms later in the book.

You do, however, have to obey a small number of conventions, in writing a density function. First, the first parameter must be a vector of values to compute likelihoods for, and this vector must be named `x`, just because R expects it to have this name. Second, there must be a parameter named `log`, and when it is set to `TRUE`, the function should return log-likelihood, rather than ordinary likelihood. Ideally, the function will return one (log-)likelihood for each element of the input `x`. Other parameters defined in the function are for the parameters of the likelihood function, the μ 's and σ 's and such. You can name those whatever you want.

Let's write the density function for the ordered categorical distribution. This function is already built into the `rethinking` package, but it's worth examining it, as an example of how build your own density function. I'll present a working function definition here, and then I'll step through its pieces and explain the important parts.

```
dordlogit <- function( x , a , phi , log=FALSE ) {
  a <- c(as.numeric(a), Inf)
  p <- logistic( a[x] - phi )
  na <- c( -Inf , a )
  np <- logistic( na[x] - phi )
  p <- p - np
  if ( log==TRUE ) p <- log(p)
  p
}
```

R code
9.1

This function has four parameters. The first, `x`, is just a list of observed values to produce likelihoods for. The second is a vector of intercepts, `a`. This is a list of cumulative log-odds of the α_k values defined in the previous section. Then there is the additive model of predictors and slopes, `phi`. Finally, the required `log` parameter, telling R whether or not to return log-likelihood, with a default value of `FALSE`. Inside the function, the first line just inserts the always-known final log-odds value, infinity or `Inf`. We'll never have to estimate this intercept, so the likelihood function supplies it automatically. The second line computes the logistic (cumulative probability) of each observation, using the supplied intercepts `a` and additive model `phi`. To get likelihoods, though, we have to take each cumulative probability and subtract from it the cumulative probability of a value one unit lower, as in Equation 9.2 (page 371). So

the second and third lines of code compute the cumulative probabilities for values one lower than the observed ones. The fourth line of code then does the subtraction seen in Equation 9.2. The last two lines just apply the optional log transform and return the vector of likelihoods (or log-likelihoods).

Whew. Now, don't be discouraged, if you feel like you couldn't possibly have written that yourself, from scratch. That's okay. You get good at this stuff by seeing many examples of working code. This will populate your mind with the tools you'll need to modify and build your own solutions. It takes time. This process is much like learning a natural human language. At first, you study vocabulary and phrases. These tools allow you to improvise, but with error. Eventually, you become skilled at recombining and embedding the tools with one another, to produce novel and moving utterances. Scientific programming is similar. No one gets this stuff right away.

So how do you use this new density function? Well, think back to a simple Gaussian model for a moment. If you have a list of Gaussian observations, and you want to compute their likelihood, you just pass the list of them to the function `dnorm`. Here's an example, with arbitrary values:

R code
9.2

```
y <- c( -1 , 0 , 1 )
dnorm( y , mean=0 , sd=2 , log=FALSE )
```

```
[1] 0.1760327 0.1994711 0.1760327
```

Those are the likelihoods of the values $-1, 0, 1$, assuming they are from a Gaussian distribution with mean zero and standard deviation 2. Similarly, we can invent some ordered categorical observations and compute their likelihood:

R code
9.3

```
y <- 1:3
ak <- c( -1 , 1 )
dordlogit( y , a=ak , phi=0 , log=FALSE )
```

```
[1] 0.2689414 0.4621172 0.2689414
```

This example assumes only three observable values: 1, 2 and 3. The vector `ak` contains the assumed log-odds intercepts for each observable value below the maximum value of 3. We already know the log-odds of the maximum observable value: ∞ . So you don't have to supply it. I made the additive model `phi` zero in this example. But you could provide any value or list of values (one for each element of `y`).

The real purpose of this function is to embed it in an estimation algorithm, so you can get maximum likelihood estimates and confidence intervals from it. It'll also let us compute AICc for a fit model. Let's explore its intended use in the context of a real data example.

9.2.4. Example: Moral intuition. The data for this example come from a series of experiments conducted by philosophers, lead by Cushman.⁹⁹ Yes, philosophers do sometimes conduct experiments. In this case, the experiments aim to collect empirical evidence relevant to debates about *moral intuition*, the forms of reasoning through which people develop judgments about the moral goodness and badness of actions. These debates are relevant to all of the social sciences, because they touch on broader issues of reasoning, the role of emotions in decision making, and the theories of moral development, both in individuals and groups.

Specifically, these experiments address scenarios sometimes known as “trolley problems” that have proved vexing or paradoxical to moral philosophers. The reason these scenarios can be vexing is that the analytical context of them can be identical, and yet people reliably reach different judgments about the goodness or badness of the actions. Previous research has lead to at least three important principles of unconscious reasoning that may explain variations in judgment. These principles are:

The action principle: Harm caused by action is morally worse than equivalent harm caused by omission.

The intention principle: Harm intended as the means to a goal is morally worse than equivalent harm foreseen as the side effect of a goal.

The contact principle: Using physical contact to cause harm to a victim is morally worse than causing equivalent harm to a victim without using physical contact.

The experimental context we'll explore these principles within comprises stories that vary these principles, while keeping many of the basic objects and actors the same. For example, the most renown story is about a speeding boxcar. Here is the version of boxcar story that implies the action principle, but not the others:

Standing by the railroad tracks, Dennis sees an empty, out-of-control boxcar about to hit five people. Next to Dennis is a lever that can be pulled, sending the boxcar down a side track and away from the five people. But pulling the lever will also lower the railing on a footbridge spanning the side track, causing one person to fall off the footbridge and onto the side track, where he will be hit by the boxcar. If Dennis pulls the lever the

boxcar will switch tracks and not hit the five people, and the one person to fall and be hit by the boxcar. If Dennis does not pull the lever the boxcar will continue down the tracks and hit five people, and the one person will remain safe above the side track.

Since the actor (Dennis) had to do something to create the outcome, rather than remain passive, this is an action scenario. However, the harm caused to the one man who will fall is not necessary, or intended, in order to save the five. Thus it is not an example of the intention principle. And there is no direct contact, so it is also not an example of the contact principle. Now, how morally permissible is it to pull the lever? The data we'll be interested in predicting are responses to that question, rated on a scale from 1 (never permissible) to 7 (always permissible).

You can contrast the above story with the same outline, but now with both the action principle and the intention principle. That is, in this version, the actor both does something to change the outcome and the action must cause harm to the one person in order to save the other five:

Standing by the railroad tracks, Evan sees an empty, out-of-control boxcar about to hit five people. Next to Evan is a lever that can be pulled, lowering the railing on a footbridge that spans the main track, and causing one person to fall off the footbridge and onto the main track, where he will be hit by the boxcar. The boxcar will slow down because of the one person, therefore preventing the five from being hit. If Evan pulls the lever the one person will fall and be hit by the boxcar, and therefore the boxcar will slow down and not hit the five people. If Evan does not pull the lever the boxcar will continue down the tracks and hit the five people, and the one person will remain safe above the main track.

Most people judge that, if Even pulls the lever, it is morally worse (less permissible) than when Dennis pulls the lever. You'll see by how much, as we analyze these data.

Load the data with:

R code
9.4

```
library(rethinking)
data(trolley)
d <- trolley
```

There are 12 columns and 9930 rows, comprising data for 331 unique individuals. The outcome we'll be interested in is `response`, which is an integer from 1 to 7 indicating how morally permissible the participant

found the action taken (or not taken) in the story. Since this type of rating is categorical and ordered, it's exactly the type of problem to apply our ordered logit model to.

The predictor variables of interest are going to be `action`, `intention`, and `contact`, each a dummy variable corresponding to each principle outlined above.

9.2.4.1. The basic model. To fit the basic model, incorporating no predictor variables:

```
library(bbmle)
m0 <- mle2( response ~ dordlogit( a=c(a1,a2,a3,a4,a5,a6) , phi=0 ) ,
  data=d , start=list(a1=-2,a2=-1,a3=0,a4=1,a5=2,a6=2.5) )
```

R code
9.5

Take a look at the parameter estimates, now. They are just log-odds estimates for the cumulative probability of each value. Again, there is no estimate for the value "7", because the cumulative probability of the maximum value must be 1. I'm going to hold off on plotting the predictions from this model, until we have models with predictor variables to compare it to.

Before moving on to adding predictor variables to the model, it's worth commenting on how to get reasonable starting values for the intercepts. In the example here, when you fit `m0`, I just guessed at some starting values, and it turned out okay. But you won't always get so lucky. Now that we're working with non-linear models, maximum likelihood sometimes needs a helping hand to get to the global maximum. If you accidentally pick poor starting values for the parameters, you can get either non-sensical estimates or R may fail to find any likely estimates at all. This can be very frustrating.

A good method for picking starting values for the intercepts in an ordered categorical model of this kind is to just use the log-odds definition to find them. Here's an example, in which I compute the implied log-odds of each observable value, using the data.

```
logodds <- function( x ) log(x/(1-x))
cumu.p <- sapply( 1:7 , function(z)
  length( d$response[ d$response<=z ] )/nrow(d) )
ak <- logodds( cumu.p )
```

R code
9.6

The values in the vector `ak` are now your candidate starting values. Compare the values in `ak` to the maximum likelihood estimates you found in `m0` above. You'll see they are very similar. If you use straight

empirical log-odds like this to get starting values, it can help the maximum likelihood search a lot.

Later in this chapter, I am going to explain how to fit this kind of model with convenience functions like `polr` (in the `MASS` library). This function actually does the above for you. But as always in this book, I labor to explain what is actually going on, because eventually you'll need to solve similar problems for yourself. And once you start using MCMC estimation, you're going to have pick starting values for yourself, in most cases.

9.2.4.2. Adding predictor variables. So far, all this agonizing about log-odds hasn't bought us a lot. Sure, by defining a likelihood function, we have been able to apply Bayes' theorem, channeled through the blunt instrument of maximum likelihood estimation. This has gotten us more than just the empirical shape of the distribution, which we could have just plotted from the raw data, because it has bought us a way to quantify uncertainty. And that's actually very valuable.

But still, what we'd really like to do is model how the distribution of these ordered categorical responses varies as a function of some predictor variables. In this data example, we're interested in how the `action`, `intention`, and `contact` codes predict differences in rated permissibility. So we need to get these variables into the GLM now.

Here's how we'll do it. The model so far is just:

$$\log \frac{\Pr(y_i \leq k)}{1 - \Pr(y_i \leq k)} = \alpha_k.$$

Or to return to our stochastic node notation, you might write:

$$y_i \sim \text{ordlogit}(p_1, p_2, p_3, \dots, p_7),$$

$$\log \frac{p_{y_i}}{1 - p_{y_i}} = \alpha_{y_i}.$$

This just says that each observed value y_i is distributed as an ordered logistic variable, with cumulative probabilities of each observable value $1 \dots n$ given by p_1 through p_7 . The log-odds of each cumulative probability are in turn are just defined by intercept values $\alpha_1 \dots \alpha_7$, one unique to each possible value of y_i . And as before, we know that $\alpha_7 = \infty$, based upon the constraint that a probability density always sums to one.

So in true GLM fashion, now we expand that definition of the log-odds to include some slope parameters and predictor variables. To add

the dummy variables `action`, `intention`, and `contact` to the model:

$$y_i \sim \text{ordlogit}(p_1, p_2, p_3, \dots, p_7),$$

$$\log \frac{p_{y_i}}{1 - p_{y_i}} = \alpha_{y_i} - (\beta_A A_i + \beta_I I_i + \beta_C C_i),$$

where A_i indicates the value of `action` on row i , I_i indicates the value of `intention` on row i , and C_i indicates the value of `contact` on row i . What we've done here is define the log-odds of each possible response to be an additive model of the features of the story corresponding to each response.

Why did I subtract the new additive portion from the intercept? If you look back at the code for `dordlogit`, you'll see that I've already snuck this assumption past you once. The reason for doing this is to make the parameter estimates easier to interpret, later. We would like a positive estimate for, say, β_A to indicate an increase in the average value of y_i , when A_i increases. But in order to increase the average y_i , we need to bunch up probability on the high end of the distribution. This corresponds to *reducing* the cumulative log-odds of every possible outcome value, leaving the remaining probability mass at the high end. I know this is weird. It will make sense once we estimate this model and plot its predictions, across changes in the predictor variables. So bear with me.

You fit this model just as you'd expect, by adding the slopes and predictor variables to the `phi` parameter inside `dordlogit`. Here's a working model:

```
m1 <- mle2( response ~ dordlogit( a=c(a1,a2,a3,a4,a5,a6) ,
  phi=bA*action + bI*intention + bC*contact ) , data=d ,
  start=list(a1=-1.9,a2=-1.2,a3=-0.7,a4=0.2,a5=0.9,a6=1.8,
  bA=0,bI=0,bC=0) )
```

R code
9.7

The parameter `phi` now contains the additive function with slope parameters and predictor variables. Notice also that I've adopted the approximate maximum likelihood estimates from the previous model, `m0`, as starting values for the intercepts. This helps `mle2` find the new maximum likelihood estimates more quickly.

Let's fit one more model of interest, before comparing them and turning to plotting model-averaged predictions. It's possible that the variables `action` and `intention` interact. This would mean that a story

containing both action and intention is non-additively worse (or better) than what you'd expect by just adding together the separate estimates for `action` and `intention`. It's just as possible that `contact` and `intention` interact. The variables `action` and `contact` cannot interact, because `contact` is just a type of `action`, in these data. But that leaves us with two interaction effects to model.

Fitting the interaction model follows the pattern you are accustomed to by now.

R code
9.8

```
m2 <- mle2( response ~ dordlogit( a=c(a1,a2,a3,a4,a5,a6) ,
  phi=bA*action + bI*intention + bC*contact
  + bAI*action*intention + bCI*contact*intention ) ,
  data=d ,
  start=list(a1=-1.9,a2=-1.2,a3=-0.7,a4=0.2,a5=0.9,a6=1.8,
  bA=0,bI=0,bC=0,bAI=0,bCI=0) )
```

No new tricks here. The above just adds two interaction terms and two interaction parameters, `bAI` and `bCI`.

Now let's compare these three models. You can use `coeftab` to get a quick comparison of both estimates and AICc values, by using the optional `compare=TRUE` parameter:

R code
9.9

```
coeftab(m0,m1,m2,compare=TRUE)
```

	m0	m1	m2
a1	-1.916100e+00	-2.837400e+00	-2.6349
a2	-1.266600e+00	-2.155500e+00	-1.9388
a3	-7.186000e-01	-1.573000e+00	-1.3435
a4	2.478000e-01	-5.515000e-01	-0.3079
a5	8.899000e-01	1.166000e-01	0.3631
a6	1.769400e+00	1.023200e+00	1.2691
bA	NA	-7.092000e-01	-0.4724
bI	NA	-7.205000e-01	-0.2827
bC	NA	-9.615000e-01	-0.3316
bAI	NA	NA	-0.4464
bCI	NA	NA	-1.2727
nobs	9.930000e+03	9.930000e+03	9930.0000
AICc	3.785446e+04	3.708991e+04	36929.2057
weight	1.213946e-201	1.269081e-35	1.0000

We'll get to AICc-implied posterior probability estimates in a bit. First, whatever do these estimates mean? The first six rows, from `a1` to `a6`, are just the α intercepts, one for each value below the maximum of "7".

These really can't be interpreted on their own, unless you are very used to reading log-odds values.

The next 5 rows, from `bA` to `bCI`, are the various slope parameters: three main effects and two interactions. These are interpretable on their own, to a limited extent. It makes sense to ask, first, if they are very far from zero. You can check the standard errors and 95% confidence intervals with `precis` and verify that all of the slope estimates are quite reliably negative. Second, all of the slopes are negative, which implies that each factor/interaction *reduces* the average response. Including action, intention or contact in a story leads people to judge it as less morally permissible. But by how much? Remember, these parameters are part of a function defining cumulative log-odds, so they can be interpreted as changes in cumulative log-odds. But unless you are very comfortable thinking about log-odds and cumulative probability densities, that doesn't help you much. It also doesn't help that this change applies to the cumulative log-odds of *every* value of the response variable, aside from the maximum one (which is fixed at cumulative log-odds ∞).

So what to do? When in doubt, plot the model's predictions. Now, since model `m2` absolutely dominates based upon AICc—it gets very nearly 100% of the posterior probability estimate—we can safely proceed here, ignoring model averaging. But if the AICc's had turned out to be less decisive, the only thing you'd change in the code that follows is passing a list of models to `sample.naive.posterior` and then making sure to plot enough samples from the posterior to represent the influence of low-probability models.

There is no perfect way to plot the predictions of these cumulative log-odds models. But a common and useful way is to use the horizontal axis for a predictor variable and the vertical axis for cumulative probability. Then you can plot a curve for each response value, as it changes across values of the predictor variable. After plotting a curve for each response value, you'll end up mapping the distribution of responses, as it changes across values of the predictor variable.

So let's do that. First, as usual, sample from the naive posterior:

```
post <- sample.naive.posterior( m2 )
```

R code
9.10

Now make an empty plot:

```
plot( 1 , 1 , type="n" , xlab="intention" , ylab="probability" ,
      xlim=c(0,1) , ylim=c(0,1) , xaxp=c(0,1,1) , yaxp=c(0,1,2) )
```

R code
9.11

Now loop over the first five-hundred samples in `post` and plot their predictions, across values of `intention`:

R code
9.12

```
kA <- 0
kC <- 1
KI <- 0:1
for ( s in 1:500 ) {
  p <- post[s,]
  ak <- as.numeric(p[1:6])
  phi <- p$bA*kA + p$bI*kI + p$bC*kC
    + p$bAI*kA*kI + p$bCI*kC*kI
  pk <- pordlogit( 1:6 , a=ak , phi=phi )
  for ( i in 1:6 )
    lines( KI , pk[,i] , col=col.alpha("slateblue",0.01) )
}
```

The first three lines in the code above define the values of `action` (`kA`), `intention` (`KI`), and `contact` (`kC`) to use for the calculations and plotting. The value assigned to `KI` is a vector, because that is the variable we are varying on the horizontal axis. If it had more values than 0 and 1, then more values would appear in this vector. Then the code loops over the first 500 samples from the naive posterior, computing cumulative probabilities for each response value, for each sample. The function `pordlogit` computes cumulative ordered logit probabilities, just as `dordlogit` computes likelihoods. Finally, each response boundary is plotted, using `lines`.

I know this plotting code is complicated, compared to previous chapters. But as the models become more monstrous, so to does the code needed to compute predicts and display them. With power comes hardship. Beware the Golem.

Finally, plot the maximum likelihood predictions, so it's easier to see where the center of each boundary is located:

R code
9.13

```
pmle <- as.list(coef(m2))
ak <- as.numeric(pmle[1:6])
phi <- pmle$bA*kA + pmle$bI*kI + pmle$bC*kC
  + pmle$bAI*kA*kI + pmle$bCI*kC*kI
pk.mle <- pordlogit( 1:6 , a=ak , phi=phi )
for ( i in 1:6 ) lines( 0:1 , pk.mle[,i] , col="white" )
```

Results in [FIGURE 9.1](#). In each plot, the blue lines indicate the boundaries between response values, numbered 1 through 7, bottom to top.

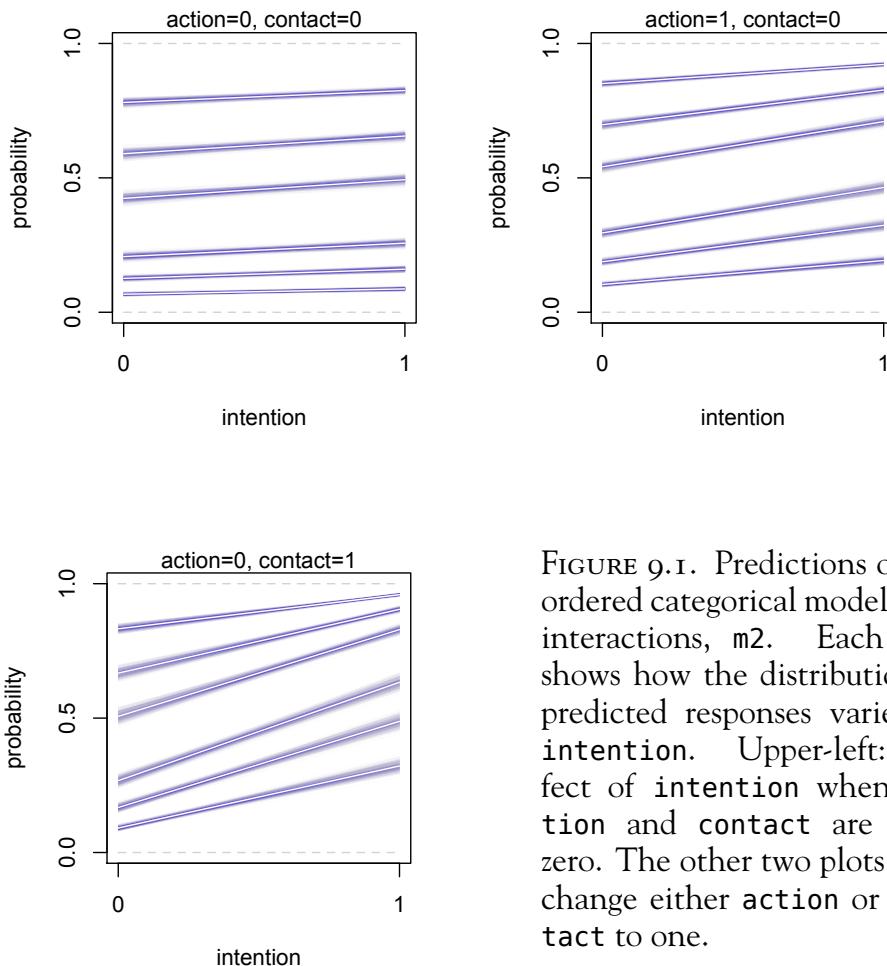


FIGURE 9.1. Predictions of the ordered categorical model with interactions, m_2 . Each plot shows how the distribution of predicted responses varies by `intention`. Upper-left: Effect of `intention` when `action` and `contact` are both zero. The other two plots each change either `action` or `contact` to one.

The thickness of the blue lines corresponds to the variation in predictions due to variation in samples from the posterior. Since there is so much data in this example, the path of the predicted boundaries is quite certain. The horizontal axis represents values of `intention`, either zero or one. The change in height of each boundary going from left to right in each plot indicates the predicted impact of changing a story from non-intention to intention. Finally, each plot sets the other two predictor variables, `action` and `contact`, to either zero or one. In the upper-left, both are set to zero. This plot shows the predicted effect of taking a story with no-action, no-contact, and no-intention and adding intention to it. In the upper-right, `action` is now set to one. This plot shows the

predicted impact of taking a story with action and no-intention (action and contact never go together in this experiment, recall) and adding intention. This upper-right plot demonstrates the interaction between **action** and **intention**. Finally, in the lower-left, **contact** is set to one. This plot shows the predicted impact of taking a story with contact and no-intention and adding intention to it. This plot shows the large interaction effect between **contact** and **intention**, the largest estimated effect in the model.

9.2.4.3. What about repeat measures? You might have realized at the start of this example that each participant responded to multiple (more than 30) different scenarios (“case” in the data frame). In other words, we have repeat measures on each participant. Not only that, but each participant responded to different versions of the same story (“story” in the data frame). So not only are there repeat measures by participant, but there are also repeat measures by story.

These data are crying out for a multilevel analysis, using random effects. Many readers, whether they’ve learned about random effects before or not, may have heard of a concern called *pseudo-replication*, which is related. It stands to reason that all of the responses from a single participant are likely correlated with one another, if for no other reason than some people like to give high responses and others give low responses. What we’re really interested in is how the factors **action**, **intention** and **contact** change a person’s response, accounting for these clustering effects. We might also be interested in the strength and patterning of the clustering itself, as well. Whether you’re interested in the clustering itself or not, failure to model it can result in imprecise or misleading estimates.

There are some good ways to deal with this kind of issue. The best approach is to use an actual *multilevel* (also known as *hierarchical*) model. You’ll have to wait until the next chapter for that solution. Another approach, quite common still in fields like economics, is to just add a categorical variable to the model, labeling each cluster of possibly-correlated outcomes with a different dummy variable. This is known as the *fixed effects* approach, for reasons you can worry about when you get to Chapter 10. In this case, the fixed effects approach is pretty expensive, computationally. There are 331 different participants in the data, so that implies 330 new independent intercept parameters! These parameters allow each participant to have their own unique average response. Then the slope parameters will indicate changes within participants, rather than changes relative to an average statistical participant.

Asking you to fit such a model, using `mle2`, would be an act of torture. You'd have to construct 330 new dummy variables and then type in an absurdly long formula for phi, with an equally absurd `start` list. So before we leave ordered categorical outcomes behind, the last thing I'd like to do is teach you how to use the convenient function `polr` to fit the same models you did above with `mle2`.

9.2.5. Using `polr`. Using `polr` to model the fixed effect on `id`.

```
m2feid <- polr( as.ordered(response) ~ as.factor(id)
+ action * intention + contact * intention ,
  data=d , Hess=TRUE )
```

R code
9.14

Look at new estimates, hiding the individual intercept estimates for each participant:

```
precis(m2feid)[331:335, ]
```

R code
9.15

	Estimate	Std. Error	2.5%	97.5%
action	-0.6594796	0.05761245	-0.7723979	-0.5465613
intention	-0.3791256	0.06177793	-0.5002081	-0.2580431
contact	-0.4425789	0.07273282	-0.5851326	-0.3000252
action:intention	-0.5866406	0.08429496	-0.7518557	-0.4214255
intention:contact	-1.7598924	0.10478376	-1.9652648	-1.5545200

Compare to previous estimates:

```
precis(m2)[7:11, ]
```

R code
9.16

	Estimate	Std..Error	2.5%	97.5%
bA	-0.4723928	0.05481514	-0.5798285	-0.3649571
bI	-0.2826887	0.05833356	-0.3970204	-0.1683571
bC	-0.3316328	0.06977651	-0.4683922	-0.1948733
bAI	-0.4464136	0.08076219	-0.6047046	-0.2881226
bCI	-1.2727347	0.09939107	-1.4675376	-1.0779318

The estimated effects have actually gotten larger, now that we've accounted to some extent for clustering by individual.

Compare AICc:

```
AICctab( m2 , m2feid , nobs=9930 , weights=TRUE )
```

R code
9.17

	dAICc	df	weight
m2feid	0.0	341	1
m2	5888.1	11	<0.001

The model with all the fixed effects seems like a sure winner. Even with 330 additional parameters, it's AICc is more than 5800 units better. There's a lot of data here, recall, so it's possible to estimate this many parameters with reasonable precision.

But I'm not inclined to go along with what the AICc analysis above tells us. Recall that the premise of AICc is that we are concerned about out-of-sample prediction, measured in a particular way. Therefore we have to consider which aspects of the model and data contain the predictions we are interested in. To what observations would we wish to generalize this model? In this case, we don't actually wish to generalize to the *same* individuals who participated already. So even though all of those individual `id` intercepts do improve fit tremendously, they aren't really going to help us predict the responses of future participants answering the same scenarios. So is it fair to reward `m2feid` for predicting the average responses of these specific individuals? I think not. Now, the estimates for the slope parameters in `m2feid` may indeed produce better out-of-sample predictions than those in `m2`. But we can't easily decide that from the AICc comparison above, because the comparison is dominated by the individual intercepts.

Instead, we'd rather have a way to focus on particular aspects of prediction, like the slope parameters, while ignoring others, like the individual intercepts. This concern is sometimes called the question of *focus*. It lies at the heart of the difference between AIC and DIC, and is intimately related to the construction and interpretation of multilevel models. We're not going to deal with this concern in any detail, right now. But when you reach Chapter 10, DIC and question of focus will appear again.

9.3. Ranked Outcomes

Problem with ranks: Must predict entire vector at once, because only one item can be #1. Use Keener and Waldman approach with orthant probabilities. Code for this is ready. Need a good example data frame. Alex's village data? Ryan's field data?

9.4. Variable Probabilities: Beta-binomial

Throughout the book so far, we've been implicitly assuming that the "true" value of each parameter was the same for every unit in the data. Every individual person, chimpanzee, nation, or location had the same α and the same β and the same p . But suppose this isn't the case. Suppose instead that different units in the data actually experience different underlying probabilities or rates.

You've already seen an empirical case in which this possibility was important, when you analyzed the UCB admissions data earlier in the previous chapter. In ignoring the fact that people applying to different departments had different overall probabilities of admission, we were lead to the wrong conclusion. It was only by estimating a different baseline, or intercept, probability in each case that we could get the model to tell us what was obvious from plotting the data.

This kind of situation is quite common. Whenever there are clusters of some kind in the data, then they might experience different probabilities of an event, because of unobserved factors linking individual observations within that cluster. In the case of the academic departments, it is that selection criteria and funding amounts are clustered by department, and so exert a common causal influence on everyone who applies, even though deviations among applicants may be due to differences among applicants.

In this section, I'm going to show you the simplest way to begin to model this kind of heterogeneity among units, without using the *fixed effects* approach we used for the UCB admissions data. There are a number of reasons to move beyond the fixed effects approach, the approach of assigning a dummy variable for each cluster. I'll speak to these reasons in much more depth in the next chapter. For now, it is sufficient to realize that the average probability in each cluster does inform us about the average probabilities in the other clusters. The fixed effects approach instead assumes that the per-cluster probabilities are completely independent of one another, and therefore the estimate of each takes no advantage of the data from the other clusters. So it will help us to pool the information and make better inferences, if we explicitly model the varying probabilities by cluster.

The second important reason for now is that often we'd like to actually estimate the distribution of the probabilities across units. We'll need such an estimate, if we want to generalize predictions to units we haven't yet sampled. For example, what about academic departments not in the UCB admission data? How can we make useful predictions for them, if we don't know their overall probabilities of admission? Well, if we know the distribution of overall probabilities, then we can at least give bracketed predictions, categorized by 10% most selective and 10% least, for example. This is important, because highly selective departments reject most everyone, and so good prediction will have to account for this possibility. Likewise, in most field biology contexts, we'd like to generalize to streams, or individuals, or transects that aren't in our data.

The fixed effects approach doesn't tell us the shape of the heterogeneity among units, and so it can't help us predict future units very well.

Finally, sometimes the scientific question itself is about heterogeneity. How much variation is there among units? In non-linear models, you can't just start calculating sums of squares in classic ANOVA fashion, so some more subtle approach is needed. One approach is to actually model the heterogeneity and estimate the parameters that describe it. That's what we're going to do now.

9.4.1. Stacking nodes. What we're going to do to address this issue is build a *mixture* model, a model that stacks together two different probability distributions. The top distribution, the top-level stochastic node, will codify our assumptions about the observed outcomes, as usual. The second distribution, a second stochastic node finally, will codify our assumptions about the parameters of the first distribution. This will allow us to explicitly model the distribution of the binomial probabilities.

What does a mixture model look like? We'll build one of the most common and useful ones here, the beta-binomial model. This model combines, as the name suggests, a binomial outcome distribution with a beta distribution that defines the probabilities in the population. This is what the model looks like:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n_i), \\ p_i &\sim \text{Beta}(\mathcal{A}, \mathcal{B}). \end{aligned}$$

Each unit i in the data produces an observed count, y_i . These counts are assumed to emerge from different underlying probabilities p_i , however, with each unit i having a different unknown probability p_i . These p_i 's in turn come from a beta distribution with shape parameters \mathcal{A} and \mathcal{B} . This is much like the y_i values come from a binomial distribution, except that we don't get to see the p_i 's. Instead we have to infer them from the observable y_i 's. This is naturally a bit confusing, so hang on, and we'll apply the approach and give you a chance to work through it in the context of an empirical example.

Before getting a better idea what the beta distribution looks like, it'd help to view the above two-node model from another perspective. Viewed purely causally, nature is sampling from the beta to produce a probability that then generates binomial outcomes. Nature samples a different p_i for each unit i . This generates heterogeneity among units. Another way to see the same model, although viewing it more mathematically now, is that the second node defines the prior probability density for p_i . Remember, when we assign a distribution to a parameter in a Bayesian perspective, we are assigning it a prior to be updated in

light of the data. And so in this context we are saying that we expect the values p_i to have a beta prior. But instead of merely assuming a flat beta density, out of ignorance, we can use the data itself to estimate the prior. This isn't cheating, because there is nothing cheating about estimating a density using Bayes' theorem. It's just that the prior for p_i is the posterior for \mathcal{A} and \mathcal{B} . So we will have to make some kind of ignorance assumptions for \mathcal{A} and \mathcal{B} , as you'll see. Those will be the prior for those parameters. But they will be updated using the data, to produce a posterior for \mathcal{A} and \mathcal{B} , which will in turn become the prior for p_i . Remember, every posterior is something's prior. Sometimes priors come from aspects of the data, while other priors in the same model do not.

9.4.2. Beta distribution. You haven't seen a beta distribution before, unless it was outside this book, so it's worth saying a little about it. The beta distribution is a probability density for probabilities themselves, continuous values between zero and one. It is a member of the exponential family, just like the Gaussian and Poisson and binomial. But I didn't mention it before, because it isn't so often used as a top-level stochastic node.¹⁰⁰ Instead, it becomes valuable when we start stacking and mixing probability distributions.

The beta distribution is very flexible, and its shape is defined by two parameters, \mathcal{A} and \mathcal{B} . Sadly, the conventional names for these parameters are α and β , and that's how they appear in most books. But since α and β are the same symbols often used to build linear models, I'm going to use the "fancy" capital letter names for them. Hopefully this will reduce any confusion for now. As you'll see a little later, we're going to re-parameterize the beta distribution anyway.

Let's aim to understand how the distribution can take different shapes. You can see some of its characteristic shapes, and how they correspond to the values of \mathcal{A} and \mathcal{B} , in FIGURE 9.2. The beta distribution can take many shapes, from flat ($\mathcal{A} = 1, \mathcal{B} = 1$, top-left), to peaked in the middle ($\mathcal{A} = 3, \mathcal{B} = 3$, top-right), to highly skewed ($\mathcal{A} = 4, \mathcal{B} = 1$, bottom-left), and to massing probability at the ends near zero and one ($\mathcal{A} = 0.9, \mathcal{B} = 0.8$, bottom-right). You can explore more shapes by changing the \mathcal{A} and \mathcal{B} values in the following code:

```
A <- 1
B <- 1
curve( dbeta(x,A,B) , from=0 , to=1 , col="slateblue" )
```

R code
9.18

As a general rule, the further the values of \mathcal{A} and \mathcal{B} get from 1, the more concentrated the density becomes, whether it is in the middle or

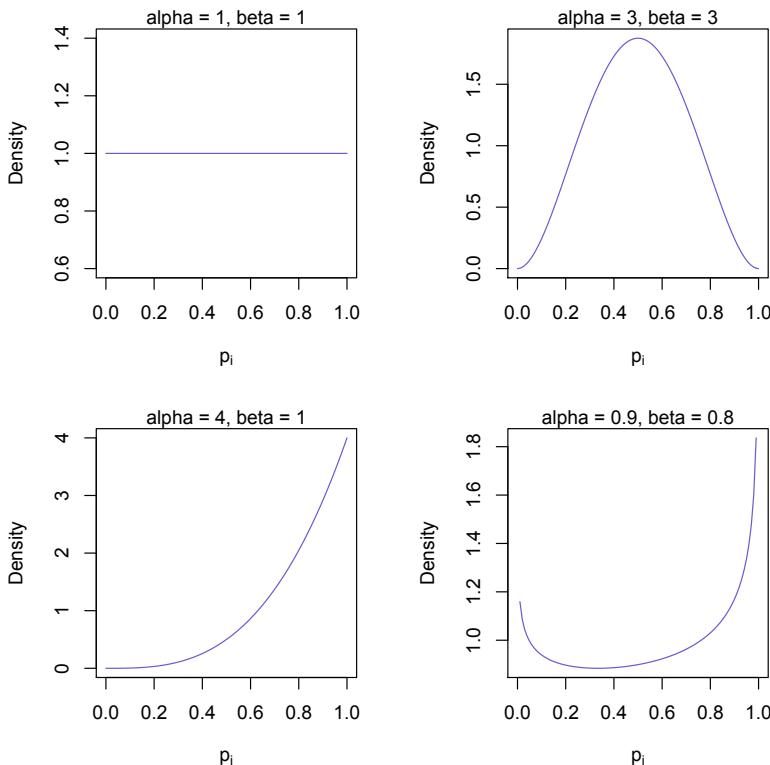


FIGURE 9.2. The beta distribution defines the probabilities of different probabilities, p_i . The horizontal axis in each plot is the range of binomial probabilities from which to draw samples. It is a parameter, but now with a different value for each case i . The vertical axis is the likelihood (density) of each probability, according to a beta distribution with shape defined by the \mathcal{A} and \mathcal{B} values shown at the top of each plot.

on the sides. The values of \mathcal{A} and \mathcal{B} must be greater than zero, but otherwise they can take any real value.

9.4.3. A mixture likelihood. We need to compute likelihoods for this mixture model, because we are going to estimate the posterior density for \mathcal{A} and \mathcal{B} . The former-parameter p is no longer estimated directly from the data, because instead we're estimating the shape of the distribution from which each p_i is sampled.

So how do we compute the posterior for \mathcal{A} and \mathcal{B} ? The posterior probability for these two parameters is defined the usual way:

$$\Pr(\mathcal{A}, \mathcal{B}|y) = \frac{\Pr(y|\mathcal{A}, \mathcal{B})}{\Pr(y)} \Pr(\mathcal{A}, \mathcal{B}).$$

The obstacle is that likelihood function, which must include both stochastic nodes. So now we expand the likelihood:

$$\Pr(y|\mathcal{A}, \mathcal{B}) = \int \Pr(y|p) \Pr(p|\mathcal{A}, \mathcal{B}) dp.$$

This just says that the likelihood of y , conditional on \mathcal{A} and \mathcal{B} , is the average likelihood of y , conditional of p , averaged across values of p . The weight given to each value of p in the average comes from $\Pr(p|\mathcal{A}, \mathcal{B})$. Another way to say this is: We don't know which value of p produced y . But we do know how likely y is, assuming different values of p . And we also know how likely each value of p is, assuming different values of \mathcal{A}, \mathcal{B} . So chaining all that logic together, we can compute the likelihood of interest. That is, provided you can solve the integral, you can. Luckily, this is one of those integrals that can be solved with standard techniques. You can just look up the solution.

But we're going to be interested in using the solution numerically, not manipulating it analytically. So we want the beta-binomial density function, within R. Several packages provide this density. I'm going to recommend using the `emdbook` package, also a part of Ben Bolker's book code, along with `bbmle`. This package provides the density as `dbetabinom`. We'll put it to use in the context of a data example.

9.4.4. Beta-binomial links. The fundamental parameters of the beta distribution are \mathcal{A} and \mathcal{B} , as you saw above. But it's going to usually be more convenient for us to model the beta distribution as a function of its *mean*, the average probability drawn from it, and *dispersion*, how spread out the distribution is. Under this parameterization, the mean is called \bar{p} (say *pee-bar*) and the dispersion θ ("theta"). These two parameters relate to \mathcal{A} and \mathcal{B} in a simple way:

$$\bar{p} = \frac{\mathcal{A}}{\mathcal{A} + \mathcal{B}}, \quad \theta = \mathcal{A} + \mathcal{B}.$$

This way of using the beta distribution is very common, and so functions like `dbetabinom` already expect input in this form, using the labels `prob` for \bar{p} and `theta` for θ .¹⁰¹

Redefining the beta distribution in this way is equivalent to using a link function. It's just like using a link, because neither of the parameters \mathcal{A} nor \mathcal{B} is the mean. But we'd like to model the mean, so we need a

function to link these parameters to the mean. Since \bar{p} is the mean, it defines our link function. But we're also going to need to use log-odds again, so that our linear model translates logically to the probability space between zero and one. So what does the model look like, under this link strategy? Defining the logit of \bar{p} as a linear model, we get:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n_i), \\ p_i &\sim \text{Beta}(\bar{p}, \theta), \\ \log \frac{\bar{p}}{1 - \bar{p}} &= \alpha + \beta x_i. \end{aligned}$$

Effectively, we've pushed the logistic down one level of stochasticity, since the binomial level is now determined by the beta probabilities. The parameters that are directly estimated in this case will be α , β , and θ .

Now, in some cases, you may need to also use a link for θ . Even if you don't want to model the dispersion as a linear model—and I'm not going to encourage that in most circumstances, mainly because trying to model both \bar{p} and θ with linear models is a formula for stumbling into a nonidentifiable model—even if you don't want to do that, you have to restrict θ to be greater than zero. An easy approach to doing this is just to use a log-link, just like the mean λ of a Poisson, so that the estimates on the natural scale are always positive. This is what it'd look like:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n_i), \\ p_i &\sim \text{Beta}(\bar{p}, \theta), \\ \log \frac{\bar{p}}{1 - \bar{p}} &= \alpha + \beta x_i, \\ \log \theta &= \tau. \end{aligned}$$

The name τ (“tau”) is arbitrary. It just needs to be different from θ , so you can keep in mind that you've estimated a parameter on the log scale as a stand-in for θ . Then you'd estimate α , β , and τ as parameters. To get the posterior for θ , you'd just exponentiate the posterior for τ . I'll use this kind of link in one of the model fits to come, so you'll get to see what it looks like in code form.

9.4.5. Example: Bolker's Reedfrogs. We'll use an example also from Bolker's book package, mortality data on reed frog tadpoles variably exposed to aquatic predators at experimentally determined densities. You can load the package and data (provided you have installed the package from CRAN) with:

```
library(emdbook)
data(ReedfrogPred)
d <- ReedfrogPred
```

R code
9.19

The data frame has 48 rows and 5 columns. We're going to be interested in predicting `surv`, the number of tadpoles that survived the duration of the experiment, our of `density` alive at the start. The predictor variables of interest will be `density`; `pred`, the presence or absence of predators; and `size`, the size of tadpoles.

Intercept-only models. It'll be useful to begin with simple models that don't include predictors, so you can get a sense for how a beta-binomial model compares in prediction to a plain binomial model. So let's fit both to these data: an old-fashioned binomial regression and a beta-binomial regression.

First, the regular binomial model. This code is just like you'd expect, after Chapter 8:

```
m0b <- mle2( surv ~ dbinom( prob=logistic(a) , size=density ) ,
  data=d , start=list(a=0) )
```

R code
9.20

No surprises there. We estimate the log-odds across all cases. Now for the Beta-binomial model:

```
m0Bb <- mle2( surv ~ dbetabinom( prob=logistic(a) , theta=theta ,
  size=density ) , data=d , start=list(a=0,theta=2) )
```

R code
9.21

This requires more explanation. Again, we're estimating log-odds, but now for the average probability of the beta density, not for the binomial. The mean probability of the beta is just $\alpha/(\alpha + \beta)$, so is a direct function of the beta parameters. The other parameter we'll use to define the underlying beta is `theta` (θ), which is equal to $\alpha + \beta$ and determines how dispersed the beta distribution is. When θ is large, the beta density is bunched up around the mode. When θ is small, the density is more spread out or, at very low values, even bunched up on the edges of the probability space. Starting search at log-odds equal to zero and $\theta = 2$ defines a flat Beta distribution.

Let's compare the fits of these model estimations, before taking a look at the estimates:

R code
9.22

```
compare(m0b,m0Bb,nobs=sum(d$density))
```

	k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC
m0Bb	2	273.0887	276.8204	1	1	0.0000	0.0000
m0b	1	574.4926	576.3602	<2e-16	<2e-16	301.4039	299.5399

Okay, so the beta-binomial model fits the data much much better, even after accounting for having one more parameter. Now how do its inferences differ, such that it does such a better job? Taking a look at the estimates from both models:

R code
9.23

```
coeftab(m0b,m0Bb)
```

	m0b	m0Bb
a	0.8431	0.9193
theta	NA	2.6157
nobs	48.0000	48.0000

So the log-odds estimates, a , aren't so different. For the binomial model, the mean posterior probability and 95% HPDI are:

R code
9.24

```
post <- sample.naive.posterior( m0b )
mean( logistic( post$a ) )
HPDI( logistic( post$a ) )
```

```
[1] 0.6990024
      lower      upper
0.6729518 0.7258218
```

And for the beta-binomial model, the mean posterior probability and 95% HPDI are:

R code
9.25

```
post <- sample.naive.posterior( m0Bb )
mean( logistic( post$a ) )
HPDI( logistic( post$a ) )
```

```
[1] 0.7132769
      lower      upper
0.6434817 0.7823066
```

Those are very similar HPDI's, and quite similar central estimates as well.

But what about the additional parameter that defines the shape of the beta density, θ ? This parameter is helping us accommodate heterogeneity in mortality across cases. We're going to plot the posterior of

the implied beta density and compare it to the observed survival rates, to see how this is so. Also, now here's a chance to teach you a new trick. The shape of the beta is determined by both the central probability p ($\alpha/(\alpha + \beta)$) and the dispersion θ ($\alpha + \beta$). So when we characterize the posterior distribution of the beta distribution, we have to account for the covariance between these two parameters. That is, we aren't going to compute a confidence interval for each or either parameter, but rather for the distribution implied by correlated samples of both parameters. Weird to say, but the code will make it easier to understand. Here's how we do it:

```
p.seq <- seq(from=0,to=1,by=0.01)
p.mu <- sapply( p.seq , function(z)
  mean( dbeta2( z , prob=logistic(post$a) , theta=post$theta ) ) )
p.ci <- sapply( p.seq , function(z)
  PCI( dbeta2( z , prob=logistic(post$a) , theta=post$theta ) ) )
```

R code
9.26

The first line just sets up a familiar sequence of horizontal axis values to compute mean predictions and confidence intervals across. The second line, beginning `p.mu`, computes the average probability, at each value in `p.seq`. It does this by passing each probability in `p.seq` into the beta density function, which is computed for each paired sample of `a` and `theta` from the posterior. Then the mean is taken and the result return. You end up with an average density at each `p.seq` value, taken over correlated samples of the underlying parameters. The third line does the same thing for the confidence interval, just replacing `mean` with `PCI` (or `HPDI` would also work here).

Now to plot it all, over the empirical distribution of survival proportions. The survival proportions in each case in the data are already available in the `propsurv` column.

```
dens(d$propsurv , col="slateblue" , xlab="proportion surviving" )
lines( p.seq , p.mu )
lines( p.seq , p.ci[1,] , lty=2 )
lines( p.seq , p.ci[2,] , lty=2 )
```

R code
9.27

The results are visible in FIGURE 9.3. The black estimated curves show the same skewed shape as the empirical density. This is what the beta-binomial model is capable of coping with, while the binomial model assumes instead a concentrated density, peaked at 0.7 on the horizontal axis in FIGURE 9.3, with 95% of its probability between 0.67 and 0.73 (the HPDI you computed earlier). Thus the pure-binomial model does

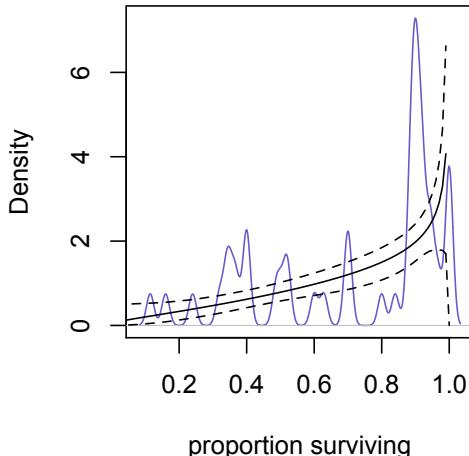


FIGURE 9.3. Estimated distribution of survival probabilities (black lines) over the empirical distribution of survival proportions (blue), for the tadpole data. Dashed curves show 95% percentile confidence interval of the Beta density estimates, sampled from the posterior.

a poor job of describing the heterogeneity in survival across cases, while the beta-binomial is built to do better at this kind of job.

Still, you might ask if the beta-binomial model is just doing better here, because we haven't yet included predictor variables to soak up some of that heterogeneity. After all, the central purpose of GLM's is to explain variation in outcomes using variation in predictors. So let's include some predictors, now.

Adding predators. Let's add a dummy variable for the presence of predators, which is likely to explain a lot of variation in survival. To make the dummy and fit both new models:

R code
9.28

```
d$pred.yes <- ifelse( d$pred=="pred" , 1 , 0 )
m1b <- mle2( surv ~ dbinom( prob=logistic(a+bp*pred.yes) ,
    size=density ) , data=d , start=list(a=0,bp=0) )
m1Bb <- mle2( surv ~ dbetabinom( prob=logistic(a+bp*pred.yes) ,
    theta=theta , size=density ) , data=d ,
    start=list(a=0,theta=2,bp=0) )
```

Comparing the models, it's again a slam dunk for the beta-binomial:

R code
9.29

```
compare(m1b,m1Bb,nobs=sum(d$density))
```

	k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC
m1Bb	3	233.9592	239.5513		1	1	0.00000 0.00000

```
m1b 2 274.2687 278.0004 1.766e-09 4.476e-09 40.30954 38.4491
```

Again, we can compare the estimates to get a better idea for how the beta-binomial is improving prediction, by accounting for heterogeneity:

```
coeftab(m1b,m1Bb)
```

R code
9.30

	m1b	m1Bb
a	2.5383	2.2138
bp	-2.6527	-2.2074
theta	NA	9.5564
nobs	48.0000	48.0000

And again the estimated log-odds (**a**) are similar across both models, as well as the new estimate for the effect of predation on log-odds, **bp**. Predators being present reduces survival probability, unsurprisingly. In fact, it takes the average survival from `logistic(2.2)≈0.9` to `logistic(2.2-2.2)≈0.5`.

But what is **theta** doing? Again, it's describing the heterogeneity that remains. Now to plot the predictions, we'll have to treat this somewhat like an interaction effect, making one plot for when predators are absent and another for when predators are present. Here's the code to compute the mean and confidence interval, when predators are present (I'll leave it to the reader to modify it for when predators are absent):

```
post <- sample.naive.posterior( m1Bb )
post$theta <- ifelse( post$theta < 0 , 0.001 , post$theta )
p.seq <- seq(from=0,to=1,by=0.01)
p.mu <- sapply( p.seq , function(z)
  mean( dbeta2( z , prob=logistic(post$a+post$bp*1) ,
    theta=post$theta ) ) )
p.ci <- sapply( p.seq , function(z)
  PCI( dbeta2( z , prob=logistic(post$a+post$bp*1) ,
    theta=post$theta ) ) )
```

R code
9.31

The only trick here is that I've forced all the samples of **theta** to be positive, because a handful are likely to be negative, which the Beta density doesn't actually allow. This has occurred because we assumed the posterior for **theta** was normal, when it can't be exactly normal, because it must be a positive number. This is the kind of aggravation that is largely avoided by using MCMC to estimate the posterior. You could also exponentiate **theta** when you use `mle2`, implying a log-link for θ . It'll have little effect on inferences in this case, but it will make you

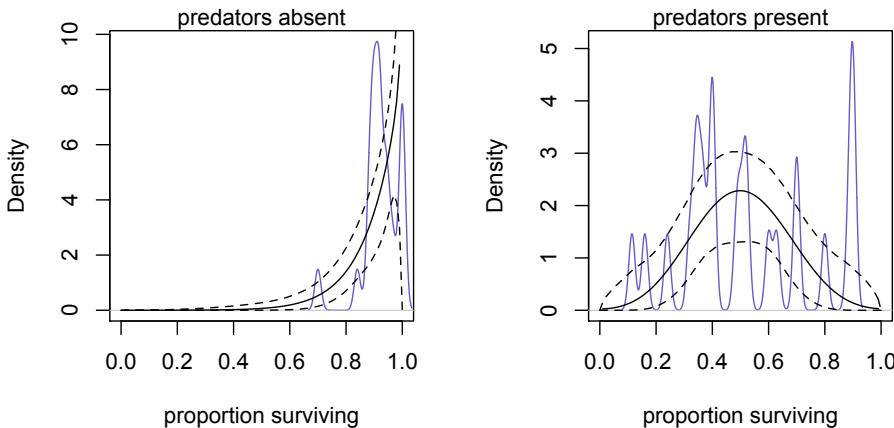


FIGURE 9.4. Estimated distribution of survival probabilities (black lines) over the empirical distribution of survival proportions (blue), for the tadpole data, now separated by cases in which predators were absent (left) and present (right). Dashed curves show 95% percentile confidence interval of the Beta density estimates, sampled from the posterior.

feel better about the inferences. To fit the same model, using a log-link for θ :

```
R code
9.32 m1Bb.logtheta <- mle2( surv ~ dbetabinom(
  prob=logistic(a+bp*pred.yes) , theta=exp(tau) ,
  size=density ) , data=d , start=list(a=0,tau=log(2),bp=0) )
```

Check the estimates from this model, exponentiating the estimate for τ , to see that you get the same central estimates as you did before. The confidence limits are not going to be in the same place, though, so assumptions do have costs. This is a great example of the kind of shenanigans that make it easy for a skilled statistician to get $P < 0.05$, by playing around with link functions in order to nudge estimates over the arbitrary boundary. In the absence of such an arbitrary threshold, the differences in inference are minor.

In any event, the results are shown in FIGURE 9.4. The predictions are of course dramatically different for non-predator cases (left) and those with predators (right). But also notice that the beta-distributed

heterogeneity allows the predicted densities to be much more spread out. The comparable pure-binomial prediction intervals are given by:

```
post <- sample.naive.posterior( m1b )
HPDI( logistic( post$a ) )
HPDI( logistic( post$a + post$bp ) )
```

R code
9.33

lower	upper
0.9042831	0.9473614

lower	upper
0.4292793	0.5109731

The first interval above is the HPDI, estimated from the pure-binomial model, for the probability of survival when no predators are present. It is concentrated around 0.92, being unable to easily predict those cases in the data with survival proportions around 0.7, as seen in the left-hand plot in FIGURE 9.4. The second interval above is for those cases in which predators were present. Now the predictions cluster around 0.47, again rather tightly. In contrast, the empirical survival proportions seen in the righthand plot in FIGURE 9.4 range from 0.1 to 0.9. The beta-distributed probabilities can estimate both the center and the spread, while the binomial model is forced to assume a single probability without heterogeneity. This is why the beta-binomial model is doing such a better job at prediction.

9.5. Variable Rates: Gamma-Poisson

The second common mixture model that explicitly models heterogeneity among units in the data is the gamma-Poisson model. Like the beta-binomial, the gamma-Poisson stacks together stochastic nodes. The gamma-Poisson is indeed to the Poisson what the beta-binomial is to the binomial. In a gamma-Poisson model or process, the mean number of events per unit time in the Poisson, λ , is assumed to vary among sampling units. These units may be different vials of fruit flies or strands of DNA or soccer teams or, as you'll model in this section, Swedish roads. Events happen within each sampling unit, but we allow for the rate of events in unit i to be λ_i . The λ_i in turn is sampled from a gamma distribution.

With these assumptions in place, you can write down the mathematical form of the gamma-Poisson model, just like you did for the beta-binomial before.

$$\begin{aligned} y_i &\sim \text{Poisson}(\lambda_i), \\ \lambda_i &\sim \text{Gamma}(\mu_i, s), \\ \log \mu_i &= \alpha + \beta x_i. \end{aligned}$$

I've gone ahead and established the link function we'll need, as well. Let me explain this model, one line at a time. The top line represents the stochastic process that produces actual observations, Poisson distributed values, y , one observed for each unique case i . This Poisson process is governed by a collection of rate parameters, λ_i , one for each unit i in the data. These varying rates in turn arise from the second line of the model. The second line represents a deeper stochastic process that creates variable λ values, according to a gamma distribution. The gamma is governed by two parameters, a mean μ_i that is unique to each unit i and a common dispersion parameter s ("scale"). Finally, we use a log-link to model μ_i , because that will ensure the gamma mean is always positive. Another common choice, as you saw in Chapter 7, is the inverse link, which can misbehave but is nevertheless the natural link for the gamma.

Why a gamma distribution? Why not use the beta again, or even a normal distribution? The first virtue of a gamma distribution is that it has the right domain for values of λ . The mean number of events in a Poisson process must be a positive real value. The gamma distribution produces positive real values. Neither the beta nor normal does. The second virtue is that the gamma is the natural sister distribution to the Poisson. If you start with a gamma distributed prior and update it with Poisson observations, the resulting posterior will also be a gamma distribution. This is just like the beta is to the binomial. The practical value of this fact is that the likelihood function for gamma-Poisson model is easy to solve for and compute. This will not necessarily be true for more complex multilevel models that you'll meet in the next chapter.

Like with the beta and the binomial, note that the gamma distribution in the model effectively defines the prior distribution for the parameter λ of the Poisson. The data, Poisson observations, will allow us to estimate the shape of this prior. The parameters of the gamma, in light of the data, will have a posterior that provides a distribution of λ_i values.

Since you met the gamma already in Chapter 7, we can jump right into using the gamma-Poisson now. That's what's next.

9.5.1. Example: Swedish traffic accidents. The data for this example come from an experiment conducted by the Swedish government in the early 1960's. They wanted to know how effective imposing speed limits would be for reducing traffic accidents. On some days a speed limit was enforced, while on other days it was not. These data have some interesting aspects we'll glide over for now. We're just going to be interested in estimating the distribution of traffic accidents, using the speed limit as a predictor variable.

Load the data with:

```
library(MASS)
data(Traffic)
d <- Traffic
d$limit.yes <- ifelse( d$limit=="yes" , 1 , 0 )
```

R code
9.34

All the last line does is make a dummy variable for the speed limit being in effect. We'll need this dummy variable later. This data frame has 184 rows, and each row is a day. We'll be interested in predicting the values in the `y` column, which is number of accidents on a given day, using the speed limit dummy variable, `limit.yes`.

Intercept models. As we did with the beta-binomial, it'll be helpful to first show how to fit a plain intercept model. I'll also compare that model to the equivalent Poisson model. Here's the code to fit both, with `m0P` being the plain Poisson model and `m0gP` being the gamma-Poisson.

```
m0P <- mle2( y ~ dpois( lambda=exp(a) ) , data=d ,
               start=list(a=log(mean(d$y))) )
m0gP <- mle2( y ~ dgampois( mu=exp(a) , scale=s ) , data=d ,
               start=list(a=log(mean(d$y)),s=1) )
```

R code
9.35

The function `dgampois`, part of the `rethinking` package, provides the needed gamma-Poisson likelihoods.¹⁰² If you go ahead and compare the AICc/BIC scores of these two models, you'll see that the gamma-Poisson is far better. But before plotting the predictions to see why, let's go ahead and include a predictor.

Adding a speed limit. The linear model of a gamma-Poisson is expanded just like that of any other GLM. So we're going to include `limit.yes` now to get an estimate of how much the speed limit reduced the number of accidents. The plain Poisson model will implicitly assume that every day in the data experienced one of two accident rates, either the no

limit rate (presumably higher) or the limited rate (presumably lower). The gamma-Poisson model, in contrast, will instead assume that each day in the data experiences a different rate, drawn from a gamma distribution. However those days on which the speed limit was in effect will draw from a different gamma distribution, with a different mean, than those on which there was no limit.

Here's the code to fit both the plain Poisson and gamma-Poisson:

```
R code
9.36
m1P <- mle2( y ~ dpois( lambda=exp(a + bl*limit.yes) ) ,
               data=d , start=list(a=log(mean(d$y)),bl=0) )
m1gP <- mle2( y ~ dgamopois( mu=exp(a + bl*limit.yes) , scale=s ) ,
               data=d , start=list(a=log(mean(d$y)),s=1,bl=0) )
```

Now let's compare all of the model fit so far:

```
R code
9.37
compare(m0P,m0gP,m1P,m1gP,nobs=nrow(d))
```

	k	AICc	BIC	w.AICc	w.BIC	dAICc	dBIC
m1gP	3	1290.212	1299.724	0.9831	0.9234	0.000000	0.000000
m0gP	2	1298.339	1304.702	0.0169	0.0766	8.126696	4.978796
m1P	2	1483.020	1489.384	<2e-16	<2e-16	192.807881	189.659980
m0P	1	1517.223	1520.416	<2e-16	<2e-16	227.010697	220.692181

The two gamma-Poisson models do much better. And the models with the speed limit predictor do much better than the corresponding intercept models. So considering the speed limit helps predictions a lot, unsurprisingly. It's really the magnitude of the effect we're interested in. And the gamma-Poisson models do much better than the plain Poisson models. They do so much better that even the intercept-only gamma-Poisson is out-performing the Poisson that includes the speed limit variable. This suggests that most of the variation in the data cannot be easily explained by the speed limit. You'll see how the gamma-Poisson deal with that variation, when we plot the predictions a little bit later.

Before plotting, though, it'll be useful to look at the parameter estimates. For model `m1gP`, which dominates the model weights:

```
R code
9.38
precis( m1gP )
```

	Estimate	Std. Error	2.5%	97.5%
a	3.1369972	0.03449857	3.0693813	3.20461318
s	2.1920987	0.33593092	1.5336862	2.85051123
bl	-0.1887361	0.05912355	-0.3046161	-0.07285606

The estimate for the speed limit is negative, -0.19 , and reliably below zero. This indicates that the speed limit did indeed reduce accidents, as assumed. But by how much? To find out, we have to reverse the link function, to get back on the count scale. The mean of the gamma will then, in turn, be the mean λ of the Poisson. The mean λ will be the mean observed count. And so we just need to exponentiate the linear model to get an MLE predicted mean number of accidents. Without and with the speed limit, the mean predictions are:

```
exp( 3.14 )
exp( 3.14 - 0.19 )
```

R code
9.39

```
[1] 23.10387
[1] 19.10595
```

So the speed limit reduced the number of accidents, on average, by 4 per day. Whether you think that is a lot or not depends upon how much you care about each accident.

But the speed limit has done more than just reduce the mean. Since the variance of counts tend to scale with the mean, it has also reduced the variation in the number of accidents. Importantly, it may have reduced the high range of counts, such that there are many fewer days with counts about, say, 30 or more, when the speed limit was in effect. In order to see the data and predictions this way, we'll need to plot the distribution of predictions, not just consider the central estimate.

I'll provide the code here to plot the predicted distribution of counts over the empirical distribution, for days on which the speed limit was in effect. The reader should be able to modify the code to produce the sister plot for days on which there was no speed limit. The plot will use samples from the posterior, as always. It'll also explicitly compare the Poisson and gamma-Poisson predictions. Here's the code:

```
post <- sample.naive.posterior( mlgP )
postP <- sample.naive.posterior( m1P )
fade <- 0.01
n <- 300
dens( d$y[d$limit.yes==1] , col="slateblue" , xlab="accidents" ,
      xlim=c(0,50) )
mtext( "limit YES" , 3 )
for ( i in 1:n ) {
  dy <- sapply( 0:50 , function(z)
    dgampois( z , mu=exp(post$a[i] + post$bl[i]) ,
              scale=post$s[i] ) )
```

R code
9.40

```

    lines( 0:50 , dy , col=col.alpha("black",fade) )
}
for ( i in 1:n ) {
  dy <- sapply( 0:50 , function(z)
    dpois( z , lambda=exp(postP$a[i] + postP$bl[i]) ) )
  lines( 0:50 , dy , col=col.alpha("black",fade) )
}

```

I'm also using this opportunity to show you another way to plot the uncertainty contained in the posterior distribution. Instead of calculating means and 95% intervals here, what I've done is draw 300 different prediction densities, sampled from the posterior. These prediction densities are overlaid on one another, with transparency, so that you can see regions of scatter and concentration. Sometimes this is a superior way to view the implications of the posterior, as it allows you to isolate actual realizations, unlike the interval plotting approach. You can alter how transparent the curves are by changing the value of `fade` on the third line of code.

FIGURE 9.5 displays the results, for both days on which the speed limit was in effect (left) and days on which it was not (right). The blue density in each plot is from the data. The black groups of transparent curves are the gamma-Poisson (wider group) and Poisson (narrower group) samples from the posterior. These plots don't make it easy to see the difference in mean number of accidents, but they do help us see the different implications of the gamma-Poisson process. The numbers of accidents across days are highly variable, much more variable than a Poisson process can suggest. In particular, on days without the speed limit (righthand plot), there is a very long tail of high accident numbers. On days with the speed limit, this tail is greatly reduced, although there are still some days with more than 40 accidents, even then. The gamma-Poisson distributions can cover this heterogeneity in number of accidents, while the Poisson curves in both cases are far too narrow and symmetrical.

You can simulate counts and directly estimate the probability of, say, 30 or more accidents per day with and without the speed limit. This will help me show you how to work with the posterior some more, as well as demonstrate that the speed limit has a more appreciable effect on the variation than it does on the mean. Remember, the average number of accidents reduced by the speed limit is only 4. But let's ask how much

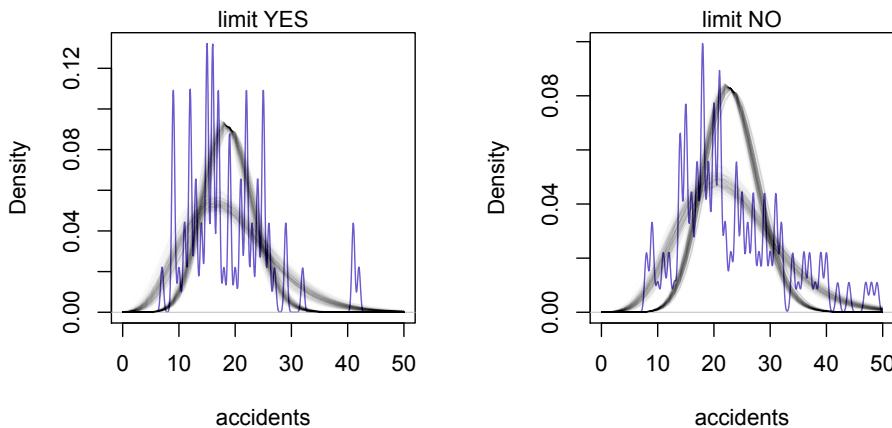


FIGURE 9.5. Predicted densities of accident counts, according to both Poisson and gamma-Poisson models. Left-hand plot shows days with the speed limit. Righthand plot shows days without it. The blue density in each plot is the empirical count distribution. The black transparent curves represent the gamma-Poisson (wider group of curves) and Poisson (narrower) prediction distributions. Each curve is a single sample from the posterior.

the upper tail gets reduced. Here is how we might compute the predicted proportions of days, with and without the limit, with 30 or more accidents.

```
post <- sample.naive.posterior( mlgP )
y.limit <- rgamposis( 10000 , mu=exp(post$a + post$bl) , scale=post$s )
y.nolimit <- rgamposis( 10000 , mu=exp(post$a) , scale=post$s )
length( y.limit[ y.limit >= 30 ] )/length(y.limit)
length( y.nolimit[ y.nolimit >= 30 ] )/length(y.nolimit)
```

R code
9.41

```
[1] 0.104
[1] 0.2081
```

The first result, 0.104, is the predicted proportion of days with 30 or more accidents, assuming a speed limit is in place. The second result is the same calculation, but assuming the speed limit is absent. While the speed limit reduced the mean number of accidents by only about 4, it halved the expected number of days with 30 or more accidents.

Or at least it did, according to these simple models. There is a lot more one could do with these data. But as far as demonstrating the potential value of modeling varying Poisson rates with a gamma-Poisson process, these models aren't leading you astray. They reveal a common truth. Many processes are too variable among units to be well-described by a simple Poisson process. They may however be more easily described by a mixture of Poisson processes.

9.6. Variable Process: Zero Inflated Outcomes

Example: bird densities. Also mention use for predicting missingness (NA).

10 Multilevel Models

This chapter introduces common generalized linear multilevel models, GLMM's. These models are also known as hierarchical models, random effects models, and mixed effects models. These terms sometimes indicate different traditions, but increasingly they refer to a common modeling strategy: allowing regression parameters to vary among clusters of observations in the data while pooling available information across clusters. Clusters may be groups of individuals, multiple observations from the same individuals, or even related clusters of deeper clusters. These models can produce better estimates of per-cluster parameters, in addition to providing direct estimates of the distribution of variation in the population. In light of the substantial heterogeneity in natural systems, these models provide a flexible strategy for improving the match of theory to statistical model, as well as improving predictions. Still, with complexity comes danger. These models are hard to estimate and sometimes even harder to understand.

You'll need to install the R package `lme4` to work with the code in this chapter. If you haven't done so yet, do so with:

```
install.packages( "lme4" , dependencies="Depends" )
```

R code
10.1

10.1. The musician who forever forgets

Intro idea: Case of Clive Wearing, as described by his wife Deborah in book *Forever Today*. Wearing is a conductor and pianist who acquired anterograde amnesia after a case of Herpes simplex encephalitis. He can still play the piano and conduct, but he cannot form new long-term memories. He is always forgetting recent events.

Statistical models that are not multilevel are similar, in the sense that as they move from one cluster (individual, group) in the data to another, trying to estimate parameters for each cluster, they forget everything about the previous clusters. They behave this way, because the model

assumptions forces them to, assuming that cluster intercepts and slopes contain no joint information. This would be like forgetting you had ever been in a coffee shop, each time you go to the coffee shop. Coffee shops do vary, but information about one does help your expectations about the others. As it is in life, it is in statistics: anterograde amnesia is bad for learning about the world. We want models that instead use all of the information in savvy ways. Multilevel models do that.

10.2. What are multilevel models good for?

The basic reason to use a multilevel model is that your data are *clustered* somehow. By clustered, I mean observations (cases) within different groups differ in their averages, resulting in correlations within groups. As a result, describing these differences can improve prediction and inference. There are two common forms of clustering: (1) repeat observations of the same entities and (2) observations of separate entities that are associated temporally, spatial or causally. In my mind, these are really the same thing, but it's worth providing examples of each.

Repeat observations arise when we observe the same individual (person, tadpole, chimpanzee) multiple times. This may arise because we use the same individuals in multiple treatments of an experiment (within subjects design). Or it may arise because some individuals simply appear more often by chance. Another common way to realize repeat observations is with a time series, in which each individual appears once at each time step, but many times within the data. In all of these contexts, heterogeneity among individuals will lead to a particular pattern of variation between observations: variation will tend to cluster, if only a little, by individual.

The entities that exhibit clustering could just as easily be days of the week, cities, or streams. We might observe individuals only once, but many individuals on common days and in common places. Observations of all individuals on Monday, for example, may cluster, because of unobserved causal influences on that day. Likewise, observations of all individuals from a common stream or other location may also cluster. In this way, we actually have repeat observations from the same time or place, and the problem is structurally similar to the repeat observation of individuals. Again, clustering may arise.

In some circles, these repeat observation and clustering scenarios are known as *pseudoreplication*, or false replication.¹⁰³ Pseudoreplication can mean two distinct things, so you have to be careful with the term. The

sense that is meant here is the presence of potentially clustered observations, for either of the reasons above. We might have repeat observations of the individual or of multiple individuals in the same place. Inference for such a data structure can confound classical analysis of variance, and the pseudoreplication concern usually arises in this context.

Classical analysis of variance can be confounded in these cases, because it is the wrong model for our hypothesis about how the data arose. The pseudoreplication concern has so permeated branches of biology that a senior colleague of mine once announced at a prestigious conference that we should treat all observations from the same individual as a single data point! Another colleague once told me that multiple fish from the same tank are equivalent to sampling a single fish! These colleagues are throwing the data out with the bath water. Of course more data is almost always better. Individuals don't always behave the same way, so multiple samples from each of them is helpful. Likewise, not every fish in the same tank behaves the same way. Thankfully, multilevel models make pseudoreplication a non-issue, in almost all cases. The mistake is to think that we can't change the model to reflect the story we believe gave rise to the data. Instead, those colleagues wanted to change the data to reflect the model.

A similar classical concern arises from *multiple comparison*. If we have many groups in the data, whether the “groups” are collections of individuals or rather of observations from the same individuals, then there are potentially many comparisons of group means. With many tests comes an inflated chance of a significant test result. This problem has lead to a number of post-estimation adjustments to P -values, such as Bonferroni corrections and such, in hopes of reducing the rate of false alarms. But an essential problem with these approaches is that they continue to assume that every group is independent of the others, and as a consequence, the estimates themselves are misleading in the first place. Post hoc correction of P -values cannot fix them. A better approach is to use a better model. A multilevel model will do the job. And of course, we shouldn't be prisoners of the Tyranny of Fisher in the first place, conflating decision with estimation.

When data are clustered, and we have good hypotheses about how that clustering might be structured, a multilevel model can provide better estimates of the variation within and between groups, as well as better estimates of common influences across groups (treatment effects, for example). So whether you are interested in the variation among groups, in itself, or rather in the effect of common predictors across groups, you are

usually going to be better off using a multilevel model. In any context in which you are tempted to add dummy variables for an even modest number of groups, you are almost certainly better off using a multilevel model with “random” effects, rather than a single-level model with “fixed” effects.

There are some costs of the multilevel approach to modeling, however. The first is that we have to make more assumptions. We have to define the distributions from which the average characteristics of the clusters arise. This is just like what we did in the previous chapter, with the beta-binomial and gamma-Poisson models. There are also new estimation challenges that come with the full multilevel approach, in which we not only estimate the parameters of the distribution of cluster characteristics, but also estimate each cluster’s unique characteristic. This estimation challenge leads us headfirst into MCMC estimation. But in this chapter, we’ll instead focus on conceptually defining these models and estimating them approximations of maximum likelihood. This will help the reader understand the models conceptually, before having to deal with the conceptual frontier of MCMC. Finally, multilevel models can be hard to understand, not only because they stack together stochastic and even non-additive stochastic effects, but also because they make predictions at different levels of the data. In many cases, we are interested in only one or a few of those levels, and as a consequence, model comparison using metrics like AIC and BIC becomes more difficult. The basic logic remains unchanged, but now we have to make more decisions about which parameters in the model we wish to focus on.

There’s a lot to accomplish in this chapter. First, let’s explore the advantages of the multilevel approach, in the context of a familiar data example from the previous chapter.

10.3. Multilevel tadpoles

In the previous chapter, I used the tadpole predation (`ReedfrogPred`) data from the `emdbook` package to illustrate the value of explicitly modeling the heterogeneity among units in the data. In that case, the units were separate aquarium tanks with tadpoles in them. The outcome of interest was the number of survivors after a fixed period. Even after accounting for differing treatment effects among the tanks, there was still substantial variation in survival rate across tanks. As a result, the beta-binomial model that explicitly modeled that variation could produce a better description of the data.

That model, recall, was specified as:

$$\begin{aligned}y_i &\sim \text{Binomial}(p_i, n_i), \\ p_i &\sim \text{Beta}(\bar{p}, \theta),\end{aligned}$$

where y_i is the number of survivors in tank i , p_i is the probability of each tadpole in tank i surviving, n_i is the number of tadpoles in tank i at the beginning of the experiment, and finally the parameters to be estimated are \bar{p} and θ . These are the parameters of the beta distribution, which estimates the shape of the variation among tanks.

But what is conspicuously missing from our work with this model are estimates of the survival probability, p_i , in each tank. We've defined those parameters, but we never estimated them. We just estimate their distribution. Wouldn't it be nice to have the individual tank estimates, in addition to the estimates defining the distribution of which they are members? If we ever hope to make decent predictions on a per-tank basis, knowing the p_i 's would be a huge help.

Of course, maybe in this case you don't care much about predicting anything in each tank, because it was an experiment, and so the tanks are regarded as ephemeral collections. But in many cases, we do care about the specific collections, because they are lasting entities. Suppose for example that the "tanks" were streams or ponds in the natural world. Now making accurate predictions is suddenly more important, and in addition identifying streams with unusually high mortality may lead to useful measurement that identifies additional causal variables. But even when we don't care about the unique clusters in our sample, it helps to estimate the per-cluster parameters, because doing so can provide better estimates of any treatment effects in the model.

Call these p_i parameters, all 48 of them, *varying effects*.¹⁰⁴ They are parameters like any other, when used to make predictions. But they are varying among clusters of observations in the data. The clusters here are tanks, and the observations are individual survive/die events of tadpoles. Varying effects provide the needed per-cluster parameters to aid in inference about both unique clusters and predictor variables. These parameters are also commonly called *random effects*. They stand in contrast to the so-called *fixed effects* or *constant effects* parameters of a single-level model, in which no assumption is made about the distribution from which the parameters arise.

So how do we modify the tadpole beta-binomial model to estimate the p_i parameters? I'm going to take a step to the side here to drop the beta distribution. Instead we'll move forward using a normal distribution now for the log-odds of survival in each tank, rather than a beta

distribution of the probability (p_i) in each tank. I do this, because this normal distribution assumption is the most common general approach to solving this problem, and it's also the easiest to estimate. In the next chapter, I'll show you how to bring the beta distribution back, if you want to. But you can get all of the conceptual lessons, either way.

What we want to do is define the log-odds of survival in each tank i as $\alpha + \alpha_i$. The parameter α , with no i , is what it has usually been: the average log-odds in the entire sample. But each α_i , with an i , is the deviation from this global average within each tank i . These will be the varying intercepts for each tank. Then we state that each varying intercept comes from a normal distribution with mean zero and standard deviation σ . In sum, the multilevel tadpole survival model is:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n_i), \\ \log \frac{p_i}{1 - p_i} &= \alpha + \alpha_i, \\ \alpha_i &\sim \text{Normal}(0, \sigma). \end{aligned}$$

You can fit this model, using techniques I'll summarize later in this chapter, with:

R code
10.2

```
library(emdbook)
data(ReedfrogPred)
d <- ReedfrogPred
d$tank <- 1:nrow(d)
mmbr <- glmm( cbind(surv,density-surv) ~ (1|tank) ,
  data=d , family=binomial )
```

This model fit provides estimates for 50 parameters: one overall sample intercept α , the variance among tanks σ , and then 48 per-tank intercepts α_i . You can see the per-tank estimates with:

R code
10.3

I won't reprint those varying intercept estimates here. Instead, I want to display the 48 estimates and compare them to the raw empirical proportions of survival in each tank. You can see them both plotted in FIGURE 10.1. The horizontal axis is tank index, from 1 to 48. The vertical is proportion of survivors in a tank. The filled blue points show the raw proportions, computed from the observed counts. These values are already present in the data frame, in the `propsurv` column. The

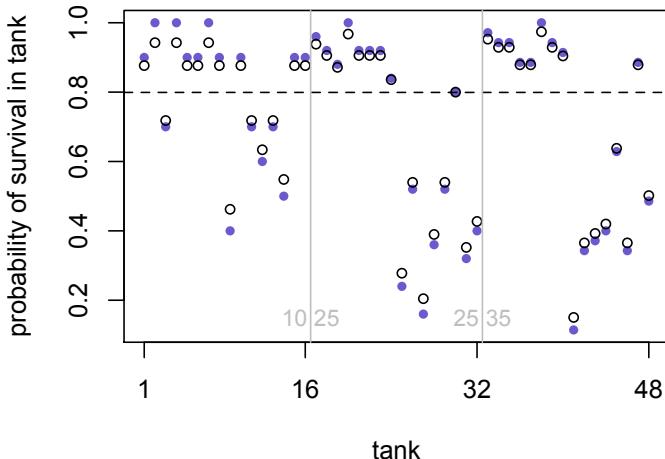


FIGURE 10.1. Empirical proportions of survivors in each tadpole tank, shown by the filled blue points, plotted with the 48 per-tank estimates from the multilevel model, shown by the black circles. The dashed line locates the overall average proportion of survivors across all tanks. The vertical lines divide tanks with different initial counts of tadpoles, 10 (left) and 25 (middle) and 35 (right). Each value on the horizontal axis is the index of a tank (row) in the data. In every tank, the estimate from the multilevel model is closer to the dashed line than the empirical proportion is. This reflects the pooling of information across tanks, to help with inference about each tank.

black circles are instead the varying intercept estimates, α_i . The horizontal dashed line at about 0.8 is the global average survival proportion, α . The vertical gray lines divide tanks with different initial counts of tadpoles, 10 (left) and 25 (middle) and 35 (right).

I want to point out three facts about these varying effects estimates. First, notice that in every case, the α_i estimate is closer to the dashed line than the raw empirical estimate is. It's as if the entire distribution of black circles has been shrunk towards the dashed line at the center of the data, leaving the blue points behind on the outside. This phenomenon is sometimes called *shrinkage*. Second, notice that the estimates

for the smaller tanks have shrunk further from the blue points. As you move from left to right in the figure, the initial densities of tadpoles increases from 10 to 25 to 35, as indicated by the gray dividers. In the smallest tanks, it is easy to see differences between the open estimates and empirical blue points. But in the largest tanks, there is little difference between the blue points and open circles. Varying effects for the smaller tanks, with smaller sample sizes, shrink more. Third, note that the further a blue point is from the dashed line, the greater the distance between it and the corresponding varying effect estimate. Shrinkage is stronger, the further a tank's empirical proportion is from the global average.

10.3.1. Pooling. All three of these phenomena arise from a common cause: pooling information across clusters (tanks) to improve estimates. What *pooling* means here is that each tank provides information that can be used to improve the estimates for all of the other tanks. Each tank helps in this way, because we made an assumption about how the varying log-odds in each tank, α_i , related to all of the others. We assumed a distribution, the normal distribution in this case. Once we have a distributional assumption, we can use Bayes' theorem to optimally share information among the clusters.

Think of it this way. Suppose you encounter the data for each tank of tadpoles in sequence. Before you see the outcome of the first tank, tank 1, you can have only a naive notion of what the observed survival proportion will be. But once you see the outcome of tank 1, you've gained information and your guess has improved. Now the second tank comes along. Before you realize the outcome of the second tank, can you say anything about its proportion that will survive? Of course you can, because you have the outcome of the first tank to improve your prediction. You won't be too confident in your guess, perhaps. But it'll be better on average than your truly naive guess was for tank 1. And once the second tank's data are realized, you can improve your guess from generalizing the first tank's outcome, using the data from the second tank. You can think of this procedure like the Bayesian updating example in Chapter 2. Tank 1's estimate improves our estimate for tank 2.

But wait. There's nothing special about the order the tanks appear, as long as they are independent. What if we lost the data for tank 1, and so started out analysis with tank 2? After calculating the observed proportion surviving in tank 2, an assistant finds the lost data for tank 1. Now, before you see what is written on the data sheet, can you say anything about what the proportion will be? Again, of course you can, because you know what happened in tank 2. You take that educated

guess and update it, once you are handed the actual data for tank 1. And so tank 2's estimate improves our estimate for tank 1.

In truth, this relationship is symmetric. Tank 1's data improves the estimate for tank 2, and likewise tank 2's data improves the estimate for tank 1. As a result, if you do the estimation correctly, according to Bayes' theorem, you end up with estimates for both tanks in which you have pooled all the information to improve each estimate. But as a result, neither estimate is exactly the same as the naive estimate you'd get by forcing the estimate for each tank to ignore the other tank. Instead, both estimates have shrunk towards the mean proportion across both tanks. These estimates will exhibit *shrinkage*, due to pooling information.

To gain a better appreciation of the shrinkage phenomenon, and why it varies as it does across the tanks, it will help to directly address the inferential benefits of pooling information in this way. That's where we turn next.

10.4. Varying effects and the bias-variance tradeoff

The first major benefit of using these varying effects estimates, instead of the empirical raw estimates, is that they provide more accurate estimates of the individual cluster (tank) intercepts.¹⁰⁵ Now, in any particular case, it's possible to get a raw estimate that is closer to the true cluster mean than the varying effect estimate is. But on average, the varying effects actually provide a better estimate of the individual tank (cluster) means. In this section, I'll explain why this is true, in the context of building the model above, the one with varying intercepts by tank.

The basic reason that the varying intercepts provide better estimates is that they do a better job of trading off bias (underfitting) and variance (overfitting). You first met this distinction and estimation problem in Chapter 5. To understand this in the context of our current data example, suppose that instead of tanks we had ponds with some continuity through time, so that we might be concerned with making predictions for the same clusters in the future. We'll approach the problem of prediction future survival in these ponds, from two extreme perspectives.

First, suppose you ignore the varying effects and just use the overall mean across all ponds, α , to make your predictions for each pond. A lot of data contributes to your estimate of α , and so it can be quite precise. However, your estimate of α is unlikely to exactly match the mean of any particular pond. As a result, the total sample mean exhibits high *bias*, it underfits the data. This approach is sometimes known as *complete*

pooling, because you pool all the data from all ponds to produce a single estimate that is applied to every pond.

In contrast, suppose you use the raw empirical survival proportions to make predictions for each pond. This is like using a separate constant regression intercept for each pond. In previous chapters, you estimated models like this by including a dummy variable for each cluster in the data, like with the academic departments in the Berkeley admissions data in Chapter 8. The blue points in FIGURE 10.1 are this same kind of estimate. In each particular pond, quite little data contributes to each estimate, and so these estimates are rather imprecise. This is particularly true of the smaller ponds, where less data goes into producing the estimates. As a consequence, the *variance* of these estimates is high, and they are rather overfit to the data. Standard errors for each intercept can be very large, and in extreme cases, even infinite. These are sometimes called the *no pooling* estimates. No information is shared across ponds, in this case.

When you estimate varying intercepts, however, you use *partial pooling* of information to produce estimates for each group that are less biased (underfit) than the grand mean and less variable (overfit) than the no pooling estimates. As a consequence, they tend to be better estimates of the true per-cluster (per-pond) means. This will be especially true when ponds have few tadpoles in them, because then the no pooling estimates will be especially overfit (variable). When a lot of data goes into each pond, then there will be less difference between the varying effect estimates and the no pooling estimates, but any difference will still be expected to be to the varying effect estimates' advantage.

It's useful to demonstrate this fact, but to do so, we'll have to simulate some tadpole data. In that case, we'll know the true per-pond survival probabilities. Then we can compare the no-pooling estimates to the varying effect estimates, by computing how close each gets to the true values they are trying to estimate.

The rest of this subsection outlines how to do such a simulation. Learning to simulate and validate models and model fitting in this way is extremely valuable. Once you start using more complex models, whether you fit them with canned algorithms or write your own Markov chains (as in the next chapter), you will want to ensure that your code is working and that you understand the model. You can help in this project by simulation code from the model, with specified parameter values, and then making sure that your method of estimation can recover the parameters within tolerable ranges of precision. Even just simulating data

from a model structure has a huge impact on understanding, in my experience.

The first step is to define the model we'll be using. I'll use the same basic multilevel binomial model as before:

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n_i), \\ \log \frac{p_i}{1 - p_i} &= \alpha + \alpha_i, \\ \alpha_i &\sim \text{Normal}(0, \sigma). \end{aligned}$$

So to simulate data from this process, we need to assign values to:

- α , the average log-odds of survival in the entire population of ponds
- σ , the standard deviation of the distribution of log-odds of survival among ponds
- α_i , a series of individual pond intercepts, one for each pond i .

We'll also need to assign sample sizes, n_i , to each pond. But once we've made all of those choices, we can easily simulate counts of surviving tadpoles, straight from the top-level binomial process, using `rbinom`.

Assign values to the parameters. I'm going to assign specific values representative of the actual tadpole data, to make the upcoming plot that demonstrates the increased accuracy of the varying effects estimates. But you can come back to this step later and change them to whatever you want.

Here's the code to initialize the values of α , σ , the number of ponds, and the sample size n_i in each pond.

```
a <- 1.4
sigma <- 1.5
nponds <- 60
ni <- rep( c(5,10,25,35) , each=15 )
```

R code
10.4

I've chosen 60 ponds, with 15 each of initial tadpole density 5, 10, 25, and 35. I've chosen these densities to illustrate how the error in prediction varies with sample size.

The values $\alpha = 1.4$ and $\sigma = 1.5$ define a normal distribution of individual pond log-odds of survival. The varying effect estimate for each pond is its offset from the grand mean, α . So now we need to sample all 60 of these α_i values from the implied normal distribution with mean 0 and standard deviation σ :

R code
10.5

```
ai <- rnorm( nponds , mean=0 , sd=sigma )
```

If you want the true log-odds of survival for a pond i , you just add $\alpha + \alpha_i$.

Finally, let's bundle some of this information in a data frame, just to keep it organized.

R code
10.6

```
dsim <- data.frame( pond=1:nponds , ni=ni , true.ai=ai )
```

Go ahead and inspect the contents of `dsim`, the simulated data. The first column is the pond index, 1 through 60. The second column is the initial tadpole count in each pond. The third column is the true log-odds survival offset for each pond, α_i .

Simulate survivors. Now we're ready to simulate the binomial survival process. Each pond has n_i potential survivors, and nature flips each tadpole's coin, so to speak, with probability of survival unique to each pond. This probability p_i is implied by the model definition, and is equal to:

$$\log \frac{p_i}{1 - p_i} = \alpha + \alpha_i,$$

$$p_i = \frac{\exp(\alpha + \alpha_i)}{1 + \exp(\alpha + \alpha_i)}.$$

The model uses a logit link, and so the probability is defined by the logistic function, as in previous chapters.

Putting the logistic into the random binomial function, we can generate a simulated survivor count for each pond:

R code
10.7

```
dsim$yi <- rbinom( nponds , prob=logistic( a + dsim$true.ai ) ,  
size=dsim$n )
```

As usual with R, if you give it a list of values, it returns a new list of the same length, and each value in the input list was used once. In the above, each paired α_i (`dsim$true.ai`) and n_i (`dsim$n`) is used to generate a random survivor count with the appropriate probability of survival and maximum count. These counts are stored in a new column in `dsim`.

Compute the no-pooling estimates. We're ready to start analyzing the fake data, now. The easiest task is to just compute the no-pooling estimates. We can accomplish this straight from the empirical data, just

by calculating the proportion of survivors in each pond. I'll keep these estimates on the probability scale, instead of translating them to the log-odds scale, because we'll want to compare the quality of the estimates on the probability scale later.

```
dsim$p <- dsim$yi / dsim$ni
```

R code
10.8

Now there's another column in `dsim`, containing the empirical proportions of survivors in each pond. These are the same no-pooling estimates you'd get by fitting a model with a dummy variable for each pond.

Compute the partial-pooling estimates. Now for the partial-pooling, or varying effect, estimates. I haven't yet explained how the code for these models works in R, and really it's not practical to use `mle2` to fit these models. The reason is that many multilevel models, including this one, cannot be exactly fit by maximum likelihood, because no one can solve the integrals required for a closed form likelihood function. This is the problem that Markov chains will solve for us, in the next chapter. But even in the special cases in which we can almost write exact likelihoods, the code would be pretty monstrous and is a bit beyond the scope of this book.

But there are pragmatic and successful approximations that allow us to get the estimates we are after, and the package `lme4` provides an efficient implementation of them. In the case of purely linear multilevel models, in which both the top-level stochastic node and the varying effects distributions are Gaussian, it is possible to get very good estimates, indeed. But even in binomial cases like this one, as long as the model isn't too complex, the estimates are quite good. Furthermore, even if you intend to eventually fit all of your models with MCMC, you'll usually want to see what `lme4` says first, if only to get reasonable values for the parameters to start search at.

Anyway, how do you specify this model in `lme4`? In the `rethinking` library, I provide `glmm` (GENERALIZED LINEAR MULTILEVEL MODEL), a convenience interface to the function `glmer` (GENERALIZED LINEAR MIXED EFFECTS REGRESSION) in `lme4`. Whichever function you use, the syntax is the same. For our case:

```
m <- glmm( cbind(yi,ni-yi) ~ (1|pond) , data=dsim , family=binomial )
```

R code
10.9

First, the outcome is specified just like you'd specify a binomial outcome with R's `glm` command, you use `cbind` to join the success counts

to matched failure counts. The counts y_i are the survivors and $n_i - y_i$ are the deaths.

Then on the other side of the tilde, we specify a varying effect by enclosing it inside parentheses. We want varying intercepts, on a per-pond basis. And so we tell `glmm` to make the intercept ("1") conditional ("|") on `pond`. So read the code piece `(1|pond)` as *intercepts vary across ponds*. The distribution of these intercepts is automatically assumed to be Gaussian, because that's all `glmm` (or `glmer`, as they are the same function) can really do. Again, once you learn MCMC, that limitation is removed. But as you'll see, you can really do a huge amount of productive modeling within such a limitation.

Back to the task of comparing no-pooling estimates to the varying effect estimates. We've fit the basic varying intercept model above. You can take a look at the estimates for α and σ with the usual `precis` approach:

R code
10.10

```
precis(m)
```

Quadratic approximation (standard errors) unreliable for variance components. Use MCMC to estimate precision of variance components.

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	1.255438	0.2021245	0.8592818	1.651595
((Intercept) pond)	1.365732	NA	NA	NA

Ignore that warning for a moment. I'll explain it later. Let's focus on the estimates themselves. This table shows only the parameters that apply across all ponds. You can access these estimates in raw form with `fixef(m)`, as these are sometimes known as *fixed effects*. The first row, labeled `(Intercept)` is the estimate of α , the grand mean log-odds of survival across all ponds. You can convert it to the probability scale with:

R code
10.11

```
logistic( 1.26 )
```

```
[1] 0.7790261
```

So the overall rate of survival is just below 80%. The second row of `precis` output, labeled `((Intercept)|pond)`, is the estimate of σ , the standard deviation of the normal distribution of individual pond log-odds of survival, the α_i values.

Now notice that there is no standard error estimate for σ here, and therefore no estimated confidence interval. The reason there is no standard error for σ here is that it would be very difficult to trust a Gaussian

approximation of the posterior of σ . It was awkward enough at times in simple linear Gaussian models in early chapters, like Chapter 4. And as models become more complex, it is increasingly unlikely that the posterior of a variance component like σ will be normal. This is why total warning appears above the `precis` output. And it is also one of the reasons MCMC has become so popular, because MCMC doesn't have any issues with Gaussian shapes or not. You'll see how that works in the next chapter.

Anyway, for the moment, we'd like to see the estimates for the remaining 60 parameters, the α_i varying effect estimates. Where are they? You can see them with:

```
ranef( m )
```

R code
10.12

Again, I won't reprint them here. But understanding the structure of what `ranef` returns is useful. When you invoke `ranef(m)`, what you get as a result is actually a complex structure of individual cluster (pond) estimates, for any varying effects you included in your model. For the moment, the result is pretty simple: You'll see a header `$pond`, indicating the grouping variable for the varying effect estimates to follow. Then you'll see a single column of estimates for α_i , titled `(Intercept)`. These are the intercepts conditional on pond.

Now that we've found these estimates, let's compute the predicted survival proportions, according to them, and add those proportions to our growing simulation data frame. I'll call the column `varying.pi`, to indicate that it contains the expected per-pond survival probabilities, derived from the varying effect estimates.

```
dsim$varying.pi <- logistic( ranef(m)$pond[,1] + fixef(m)[1] )
```

R code
10.13

If we want to compare to the true per-pond survival probabilities used to generate the data, then we'll also need to compute those, using the `true.ai` column:

```
dsim$actual.pi <- logistic( a + dsim$true.ai )
```

R code
10.14

The last thing we need to do, before we can plot the results and realize the point of this lesson, is to compute the absolute error between the estimates and the true varying effects. This is easy enough, using the existing columns:

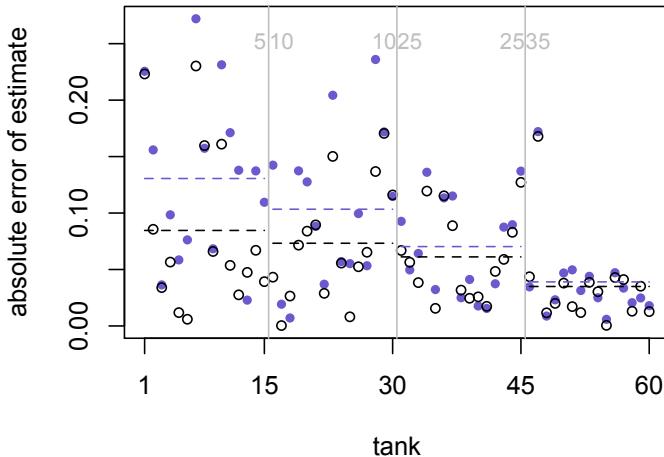


FIGURE 10.2. Error of no-pooling and varying effects estimates, for the simulated tadpoles. The horizontal axis displays pond number, i . The vertical axis measures the absolute error in the predicted proportion of survivors, compared to the true value used in the simulation. The higher the point, the worse the estimate. No-pooling empirical estimates of proportion surviving in each tank are shown by the filled blue points. Varying estimates are shown by the black circles. The dashed blue and black lines show the average error for each kind of estimate, across each initial density of tadpoles (sample size). Smaller tanks produce more error, but the varying estimates are also always better, on average, especially in smaller tanks.

R code
10.15

```
nopool.error <- abs( dsim$p - dsim$actual.pi )
varying.error <- abs( dsim$varying.pi - dsim$actual.pi )
```

Now we're ready to plot. The result you've been waiting for is in FIGURE 10.2. The horizontal axis is the pond index, from 1 through 60. The vertical axis is the distance between the predicted probability of survival and the actual probability of survival. So points close to the bottom had low error, while those near the top had a large error, more

than 20% off in some cases. The filled blue points display the no-pooling estimates. The black circles show the varying effect estimates. The vertical gray lines divide the groups of ponds with different initial densities of tadpoles. And finally, the dashed blue and black lines show the average error of the no-pooling and varying effect estimates, respectively, for each group of ponds.

The first thing to notice about this plot is that both kinds of estimates are much more accurate for larger ponds, on the right side. This arises because more data means better estimates, usually. In the small ponds, sample size is small, and neither kind of estimate can work magic. Therefore, prediction suffers on the left side of the plot. Second, note that the blue dashed line is always above the black dashed line. This indicates that the no-pool estimates, shown by the blue points, have higher average error in each group of ponds. So even though both kinds of estimates get worse as sample size decreases, the varying effect estimates always have the advantage, at least on average. Third, and this is the observation that will help you grasp shrinkage, the distance between the blue dashed line and the black dashed line grows, the smaller the pond gets. So while both kinds of estimates suffer from reduced sample size, the varying effect estimates suffer less.

Okay, so what are we to make of all of this? Remember, back in FIGURE 10.1 (page 413), the smaller tanks demonstrated more shrinkage towards the mean. Here, the ponds with the smallest sample size show the greatest improvement over the naive no-pooling estimates. This is no coincidence. Shrinkage towards the mean results from trying to negotiate the underfitting and overfitting risks of the grand mean on one end and the individual means of each pond on the other. The smaller tanks/ponds contain less information, and so their varying estimates are influenced more by the pooled information from the other ponds. In other words, small ponds are prone to overfitting (high variance), and so they receive a bigger does of the underfit grand mean (bias), to move them towards the optimal tradeoff of underfitting and overfitting. Likewise, the larger ponds shrink much less, because they contain more information and are prone to less overfitting. Therefore they need less correcting to negotiate the bias-variance (underfitting-overfitting) trade-off. When individual ponds are very large, pooling in this way does hardly anything to improve estimates, because the estimates don't have far to go. But in that case, they also don't do any harm, and the information pooled from them can substantially help prediction in smaller ponds.

The partially pooled estimates provide better average prediction. And they do so because they adjust individual cluster (pond) estimates to negotiate the underfitting-overfitting tradeoff. For this reason alone, varying effect estimates are often worth including in a model. But there are other advantages to them, as well.

10.5. Everything can vary and probably should

Reasons to use varying slopes.

10.5.1. Using varying intercepts to improve constant estimates. Another reason for the growing popularity of multilevel models is to improve inference about predictor variables. Even if you don't hypothesize that the effect of a variable is different across clusters, accounting for variation in the overall responses among clusters can help you get better estimates of even the non-varying effects.

The easiest way to appreciate this fact is to recall the `UCBadmit` data from Chapter 8. In those data, failing to model the varying means across departments lead to exactly the opposite inference of the truth. I addressed the problem back then by using naive no-pooling estimates derived from adding a dummy variable for each department. But since the varying effect estimates are on average superior to such no-pooling estimates, we could have done even better by using a multilevel model.

In fact, let's quickly return to those data and do just that. This will also allow me to show you some more aspects of the notation and of these models. Unlike in the tadpole data, now our clusters (departments) will span the rows in the data frame. So we're going to need another index, j , to indicate clusters, while we keep i to indicate rows within them. Let's load the data and add these i and j labels, to make this point clear.

R code
10.16

```
data(UCBadmit)
d <- UCBadmit
d$male <- ifelse( d$applicant.gender=="male" , 1 , 0 )
d$i <- rep( 1:2 , times=6 )
d$j <- rep( 1:6 , each=2 )
d
```

	dept	applicant.gender	admit	reject	applications	i	j
1	A	male	512	313	825	1	1
2	A	female	89	19	108	2	1
3	B	male	353	207	560	1	2
4	B	female	17	8	25	2	2
5	C	male	120	205	325	1	3

6	C	female	202	391	593	2	3
7	D	male	138	279	417	1	4
8	D	female	131	244	375	2	4
9	E	male	53	138	191	1	5
10	E	female	94	299	393	2	5
11	F	male	22	351	373	1	6
12	F	female	24	317	341	2	6

On the righthand side, you can see the new columns. The first row is row 1 within cluster 1. The 6th row is the 2nd row within cluster 3.

With this new notation in hand, the model we want to estimate is:

$$\begin{aligned} a_{ij} &\sim \text{Binomial}(p_{ij}, n_{ij}), \\ \log \frac{p_{ij}}{1 - p_{ij}} &= \alpha + \alpha_j + \beta m_{ij}, \\ \alpha_j &\sim \text{Normal}(0, \sigma). \end{aligned}$$

The outcome variable a_{ij} is the number of admit decisions, `admit`, and the sample size in each case is n_{ij} , `applications`. Notice that the index on the p values is now ij . This is because probabilities vary by department, as indexed by j . Once there are predictor variables in the log-odds linear model, p_{ij} will also vary by i . Likewise, the parameters to estimate, α_j , also vary by department. All that has really changed is that now multiple rows in the data will contribute data to the estimated varying effect for each cluster. But the above notation is very standard and actually much more common than the tadpole example above. So it's worth making sure you understand it.

Here's the code to fit this model:

```
m2 <- glmm( cbind(admit,reject) ~ (1|dept) + male ,
  data=d , family=binomial )
precis( m2 )
```

R code
10.17

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	-0.59311313	0.45708854	-1.4889902	0.3027639
male	-0.09400996	0.08062667	-0.2520353	0.0640154
((Intercept) dept)	1.10954533	NA	NA	NA

The estimated effect of `male` is very similar to what we got in Chapter 8. But now we also have better estimates of the individual department average acceptance rates. You can inspect those estimates with `ranef(m2)`. Go ahead and look at them now. You'll see that the departments are ordered from those with the highest proportions accepted to the lowest. Remember, the values provided by `ranef(m2)` are the α_j estimates, and

so they are deviations from the global mean α , which in this case has MLE -0.59 .

10.5.2. Varying effects of being male. Now, in order to teach you about varying slopes, let's consider the variation in gender bias among departments. Sure, overall there isn't much evidence of gender bias in the previous model. But what if we allow the effect of an applicant's being male vary in the same way we already allowed the overall rate of admission to vary? Such a model is a *varying slopes* (or *random slopes*) model. In this case, the model is specified by defining a series of individual department effects of the variable `male`. In effect, we assume that every department not only has its own intercept (its log-odds of admission), but it also has its own slope (the change in log-odds of admission arising from changing an applicant's gender from female to male).

Specifying this model will require a little more work, although conceptually it is familiar territory. The trouble is that, in order to successfully pool information, we'll need to define distributions of not only the varying intercepts and slopes, but also their correlation. Another way to think of this is that we are defining a prior probability density for the cluster intercepts and slopes. We'll estimate this prior from the data, so it's an empirical prior, but we still need to define its basic shape, the parameters that will describe it. If we don't allow for the Gaussian distribution of intercepts to be correlated with the Gaussian distribution of slopes, then we are assuming they are uncorrelated. Now, maybe the overall rate of admissions in each department really is uncorrelated with any bias by gender in admissions. But what if they are correlated? What if departments that admit most applicants also favor female applicants? In that case, defining the distribution of varying intercepts and slopes as being correlated allows us to pool information not only across clusters (departments), but also across parameter types.

Suppose for example that you analyze the data for all departments except for the smallest one, finding a strong correlation between the overall probability of admission and the amount of gender bias. Now, before even seeing the data for the final department, you expect its intercept and slope to be correlated. Furthermore, if you learn its slope, it changes your guess about its intercept. Likewise, if you learn its intercept, it changes your guess about its slope. In reality, both estimate simultaneously inform one another, just as all departments simultaneously inform one another. This is how allowing the varying intercepts and slopes to be correlated allows us to better pool information across clusters in the data.

Another reason to allow for the correlation is that, if you want to generalize to new sampling clusters in the future, then you will need a complete definition of the joint distribution of the varying intercepts and slopes. If you assume they are uncorrelated, you might well make worse predictions than if you estimated their correlation. Now, if you have good reason—either theoretical or practical—to assume the correlation is zero, then certainly do so. But otherwise, assuming that the distribution of varying intercepts is uncorrelated with the distribution of varying slopes is a lot like assuming that the posterior distributions of any two parameters are perfectly uncorrelated. As you know by now, having used so many variance-covariance matrixes in this book, this is sometimes nearly true, but often not even close to true.

This is what the model looks like:

$$\begin{aligned} a_{ij} &\sim \text{Binomial}(p_{ij}, n_{ij}), \\ \log \frac{p_{ij}}{1 - p_{ij}} &= \alpha + \alpha_j + (\beta + \beta_j)m_{ij}, \\ \begin{pmatrix} \alpha_j \\ \beta_j \end{pmatrix} &\sim \text{Normal}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \rho\sigma_\alpha\sigma_\beta \\ \rho\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix}\right). \end{aligned}$$

The symbol m_{ij} indicates the value of `male` for the i -th row of the j -th department. It is multiplied by the sum $\beta + \beta_j$, which is a total slope defined by both a value common to all departments, β , and a value unique to each department, β_j . Just as with the varying intercepts α_j , the varying slopes β_j are drawn from a normal distribution, and the distribution of both the varying intercepts and slopes is defined by the third line.

So how are you supposed to read that third line? This mess is what is needed to define a joint prior for the varying intercepts, α_j , and varying slopes, β_j . Now we have two standard deviation parameters, one governing each type of varying effect, so I have relabeled them σ_α and σ_β to indicate the batch of effects each describes. But these two normal distributions are also possibly correlated with one another, and their correlation is measured by ρ (“rho”). The final line above reads: *sample pairs of α_j and β_j values, one pair for each department j , from a bivariate normal distribution with variance-covariance matrix defined by the variances σ_α^2 and σ_β^2 and correlation ρ .*

Think of these α_j and β_j values as being analogous to the sets of samples you drew from the naive posterior in previous chapters. Using the variance-covariance matrix of the posterior (`vcov`), you could define a multivariate normal distribution and sample random numbers from.

The values across columns in the resulting table had a special correlation structure. Likewise, these varying intercepts and slopes will have a special correlation structure, defined by ρ . The notation is a bit weird at first, but you'll get used to it, after a few more examples.

To estimate this model, use:

R code
10.18

```
m3b <- glmm( cbind(admit,reject) ~ (1+male|dept) + male ,
  data=d , family=binomial )
```

The varying effect specification `(1+male|dept)` indicates varying intercepts, 1, as well as varying slopes for the effect of `male`, both condition on department, `dept`. Now let's look at the non-varying estimates:

R code
10.19

```
precis(m3b)
```

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	-0.4848563	0.5473287	-1.5576007	0.5878882
male	-0.1610197	0.1740543	-0.5021599	0.1801204
((Intercept) dept)	1.3271687	NA	NA	NA
(male dept)	0.3578290	NA	NA	NA

The first row of output is the estimate for α , the average log-odds of admission across all departments. Notice that is a highly imprecise estimate, which results from there being a lot of variation among departments in their admissions rates. The second row of output is the estimate of β , the average effect of being male on log-odds of admission. This is negative, but also highly imprecise. The last two rows of output are the estimates for σ_α and σ_β , the standard deviations of the varying intercepts and slopes.

What can you say from these estimates? Notice that the standard deviation of the intercepts is more than three times as large as the standard deviation of the slopes for `male`. A solid inference to draw from this difference is that more of the variation in log-odds of admission in the entire data arises from differences among departments in overall rates of admission than it does from differences in gender bias. That is, heterogeneity in overall admission rates is more influential than heterogeneity in gender bias in admission.

So what about ρ , the correlation between overall admission rate and gender bias, across departments? You can see its estimate by typing `show(m3b)`. Part of the output will be a table near the top labeled **Random effects**:

Random effects:

Groups	Name	Variance	Std.Dev.	Corr
--------	------	----------	----------	------

dept	(Intercept)	1.76138	1.32717
	male	0.12804	0.35783 -0.820

You can see the same standard deviation estimates here, as well as their squares under the `Variance` heading. The far righthand column contains the estimate of the correlation between the varying (aka random) effects. So the estimate of ρ is -0.82 . The varying intercepts and slopes are strongly negatively correlated. This implies that departments with high rates of admission have smaller rates of admitting males, while departments with low rates of admission have higher rates of admitting males.

The estimates of σ_α , σ_β , and ρ define the bivariate distribution of intercepts and slopes across departments. But we also have estimates of the α_j and β_j values for each department. To inspect those:

```
ranef( m3b )
```

R code
10.20

\$dept			
	(Intercept)	male	
A	1.7517016	-0.58888628	
B	1.3862674	-0.22203934	
C	-0.1509158	0.20637841	
D	-0.1289359	0.06215337	
E	-0.6379813	0.25487015	
F	-2.2088250	0.29163063	

Scanning this table, you can see that large values of α_j , the first column, are associated with smaller values of β_j , the second column. Note also that the values in the first column are of greater magnitude than those in the second column. This reflects the difference standard deviation estimates of the distributions of these varying effects. As a consequence, the varying effects for `male` contribute less to prediction than do the varying intercepts.

How can you compute predictions from these varying effect estimates? Just use the model formula, as always. The log-odds of admission for any case ij is given by:

$$\log \frac{p_{ij}}{1 - p_{ij}} = \alpha + \alpha_j + (\beta + \beta_j)m_{ij}.$$

The varying effects table above provides the values for α_j and β_j . The values of α and β come from the `precis` output or the `show(m3b)` output. So to compute the expected probabilities of admission for both males and females, using the estimates in this case:

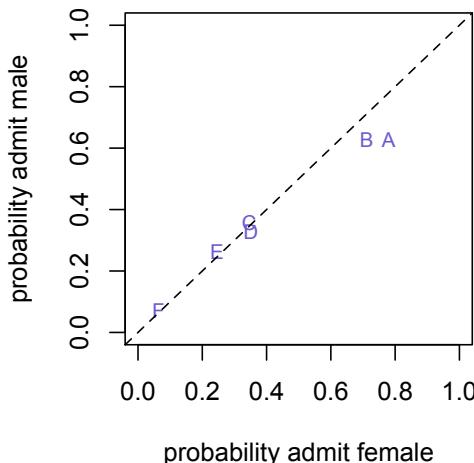


FIGURE 10.3. Predicted probabilities of admission, for male and female applicants. Point labels are department names in the UCB admission data. The dashed diagonal shows the line of no gender bias. Above this line, departments are expected to prefer male applicants. Below it, they prefer female applicants.

R code
10.21

```
aj <- ranef(m3b)$dept[,1]
bj <- ranef(m3b)$dept[,2]
a <- fixef(m3b)[1]
b <- fixef(m3b)[2]
pf <- logistic( a + aj )
pm <- logistic( a + aj + b + bj )
```

Now `pf` contains expected probabilities for female applicants, by department, and `pm` contains expected probabilities for male applicants. If you plot `pf` against `pm`, you get FIGURE 10.3.

10.5.3. What about uncertainty? The package `lme4` is amazingly efficient and flexible. It accomplishes heroic things with tutored maximum likelihood and strategic approximation. But there are limits to what these estimation tactics can accomplish. They don't really provide good estimates of the posterior for varying effects. This is by design, in the case of `lme4`, because as the model gets more complex, it will be harder to trust any naive approximation of the posterior for a variance parameter. It would certainly be possible to write `lme4` so that it produced an approximate standard error for parameters like σ_α , but they would

routinely be misleading, as typically just posteriors are highly asymmetric. You really need to use MCMC to get serious inferences about uncertainty at the level of the varying effects, as well as about variance components.

You can get a little information about the posterior distribution of the varying effects, using the function `se.ranef` in the `arm` package. But even there, you don't get covariances with the other, non-varying, parameters. If you want reliable information about the uncertainty of the per-cluster parameters, as well as the variance components, you really have to use MCMC, I'm afraid.

It's probably better to display the standard errors returned by `se.ranef` than nothing at all. But if you want to simulate predictions for the particular clusters in your data, MCMC is going to make you a lot more comfortable. And even if you just want predictions for entirely new clusters, sampled from the distributions defined by parameters like α and σ_α , you'll still need information on the posterior of σ_α .

Add section on what the Laplace approximation is?

10.6. Wrestling with lemurs

How to use `lme4`'s `lmer` function.

GLM families and default links:

Distribution	GLM family name	default link
Normal	<code>gaussian</code>	<code>identity</code>
Binomial	<code>binomial</code>	<code>logit</code>
Poisson	<code>poisson</code>	<code>log</code>

Varying intercepts. The binomial model:

$$\begin{aligned} y_{ij} &\sim \text{Binomial}(p_{ij}, n_{ij}), \\ \log \frac{p_{ij}}{1 - p_{ij}} &= \alpha + \alpha_j + \beta x_{ij}, \\ \alpha_j &\sim \text{Normal}(0, \sigma_\alpha). \end{aligned}$$

The `lme4` syntax is:

```
m <- glmm( cbind(y,n-y) ~ (1|cluster) + x ,
  data=d , family=binomial )
```

R code
10.22

This model produces estimates for α , β , σ_α , and one α_j for each cluster.

The Gaussian model:

$$\begin{aligned} y_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma), \\ \mu_{ij} &= \alpha + \alpha_j + \beta x_{ij}, \\ \alpha_j &\sim \text{Normal}(0, \sigma_\alpha). \end{aligned}$$

The `lme4` syntax is:

R code
10.23

```
m <- glmm( y ~ (1|cluster) + x , data=d , family=gaussian )
```

This model produces estimates for α , β , σ , σ_α , and one α_j for each cluster.

Varying slopes, constant intercept. The binomial model:

$$\begin{aligned} y_{ij} &\sim \text{Binomial}(p_{ij}, n_{ij}), \\ \log \frac{p_{ij}}{1 - p_{ij}} &= \alpha + (\beta + \beta_j)x_{ij}, \\ \beta_j &\sim \text{Normal}(0, \sigma_\beta). \end{aligned}$$

The `lme4` syntax is:

R code
10.24

```
m <- glmm( cbind(y,n-y) ~ (0+x|cluster) + x ,
            data=d , family=binomial )
```

This model produces estimates for α , β , σ_β , and one β_j for each cluster.

The Gaussian model:

$$\begin{aligned} y_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma), \\ \mu_{ij} &= \alpha + (\beta + \beta_j)x_{ij}, \\ \beta_j &\sim \text{Normal}(0, \sigma_\beta). \end{aligned}$$

Code:

R code
10.25

```
m <- glmm( y ~ (0+x|cluster) + x , data=d , family=gaussian )
```

This model produces estimates for α , β , σ , σ_β , and one β_j for each cluster.

Varying intercepts and slopes. The binomial model:

$$\begin{aligned} y_{ij} &\sim \text{Binomial}(p_{ij}, n_{ij}), \\ \log \frac{p_{ij}}{1 - p_{ij}} &= \alpha + \alpha_j + (\beta + \beta_j)x_{ij}, \\ \begin{pmatrix} \alpha_j \\ \beta_j \end{pmatrix} &\sim \text{Normal}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \rho\sigma_\alpha\sigma_\beta \\ \rho\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix}\right). \end{aligned}$$

The `lme4` syntax is:

```
m <- glmm( cbind(y,n-y) ~ (1+x|cluster) + x ,
  data=d , family=binomial )
```

R code
10.26

This model produces estimates for α , β , σ_α , σ_β , ρ , and one α_j and β_j for each cluster.

The Gaussian model:

$$\begin{aligned} y_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma), \\ \mu_{ij} &= \alpha + \alpha_j + (\beta + \beta_j)x_{ij}, \\ \begin{pmatrix} \alpha_j \\ \beta_j \end{pmatrix} &\sim \text{Normal}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \rho\sigma_\alpha\sigma_\beta \\ \rho\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix}\right). \end{aligned}$$

The `lme4` syntax is:

```
m <- glmm( y ~ (1+x|cluster) + x , data=d , family=gaussian )
```

R code
10.27

This model produces estimates for α , β , σ , σ_α , σ_β , ρ , and one α_j and β_j for each cluster.

10.6.1. Multiple clustering variables. If you have more than one kind of clustering in your data, either because there are groups nested within groups (e.g. students within classrooms within schools) or because there are multiple cross-classifying groups (e.g. repeat observations of individuals across multiple sessions), then you might want to include more than one varying effect grouping variable. This is straightforward, although especially in the case of cross-classified groups, getting good estimates might require some kind of MCMC. For Gaussian models, however, and even in many non Gaussian cases, `lme4` does an amazingly good job of this difficult estimation problem.

Two kinds of varying intercepts. Consider the students within classrooms with schools example. Let i be each student within a classroom, j be the classroom within a school, and k be the school. The binomial model:

$$\begin{aligned} y_{ijk} &\sim \text{Binomial}(p_{ijk}, n_{ijk}), \\ \log \frac{p_{ijk}}{1 - p_{ijk}} &= \alpha + \alpha_{Cj} + \alpha_{Sk} + \beta x_{ijk}, \\ \alpha_{Cj} &\sim \text{Normal}(0, \sigma_{\alpha_C}), \\ \alpha_{Sk} &\sim \text{Normal}(0, \sigma_{\alpha_S}). \end{aligned}$$

The `lme4` syntax is:

R code
10.28

```
m <- glmm( cbind(y,n-y) ~ (1|school/class) + x ,
           data=d , family=binomial )
```

Think of the expression `school/class` as being *classes within schools*. This notation will ensure that you get a unique intercept estimate for each classroom, even if you reuse the same classroom labels across schools. That is, `lme4` is smart enough to know that you don't think classroom "4" in school 2 is the same entity as classroom "4" in school 3. But only if you use the `school/class` syntax. Otherwise, it will assume you really think classroom "4" is the name of a unique entity in the world, which happens to exist in multiple schools. If you instead construct your `class` variable so that each classroom has a unique label across all schools, then you'll get the same estimates with:

R code
10.29

```
m <- glmm( cbind(y,n-y) ~ (1|class) + (1|school) + x ,
           data=d , family=binomial )
```

This model produces estimates for α , β , σ_{α_C} , σ_{α_S} , and one α_{Cj} and α_{Sj} for each cluster. So for example, if there are 20 classrooms in each of 5 schools, then there will be 100 (20×5) varying intercept estimates for classrooms and 5 for schools, for 105 varying intercepts estimated overall.

Even more clustering variables. You can logically extend this approach to deeper levels of hierarchically nested varying effects. For example, if you have repeat observations for each student, then you could add varying intercepts across students.

R code
10.30

```
m <- glmm( cbind(y,n-y) ~ (1|school/class/student) + x ,
           data=d , family=binomial )
```

Again, be careful to construct a variable that provides a unique identifying label for each unique student in the data.

Variable slopes by more than one kind of cluster. You can extend this strategy to varying slopes as well. To allow both intercepts and slopes to vary by classroom and school:

R code
10.31

```
m <- glmm( cbind(y,n-y) ~ (1+x|school/class) + x ,
           data=d , family=binomial )
```

or equivalently:

```
m <- glmm( cbind(y,n-y) ~ (1+x|class) + (1+x|school) + x ,
    data=d , family=binomial )
```

R code
10.32

This second notation, splitting the clustering variables apart, is useful for when you want varying intercepts for both classrooms and schools but varying slopes for only, for example, schools. In that case, you can specify:

```
m <- glmm( cbind(y,n-y) ~ (1|class) + (1+x|school) + x ,
    data=d , family=binomial )
```

R code
10.33

Now you get varying intercepts for both classes and schools, varying slopes for schools, and a correlation estimate for varying intercepts and slopes across schools.

10.6.2. Example: Chimpanzees again and cross-varying effects. Re-estimate chimpanzee model, using varying effects and now being able to include the experiment block:

$$\begin{aligned} L_{ijk} &\sim \text{Binomial}(p_{ijk}, 1), \\ \log \frac{p_{ijk}}{1 - p_{ijk}} &= \alpha + \alpha_{Aj} + \alpha_{Bk} + \beta_P P_{ijk} + \beta_{PC} P_{ijk} C_{ijk}, \\ \alpha_{Aj} &\sim \text{Normal}(0, \sigma_{\alpha_A}), \\ \alpha_{Bk} &\sim \text{Normal}(0, \sigma_{\alpha_B}), \end{aligned}$$

where L_{ijk} is `pulled.left` for case i and actor j and block k , α_{Aj} is the varying effect of actor j , α_{Bk} is the varying effect of experiment block k , and σ_{α_A} and σ_{α_B} are the standard deviations that correspond to each varying effect type. The predictors P_{ijk} and C_{ijk} are `prosoc.left` and `condition`. To load the data and estimate the model:

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
m <- glmm( pulled.left ~ (1|actor) + (1|block)
    + prosoc.left*condition - condition ,
    data=d , family=binomial )
```

R code
10.34

Looking at the non-varying estimates:

R code
10.35`precis(m)`

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	0.3927680	0.7257539	-1.0296835	1.8152195
prosoc.left	0.8190849	0.2593696	0.3107298	1.3274400
prosoc.left:condition	-0.1338483	0.2956853	-0.7133809	0.4456844
((Intercept) actor)	1.8513365	NA	NA	NA
((Intercept) block)	0.0811818	NA	NA	NA

A lot of variation among chimpanzees (**actor**), as you saw in Chapter 8. But not much by experiment block (**block**).

Notice also that if we compare the estimates above to the simple one-level GLM estimates, the effect of the 1/1 option (**prosoc.left**) being on the left increases:

R code
10.36

```
m.constant <- glm( pulled.left ~ prosoc.left*condition - condition ,
  data=d , family=binomial )
coeftab( m , m.constant )
```

	m	m.constant
(Intercept)	0.3928	0.0476
prosoc.left	0.8191	0.6100
prosoc.left:condition	-0.1338	-0.1043
((Intercept) actor)	1.8513	NA
((Intercept) block)	0.0812	NA
nobs	504.0000	504.0000

This is a quite common result. Including varying intercepts can “control” a lot of noise that makes it harder to make inferences about regression effects.

Including dummy variables for each chimpanzee would also work. But as you saw earlier in this chapter, the varying intercepts approach is likely to overfit less, because it pools information across chimpanzees. And in cases in which one cluster (here, chimpanzee) exhibits extreme behavior, the non-pooling dummy variable approach is actually not identifiable, even though R will pretend it is. Here’s the non-pooling model, with a dummy variable for each actor:

R code
10.37

```
m2fe <- glm( pulled.left ~ as.factor(actor)
  + prosoc.left * condition - condition ,
  data=d , family=binomial )
precis(m2fe)
```

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	-0.7266	0.2686	-1.2531	-0.2001
as.factor(actor)2	18.9491	754.9798	-1460.7842	1498.6824
as.factor(actor)3	-0.3050	0.3500	-0.9910	0.3811
as.factor(actor)4	-0.3050	0.3500	-0.9910	0.3811
as.factor(actor)5	0.0000	0.3440	-0.6743	0.6743
as.factor(actor)6	0.9395	0.3489	0.2557	1.6233
as.factor(actor)7	2.4837	0.4507	1.6003	3.3670
prosoc.left	0.8224	0.2613	0.3102	1.3345
prosoc.left:condition	-0.1324	0.2972	-0.7149	0.4501

Look at the second estimate, the intercept for chimpanzee number 2. The log odds are very large, ensuring essentially 100% chance of pulling left, and the standard error is massive, leading to an essentially infinite confidence interval. Indeed, this is an non-identifiable estimate. But with pooling, it becomes identifiable, because the behavior of the other individuals provides information we can use to optimally adjust the estimate for actor 2. The substantive inference doesn't change in this case, but at least with the varying effect approach, you won't find yourself trying to explain why you can trust the estimates of a model that failed to identify one or more of its parameters.

10.7. The importance of variance priors

A very common result of fitting a multilevel model with the `lme4` library, or rather fitting with pure maximum likelihood or an approximation of it, is one or more estimates of zero for the between-cluster variance parameter(s). As a consequence, all of the varying effect estimates themselves, the α_j values, will also be zero. While such estimates are rarely published, I've encountered them in my own analyses many times. And I frequently entertain inquiries from colleagues who encounter such bizarre estimates and wonder if they have done something wrong.

Assigning a value of zero to the between-cluster variance is an odd result. Do the data really support the inference that there is no variation among clusters? In most cases, no. Instead maximum likelihood is getting fouled up near the zero boundary for variance parameters. But in this section, I'll show you an easy way to *regularize* inference about the variance parameters and nudge the estimate off of the zero boundary. Indeed, I will also show you that this nudging will produce better inferences. And we're going to achieve that improvement by using a weakly informative prior for the variance parameters.

Let's look at a classic example of the problem itself.¹⁰⁶ Here's the code to load the data, which is contained in the `lme4` package already, and fit the varying intercept model to it:

R code
10.38

```
library(lme4)
data(Dyestuff2)
d <- Dyestuff2
m <- glmm( Yield ~ (1|Batch) , data=d , family=gaussian )
precis(m)
```

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	5.665600	0.6669857	4.358332	6.972868
((Intercept) Batch)	0.000000	NA	NA	NA
(Residual)	3.653231	NA	NA	NA

These data are concentrations of something called “dyestuff,” which is used as a pigment. Different batches produce different average amounts of the product. There is variation within each batch as well as variation among the batches. And so a multilevel model is a natural choice for analysis. However, as you can see in the output above, the estimate for the standard deviation among batches, σ_α , is zero. All of the variation in the data has been assigned to the within batch variation, σ . If you inspect the α_j estimates in `ranef(m)`, you'll see that they are all zero, too.

This sort of result is actually quite common, whenever the variation between clusters is small relative to the variation within clusters. There is certainly variation among the clusters, but maximum likelihood gets too close to the zero boundary when contemplating the estimate for σ_α . Indeed, some classical estimators for these parameters will produce negative estimates, if the software allows it.

The variation among clusters is certainly not zero, and we'd like our estimates to reflect that fact. We'd like to *regularize* inference in order to cope with the distorted estimates maximum likelihood can produce near a boundary. There are several good ways to accomplish such regularization. One of the easiest and logically transparent approaches is to use a prior probability density for the variance parameters, like the σ and σ_α of our simple varying intercepts model. What we desire is a prior that is exactly zero when $\sigma_\alpha = 0$, but does not dominate the likelihood elsewhere.

A good choice is a gamma prior.¹⁰⁷ You shouldn't use just any old prior, because you also want to ensure that probability increases smoothly from zero, instead of there being some kind of dead zone at $\sigma_\alpha = 0$. A

gamma prior satisfies this criterion, provided you parameterize it correctly. A good choice is setting the gamma’s “shape” (α) to 2 and its “rate” (λ) to 10^{-4} or so. This is equivalent to setting the distribution’s mean to 20-thousand and its scale (s) to 10-thousand. This will produce a very nearly flat prior that nevertheless rules out estimates of zero and very weakly favors larger estimates. The likelihood will still dominate inference, but the prior will be able to nudge inference off of zero.

The reward for all of this bother is better, more accurate inference about both the variance components, like σ_α , but also about all of the varying effects, which depend upon them. In the next chapter, I’ll show you how to write this prior into a Markov chain, but if you want to stick with `lme4`, and indeed often that’s all you need, then the R package `blme` will amend its estimation procedure to use various weakly informative priors and regularize inference about variance components.

Here’s how you can do it. Make sure you’ve installed `blme` from CRAN. Then you can use the convenience function `bglmm` in the `rethinking` package or invoke `bglmer` directly. The syntax is the same either way. Using `bglmm`:

```
mb <- bglmm( Yield ~ (1|Batch) , data=d , family=gaussian )
precis(mb)
```

R code
10.39

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	5.662383	0.8854168	3.926998	7.397768
((Intercept) Batch)	1.471597	NA	NA	NA
(Residual)	3.564682	NA	NA	NA

That’s better. You can inspect the varying effects with `ranef(mb)`. They are no longer all zero, although they are modest in size, because there isn’t a lot of variation among batches.

What has happened to the inferred posterior for σ_α can be seen in FIGURE 10.4. The plot displays the posterior density estimate produced from samples from the posterior, both with a flat prior (black) and the weakly informative gamma prior (blue). I produced these samples using MCMC, which you’ll learn how to do in the next chapter. What I want the reader to notice is just how much of an effect the very weak prior can have. Even though the likelihood dominates the prior everywhere except at $\sigma_\alpha = 0$, only a tiny nudge is needed to send inference of into another conformation of the parameters. This produces substantially different, non-zero, inferences for σ_α and the varying effects α_j .

But how do we know that these new estimates are better? Well, the unsatisfying answer is that theory tells us it is. But many readers might

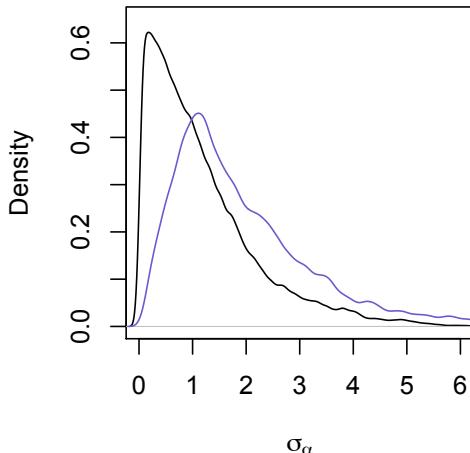


FIGURE 10.4. The impact of a weakly informative prior on the posterior density of σ_α , the standard deviation among batches. The plot compares the posterior produced using a flat prior (black) to the posterior produced using a weakly informative prior (blue).

want a more direct demonstration. In order to provide that, I'm going to simulate multilevel data that produces this pathology and then compare the accuracy of the varying effect estimates produced by `glmm` (or `glmer`) to those produced by the function that uses weakly informative priors, `bglmm` (or `bglmer`).

You can manufacture data that will produce zero variance estimates quite easily. All you really need is for the variation among clusters to be much smaller than the variation within them. A ratio of 1:4 will serve to illustrate the phenomenon. Here's a function to simulate dyestuff-style data:

R code
10.40

```
simdye <- function( nj=6 , sigma=sqrt(20) , sigma.a=sqrt(5) ) {
  nij <- 6
  aj <- rnorm( nj , 0 , sigma.a )
  a <- rep( aj , each=nij )
  y <- rnorm( nj*nij , a , sigma )
  d2 <- data.frame( y=y , cluster=rep(1:nj,each=nij) )
  m0 <- glmm( y ~ (1|cluster) , data=d2 , family=gaussian )
  m1 <- bglmm( y ~ (1|cluster) , data=d2 , family=gaussian )
  m0.re <- ranef(m0)$cluster[,1] + fixef(m0)[1]
  m1.re <- ranef(m1)$cluster[,1] + fixef(m1)[1]
  m0.err <- abs( m0.re - aj )
  m1.err <- abs( m1.re - aj )
  c( mean( m0.err ) , mean( m1.err ) )
```

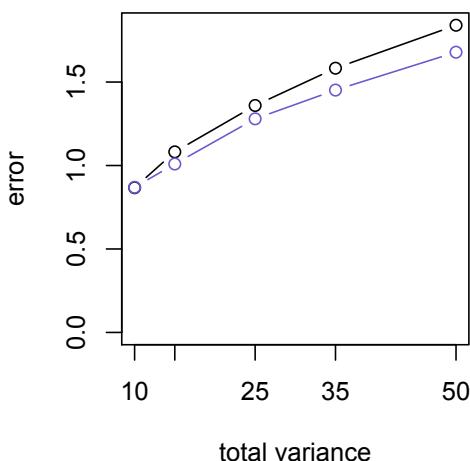


FIGURE 10.5. The advantage of weakly informative priors. The vertical axis is the average estimation error of the varying intercepts, α_j . The horizontal axis displays the total variation, $\sigma^2 + \sigma_\alpha^2$. The black trend is the estimation error for the model with flat priors. The blue trend is error for the model with weakly informative priors for σ and σ_α .

}

This function, by default, will simulate 6 observations for each of 6 batches, using $\sigma = \sqrt{20}$ and $\sigma_\alpha = \sqrt{5}$. Thus the variation between batches is a fifth of that within batches. The function then returns the average error of the varying effect estimates produced by `glmm` to those produced by `bglmm`. Not every simulation will produce the zero-estimate phenomenon, but many will. When `glmm` does not produce a zero estimate, then inference from both methods is essentially the same. But when there is a zero estimate, inference can be rather different, as you've already seen.

In FIGURE 10.5, I display the results of using this function to produce 100 simulations at different values of the overall variation in the data. The vertical axis is the average estimation error of the varying effect estimates, across simulations at each value on the horizontal axis. The horizontal axis displays the total variation in the data-generating model, $\sigma^2 + \sigma_\alpha^2$. The black trend is the estimation error for the model with flat priors, fit with `glmm/glmer`. The blue trend is error for the model with weakly informative priors for σ and σ_α , fit with `bglmm/blmer`. The informative prior regularizes inferences, improving the estimates, and therefore the blue trend is below the black trend.

Here's the rest of the code needed to produce the figure:

R code
10.41

```
f2 <- function(m) {
  z <- replicate(100,
    simdye(sigma=sqrt(m*4/5), sigma.a=sqrt(m/5)) )
  apply(z, 1, mean)
}
vlist <- c(10,15,25,35,50)
z <- sapply(vlist, function(zz) f2(zz) )
plot(z[1,] ~ vlist, type="b", ylab="error",
  xlab="total variance", ylim=c(0,max(z)), xaxt="n" )
axis(1, at=vlist, labels=vlist)
lines(vlist, z[2,], col="slateblue", type="b" )
```

When we have information, such as that the variances are never zero, then getting that information into the analysis can improve inference.

10.8. Centering Strikes Back

Grand mean versus group mean issues.

10.9. How many parameters?

Throughout this chapter, I have yet to mention comparing the multilevel models, using AICc or BIC. One reason to omit this issue so far is that there was plenty to learn without mixing it in. But the real reason I've left this important issue until the end is that it is suddenly much more complicated, once a model contain multiple levels of parameters.

Now, the truth is that you have to more clearly define what sort of predictions or inference you want to make, before you can decide how to compare the models. Indeed, it is now not even easy to answer how many parameters a model has.

It'll help to consider the issue in the context of an example. Take another look at the multilevel tadpole model we started the chapter with.

$$\begin{aligned} y_i &\sim \text{Binomial}(p_i, n_i), \\ \log \frac{p_i}{1-p_i} &= \alpha + \alpha_i, \\ \alpha_i &\sim \text{Normal}(0, \sigma). \end{aligned}$$

Now ask how many parameters this model has. The top-level parameters are easy to count: α and σ are two parameters. But what about the 48 α_i varying intercepts? Aren't these 48 more parameters, for a total of 50?

If the goal is to compute the penalty term for AICc or BIC, then the answer is absolutely not. Or rather, there are 50 parameters, but

the 48 varying effects don't each count as a full parameter, in the context of AICc/BIC. This might make more sense, if you consider that what is relevant to the AICc/BIC penalties is not exactly the number of parameters, but rather some measure of how flexible the model family is. More flexibility means more overfitting. When models have only one stochastic node, every parameter is estimated the same way, and so under certain conditions (upon which AICc and BIC depend), a good estimate of the relevant model family flexibility is just the number of parameters.

But in a multilevel model family, who batches of parameters are varying effects that are estimated jointly, through pooling. The top-level distributional assumptions define how the pooling works, but once pooling is in operation, each of $48 \alpha_i$ parameters in the tadpole model isn't as flexible as the kinds of parameters you have come to know from previous chapters. As a result, a multilevel model many need a different kind of approximation, in order to estimate the expected out-of-sample deviance.

But it's also sometimes useful to compare multilevel models by ignoring all of the varying effect parameters and just counting the non-varying effects, like the α and σ and any β parameters. Other times, you have to pay attention to the varying effects and try to "count" them correctly. The way to decide which situation you are in is to try to address the question of the *focus* of your predictions.

10.9.1. Focus. The question of *focus* is: Which parameters are relevant to the kinds of predictions you want to make? What do you want to do with your estimates? In the context of multilevel regression models, there are usually two sorts of generalization targets. First, you might want to generalize to the same clusters present in your sample. Second, you might want to generalize to a new sample of clusters, none of which are present in your sample. Let's consider each of these in turn.

Same clusters. Suppose the tadpole tanks are actually ponds, as you imagined earlier in the chapter, when I tried to explain why varying effect estimates can be more accurate than non-varying estimates. These imaginary ponds may have real persistence in the world, and so you may want to generalize to future tadpole populations in them. In that case, you really care about the specific varying effect estimates, and you care about keeping them identified with the real world clusters they represent.

This kind of intention for a multilevel model arises in many contexts. For example, if the clusters are States within the U.S.A., then they are

rather persistent entities. Chances are any inference and predictions you want to draw from the model will pertain to these same clusters, the same 50 States.

New clusters. Now suppose that you don't actually care about those specific tadpole ponds. Instead you want to predict to brand new ponds, or rather generalize to a much broader population of ponds than those in your sample. You think the new ponds are members of the same population of variable intercepts, and so you can use your α and σ estimates to calibrate your predictions about them. But the specific α_i estimates you have made are useless to you, because you won't want to predict anything for those ponds.

Indeed, in the actual tadpole data, the “ponds” are just aquariums. They don't have any persistence, beyond the experiment. The α_i estimates you made have helped your inference about other, non-varying regression parameters. But in themselves they aren't likely to be useful to you.

Same and new clusters. It's even possible that there are some cluster types in your model that have persisting importance to you, while other cluster types do not. Consider again the 50 States example. If we have data on individuals within counties within States, it's possible to desire to generalize to the same 50 States—there aren't any others, after all—while not being interested in prediction for the same specific counties.

Or, to go yet one level deeper, if you have repeat observations on the individuals themselves, then you have three clustering variables: individuals, counties, States. Even if you care about the persistent identities of the counties and States, you probably don't expect to make predictions for these exact same individuals ever again. They are a tiny sample of the State populations they are drawn from, so usually you'd prefer to generalize inference beyond their sample. In contrast, it wouldn't make sense to say you're going to generalize inference about each State to the population of States they were drawn from.

10.9.2. Think about cross-validation. And so we end up having to decide our focus before we can evaluate a strategy for penalizing models for complexity. It might help to explain a way forward now, by returning to the cross-validation approach that was briefly described in Chapter 5. Recall that I demonstrated the characteristic difference between underfit (biased) and overfit (high variance) models by sequentially deleted each data point from The Zipper example. When you evaluate a model

on its ability to predict omitted data, rather than its ability to fit data, you are cross-validating.

It is easier to see a way forward, with cross-validation, than it is with AIC at the moment. A very common and well-studied form of cross-validation is *leave-one-out*, which is actually what I illustrated in Chapter 5. It turns out that, in a sufficiently large sample, AIC and leave-one-out cross-validation produce the same estimate of out-of-sample deviance.¹⁰⁸ When you leave out each individual case (row), one at a time, fit the model and then predict the omitted case, you are testing the model's accuracy out of sample. The average deviance of the omitted cases, each predicted using estimates trained on all of the other cases, aims at the same basic criterion as AIC does.

So how would cross-validation work in the context of a multilevel model? It depends upon your focus. Suppose for example that you do care about the persistence of the clusters in your sample. You want to make future predictions of survival in the tadpole ponds, for example. Now, the individual tadpoles you train the model on aren't the target of your prediction. But the characteristics of each pond are. So how would you develop a cross-validation strategy, to see how well your model does out-of-sample?

A good approach might still be to leave out each individual case (tadpole) in the sample and try to predict it, using estimates produced in its absence. This procedure evaluates the usefulness of your varying effects, as they will be needed to make predictions for any individuals in the same ponds. Call this *leave-one-case-out* cross-validation.

But if instead you don't care about those ponds, then it doesn't make sense to evaluate your model on the basis of the specific varying effect estimates for the sample. Instead, you'd want to evaluate the other estimates, the top-level α and σ and such that describe the population of ponds your sample came from. So how would you cross-validate in this context? Leave out each *pond*, one at a time, and predict the omitted pond. In order to make a good prediction, you'll need good estimates of α and σ and any regression parameters otherwise. But the varying effects of the sample are irrelevant to your inference (aside from the fact that they helped you get better estimates of the top-level parameters). Call this procedure of leaving out each pond, one by one, *leave-one-cluster-out* cross-validation.

10.9.3. AIC is like leave-one-cluster-out. Both leave-one-case-out and leave-one-cluster-out strategies are useful, depending upon your focus. And if you have the time and programming skill, you can do both and probably learn something extra from any differences in their estimates.

But you are probably after a metric like AIC, that won't require all the fuss of cross-validation.

You might think that AIC is equivalent to leave-one-case-out cross-validation, because in the context of a single level model, the two methods are large-sample equivalent. But in the context of multilevel models, they are not. Instead, AIC is more similar to leave-one-cluster-out.¹⁰⁹ The reason is that AIC's derivation doesn't see any levels deeper than the most clustered description of the data. What is being left out and predicted are chunks of interdependent observations. In other words, they are more like the ponds than the tadpoles. As a result, under favorable conditions, AIC (and AICc) provide useful estimates of leave-one-cluster-out cross-validation, which is appropriate when you don't care to generalize to any of the actual clusters in your sample.

But AIC will not provide as good an estimate of leave-one-case-out cross-validation. Instead, you need a new metric that takes account of the varying effects, gazing deeper into the structure of the model.

10.9.4. DIC. This area of research is still relatively young, and so there aren't as many choices as there are for single-level models.¹¹⁰ But there is one that is becoming increasingly common, because it was built into a popular piece of software: DIC, the deviance information criterion.

DIC is an attempt to estimate the leave-one-case-out criterion, using samples from the posterior of a multilevel model.¹¹¹ In other words, it is the multilevel analog of AIC. In fact, you can compute DIC for a single-level model, and its expected value will be the same as AIC. When I was learning about DIC, I did simulations of that kind, to prove it to myself.

Unfortunately, the origins of the metric are more obscure than the metric itself, once it was available as part of WinBUGS. Many people know it, and write about it in the literature, as a "Bayesian" alternative to AIC. This is only true in the weakest sense. If DIC is "Bayesian," then so is AIC, if by "Bayesian" we mean consistent with a Bayesian interpretation of model inference as using probability densities of parameters to make inferences about models. Indeed, since AIC (but not quite AICc, as AICc is a corrected estimate) is a special case of DIC, when there are no varying effects.

But DIC is indeed "Bayesian" in the sense that it is typically computed from samples from the posterior of a model fit via MCMC. You could compute AIC the same way, but it is hardly necessary, in most cases.

So can you compute DIC already, for the multilevel models in this chapter? You can get estimates of it, yes, using the `extractDIC` function in the `arm` package. But since models fit with `lme4` don't really have full

posterior density information, at least not in the detail you'd prefer, I'm going to punt on the formal definition of DIC and how to estimate it, until the next chapter. At that point, I'll show you how to compute DIC directly from samples from the posterior.

But even more, I will explain how to use MCMC to sample model families, a technique sometimes known as *reversible-jump* MCMC (RJMCMC). RJ-MCMC isn't easy, and often it can't be made to work at all. But when it can, it verifies that the posterior probability estimates of model families provided by metrics like AIC, BIC, and DIC are really doing what they claim. They aren't always perfect. But then again, neither are the models.

10.9.5. So can I use AIC to evaluate multilevel models or not? There are two basic contexts in which it is not terrible to use AIC (or AICc) to compare multilevel models.

First, you can use AIC (or better, AICc) to compare multilevel models when you are sure you don't care about generalizing to the same clusters. In that case, the relevant "number" of parameters is the count of non-varying parameters in the model. This is indeed how `lme4` computes the AIC value it displays.

But even under this favorable focus, AICc may not provide nearly as good an estimate as it would for a single-level model. Reasonable people can disagree about what the best course of action is. We have to adjust for overfitting, somehow. Is it better to use a flawed, possibly high variance, estimate to do so, leading to possibly overly-conservative and unreliable inferences? Or is it better instead to remain agnostic and have no estimate of overfitting? We almost always need to shrink regression parameter estimates towards zero, and model comparison is a logically coherent way to accomplish that. But there's no guarantee that it's going to be easy, in any particular context.

Second, if all of the models you are comparing have the exact same multilevel structure—the same clusters and varying effect definitions—then you can usefully compare them with AIC. This is because, whatever the additional penalty for the varying effect parameters, it is constant across all of the models, and therefore subtracts out of the comparisons. So if you can usefully constrain yourself to always include the same varying intercepts and slopes, and sometimes you can, then AIC (AICc) isn't actively misleading you.

Much of the time, neither of the above situations holds. So what can we do? What we can all agree upon is that DIC (or any measure like it), if we had a way to compute it accurately, will never count each varying effect parameter as a whole parameter, in terms of the relevant

penalty. Indeed, often each doesn't even count as half a parameter. So if you compute a kind of "hostile" AIC that assumes each varying effect parameter is a whole parameter, you are conducting a very conservative analysis. If the more complex model is still much better, after applying the hostile penalty, then you can be rather sure that the multilevel model is superior, in terms of out-of-sample prediction.

None of the above answers may be reassuring to you. I'm afraid the world is a very uncertain place, and model comparison metrics are still relatively young. In time, perhaps we'll have better ways to compare model families. But for now, it does no good to sell a false certainty about our methods. Instead, embrace the uncertainty and honestly communicate it with your colleagues.

Comfortingly, statistics is not a substitute for science. Our statistical methods do not have to solve all problems of inference, nor could they. But when we try to force statistical methodology to produce certainty where none is forthcoming, than statistics can actually damage science.

Maybe add simulations here to demonstrate the points? In that case, probably enough material to add another chapter, after MCMC chapter, that goes into more depth, explain DIC and focus etc., and gives a simple example of RJ-MCMC code.

11 Markov Chain Monte Carlo Estimation

11.1. Fortuna Strikes Back

For most of human history, chance has been a villain. In classic Roman civilization, chance was personified by Fortuna, goddess of cruel fate, with her ever spinning wheel of luck. Opposed to her sat Minerva, goddess of wisdom and understanding. Only desperate would pray to Fortuna, while everyone implored Minerva for aid. Certainly science was the domain of Minerva, a realm with no useful role for Fortuna to play.

But by the beginning of the 20th century, the opposition between Fortuna and Minerva had changed to a collaboration. Scientists, servants of Minerva, began publishing books of random numbers, instruments of chance to be used for learning about the world. Now, chance and wisdom share a cooperative relationship, and few of us are any longer bewildered by the notion that an understanding of chance could help us acquire wisdom. Everything from weather forecasting to finance to evolutionary biology is dominated by the study of stochastic processes.

This chapter introduces one of the more marvelous examples of how Fortuna and Minerva cooperate: the estimation of posterior probability densities using a random process known as Markov Chain Monte Carlo (MCMC) estimation. Unlike in every earlier chapter in this book, here we'll produce samples from the joint posterior of a model without maximizing anything. Instead of having to lean on quadratic and other approximations of the shape of the posterior, now we'll be able to sample directly from the posterior without assuming any regular shape for it. The cost of this power is that it will take much longer for our estimates to finish, and usually more work is required to specify the model as well. But the benefit is escaping the awkwardness of assuming multivariate normality—as we did in most of the book and indeed most people do in their statistics without ever realizing it. Equally important is the ability

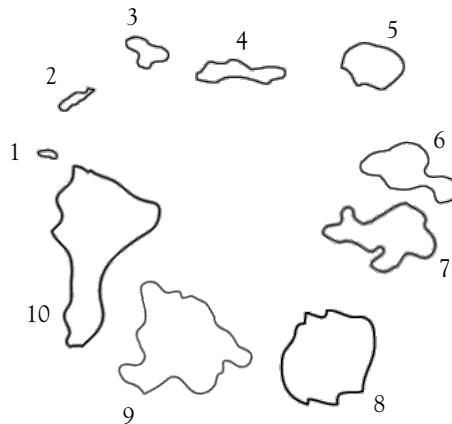


FIGURE 11.1. Good King Markov's island kingdom.

to estimate models, such as the multilevel models of the previous chapter, even when we cannot write a single solved likelihood function for the model. This will turn out, as a colleague of mine says, to save our bacon.

11.2. Good King Markov and His Island Kingdom

For the moment, forget about posterior densities and MCMC. Consider instead the tale of Good King Markov. King Markov was a benevolent autocrat of an island kingdom, a circular archipelago, with 10 islands. Each island was neighbored by two others, and the entire archipelago formed a ring. The islands were of different sizes, and so had different sized populations living on them. The second island was about twice as populous as the first, the third about three times as populous as the first, and so on, up to the largest island, which was 10-times as populous as the smallest. The good king's island kingdom is displayed in FIGURE 11.1, with the islands numbered by their relative population sizes.

The good king was an autocrat, but he did have a number of obligations to his people. Among these obligations, King Markov agreed to visit each island in his kingdom from time to time. Since the people love their king, each island would prefer that he visit them more often. And so everyone agreed that the king should visit each island in proportion to its population size, visiting the largest island 10-times as often as the smallest, for example.

The Good King Markov, however, wasn't one for schedules or book-keeping, and so he wanted a way to fulfill his obligation without planning his travels months ahead of time. Also, since the archipelago was a ring, the King insisted that he only move among adjacent islands, to minimize time spent on the water—like many citizens of his kingdom, the king believes there are sea monsters in the middle of the archipelago. The king's advisor, a Mr Metropolis, engineered a clever solution to these demands. We'll call this solution the *Metropolis algorithm*. Here's how it works.

- (1) Wherever the King is, each week he decides among staying put for another week or moving to one of the two adjacent islands. To decide his next move, he flips a coin.
- (2) If the coin turns up heads, the King considers moving to the adjacent island clockwise around the archipelago. If the coin turns up tails, he considers instead moving counter-clockwise. Call the island the coin nominates the *proposal island*.
- (3) Now, to see whether or not he moves to the proposal island, King Markov counts out a number of seashells equal to the relative population size of the proposal island. So for example, if the proposal island is number 9, then he counts out 9 seashells. Then he also counts out a number of stones equal to the relative population of the current island. So for example, if the current island is number 10, then King Markov ends up holding 10 stones, in addition to the 9 seashells.
- (4) When there are more seashells than stones, King Markov always moves to the proposal island. But if there are fewer shells than stones, he places all of the shells and stones in a bag. Then he reaches in and randomly pulls out one object. If it is a shell, he moves to the proposal island. Otherwise, he stays put another week. As a result, the probability that he moves is equal to the number of shells divided by the number of stones.

This procedure may seem baroque and, honestly, a bit crazy. But it does work. The king will appear to move around the islands randomly, sometimes staying on one island for weeks, other times bouncing around without apparent pattern. But in the long run, this procedure guarantees that the king will be found on each island in proportion to its population size.

You can prove this to yourself, by simulating King Markov's journey. Here's a short piece of code to do this, storing the history of the king's island positions in the vector **positions**:

R code
11.1

```

num.weeks <- 10000
positions <- rep(0,num.weeks)
current <- 10
for ( i in 1:num.weeks ) {
  # record current position
  positions[i] <- current

  # flip coin to generate proposal
  proposal <- current + sample( c(-1,1) , size=1 )
  # now make sure he loops around the archipelago
  if ( proposal < 1 ) proposal <- 10
  if ( proposal > 10 ) proposal <- 1

  # move?
  prob.move <- proposal/current
  current <- ifelse( runif(1)<prob.move , proposal , current )
}

```

I've added comments to this code, to help you decipher it. The first three lines just define the number of weeks to simulate, an empty history vector, and a starting island position (the biggest island, number 10). Then the `for` loop steps through the weeks. Each week, it records the king's current position. Then it simulates a coin flip to nominate a proposal island. The only trick here lies in making sure that a proposal of "11" loops around to island 1 and a proposal of "0" loops around to island 10. Finally, a random number between zero and one is generated (`runif(1)`), and the king moves, if this random number is less than the ratio of the proposal island's population to the current island's population (`proposal/current`).

You can see the results of this simulation in FIGURE 11.2. The left-hand plot shows the king's location across the first 200 weeks of his simulated travels. As you move from the left to the right in this plot, the points show the king's location through time. The king travels among islands, or sometimes stays in place for a few weeks. This plot demonstrates the seemingly pointless path the Metropolis algorithm sends the king on. The righthand plot shows that the path is far from pointless, however. The horizontal axis is now islands (and their relative populations), while the vertical is the number of weeks the king is found on each. After the entire 10-thousand weeks of the simulation, you can see that the proportion of time spent on each island converges to be almost exactly proportional to the relative populations of the islands.

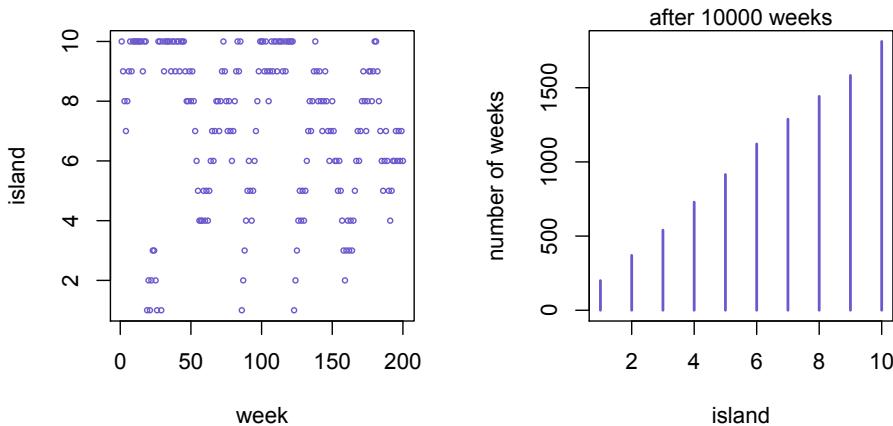


FIGURE 11.2. Results of the king following the Metropolis algorithm. The lefthand plot shows the king’s position (vertical axis) across weeks (horizontal axis). In any particular week, it’s nearly impossible to say where the king will be. The righthand plot shows the long-run behavior of the algorithm, as the time spent on each island turns out to be proportional to its population size.

The algorithm will still work in this way, even if we allow the king to be equally likely to propose a move to any island from any island, not just among neighbors. As long as King Markov still uses the ratio of the proposal island’s population to the current island’s population as his probability of moving, in the long run, he will spend the right amount of time on each island. The algorithm would also work for any size archipelago, even if the king didn’t know how many islands were in it. All he needs to know at any point in time is the population of the current island and the population of the proposal island. Then, without any forward planning or backwards record keeping, King Markov can satisfy his royal obligation to visit his people proportionally.

11.2.1. The Metropolis algorithm. The precise algorithm you used above is a special case of the general *Metropolis algorithm* from the real world.¹¹² In real applications, the goal is of course not to help an autocrat schedule his journeys, but instead to draw samples from an unknown and usually complex target distribution. The “islands” in our objective are parameter values, and they need not be discrete, but can instead take

on a continuous range of values as usual. The “population sizes” in our objective are the posterior probabilities at each parameter value. The “weeks” in our objective are samples taken from the joint posterior of the parameters in the model. Provided the way we choose our proposed parameter values at each step is symmetric—so that there is an equal chance of proposing from A to B and from B to A—then the Metropolis algorithm will eventually give us a collection of samples from the joint posterior. We can then use these samples just like all the samples you’ve already used in this book.

The Metropolis algorithm is an example of a class of algorithms sometimes known as Markov chain Monte Carlo (MCMC). These algorithms describe the state of a system with one or more parameter values. Think of these values as describing where in parameter space the system sits. Then at each step the system can jump to another state. The probability it jumps to any other combination of parameter values is solely a function of the current state, and this is one of the features that makes the algorithm a Markov chain.

The Monte Carlo moniker arises from the essential random (gambling) approach the algorithm takes. Instead of analytically describing the posterior density now, we’re going to use random numbers to explore it, spending more time in regions of high posterior probability and less time in regions of low posterior probability. Provided we take enough samples from the distribution, we’ll end up with an excellent approximation of it.

Why use MCMC. There are a few good reasons to use such a clunky algorithm. First, when grid approximation (remember that, from Chapter 2 and Chapter 4?) is too computationally intensive or awkward, MCMC excels. It provides a messier approximation than grid approximation would, but it can provide a very good approximation in much less computer time. MCMC is especially useful in this way when our model has many parameters, or *dimensions*, to describe the posterior over. Grid approximation becomes practically impossible in such circumstances, while MCMC suffers much less as the dimensionality of the model increases. So instead of using MLE and assuming a multivariate normal “naive” posterior, we can use MCMC to sample from a high-dimension posterior with non-Gaussian shape.

The second and extreme value of the MCMC approach arises when it’s not possible to derive a single analytical expression for the likelihood component of a model. This arises very commonly for complex multilevel models, in which the integrals required to compute posterior probabilities cannot be solved analytically. It also arises for data

that are highly interdependent, such as distance matrixes, phylogenetic trees, and social networks. It can be very difficult to derive even approximate likelihood functions for the full model in these cases. But usually it is possible to define the likelihood at each level of the model—for each stochastic node, for example—and therefore use MCMC to step through the parameters one at a time.

Reestimating Polynesian islands and tool complexity. It will help to provide a data analysis example, both to illustrate the translation from the King Markov allegory and to show you how to include more than one parameter in the algorithm. When there is more than one parameter in the model, the Metropolis algorithm simply considers each in turn, making a proposal for it and accepting or rejecting that proposal, independent of all of the other parameters. The posterior probability at any point of course may depend upon all of the parameters, but only the parameter under consideration at that point can change value. This will make more sense, once you see the working code.

The model we'll estimate is the same one from Chapter 8:

$$T_i \sim \text{Poisson}(\lambda_i), \\ \log \lambda_i = \alpha + \beta_P \log P_i + \beta_C C_i,$$

where T is the outcome variable `Total.Tools`, P is the population size, and C is a 0/1 variable indicating high contact rate. First, load the data and make the necessary dummy variable for contact rate:

```
library(rethinking)
data(islands)
d <- islands
d$log.pop <- log(d$Population)
d$contact.hi <- ifelse( d>Contact=="high" , 1 , 0 )
```

R code
11.2

Now I'm going to define a specialized log-likelihood function for our model, so we can call it over and over again within the Markov chain, without fuss.

```
LL.islands <- function( a , bp , bc ) {
  sum( dpois( d>Total.Tools ,
    lambda=exp(a + bp*d$log.pop + bc*d$contact.hi ) ,
    log=TRUE ) )
}
```

R code
11.3

This function just accepts values for the parameters a , bp , and bc and then returns the log-likelihood of the data.

Now to set up storage for the samples. We'll perform 100-thousand samples from the chain. Don't worry, with only 10 rows in the data and only three parameters, this will be very fast. Here is the code to define the number of samples and to create three empty vectors, each 100-thousand in length, to eventually hold all our samples.

R code
11.4

```
n.samples <- 100*1000
samples.a <- rep(0,n.samples)
samples.bp <- rep(0,n.samples)
samples.bc <- rep(0,n.samples)
```

You absolutely want to initialize these empty vectors before you start sampling, rather than starting with short vectors and increasing their length as you go. R will move much more slowly, if it has to grow the length of a vector, than if it just has to assign a value to a position within an existing vector. So always make the vectors as long as you'll eventually need them to be, as above.

Now for the last bit of initialization before we start sampling, we'll assign starting values to the parameters. This is where the Markov chain will begin, before it starts wandering around the parameter space. If you have good guesses for the high-density region, you should use those guesses here. This is really much like choosing starting values for maximum likelihood search, although later in this chapter I'll also recommend a somewhat different strategy, designed to help you make sure your Markov chain is working as intended.

R code
11.5

```
a <- log( mean(d$Total.Tools) )
bp <- 0
bc <- 0
step <- 1/10
```

The final line above, which assigns a value of 0.1 to `step`, defines how we'll choose proposal values for the parameters. You'll see how it's used in the code to come.

Now for the bulk of the code. This is going to be a big chunk, but I'll explain each piece of it, afterwards. Really, it just repeats the same kind of sampling code three times, once for each parameter, α (`a`), β_P (`bp`), and β_C (`bc`).

```

s <- Sys.time()
for ( i in 1:n.samples ) {

  # record samples
  samples.a[i] <- a
  samples.bp[i] <- bp
  samples.bc[i] <- bc

  # sample a
  a.proposal <- rnorm( 1 , a , step )
  LL.current <- LL.islands( a , bp , bc )
  LL.proposal <- LL.islands( a.proposal , bp , bc )
  pr.accept <- exp( LL.proposal - LL.current )
  a <- ifelse( runif(1) < pr.accept , a.proposal , a )

  # sample bp
  bp.proposal <- rnorm( 1 , bp , step )
  LL.current <- LL.islands( a , bp , bc )
  LL.proposal <- LL.islands( a , bp.proposal , bc )
  pr.accept <- exp( LL.proposal - LL.current )
  bp <- ifelse( runif(1) < pr.accept , bp.proposal , bp )

  # sample bc
  bc.proposal <- rnorm( 1 , bc , step )
  LL.current <- LL.islands( a , bp , bc )
  LL.proposal <- LL.islands( a , bp , bc.proposal )
  pr.accept <- exp( LL.proposal - LL.current )
  bc <- ifelse( runif(1) < pr.accept , bc.proposal , bc )

  # display progress
  progbar( i , min=0 , max=n.samples , starttime=s )
}


```

R code
11.6

Okay, start at the top. The first line uses `Sys.time()` to make a time stamp of when the chain begins. We'll use this to time the process. Then the loop begins. The first block of code, commented `# record samples`, just saves the state of chain at that point.

Then the next block of code completes a Metropolis step for the parameter α (`a`). I'll walk you through this block of code, one line at a time, so you can follow each step to the Metropolis algorithm. The first line in this block creates a random proposal value for α to jump to. It uses the most common approach, sampling the proposal from a

normal distribution with mean at the current value, $a(\alpha)$, and a narrow standard deviation, defined by the `step` variable we defined earlier. I'll say more about the choice of this step size later. For now, the 0.1 value we've chosen will work fine. The new proposal value for α is stored in `a.proposal`. Then we compute the "population size" on the current "island," computing the log-likelihood of the `Total.Tools` values, at the current parameter values. Then we compute the log-likelihood at the proposal value of α . To get the probability of accepting the proposal in `a.proposal`, we compute the ratio of the posterior probability at the proposal to the posterior probability at the current values. We're going to use flat priors for the moment (later, I'll show you how to include them), and so this reduces to just the ratio of the likelihoods. The ratio is computed on the log scale, where the math is more accurate, and then the difference is exponentiated to make it a probability again, stored in `pr.accept`. Finally, a random number between zero and one is generated (`rrunif(1)`), and if this number is less than the acceptance probability, `a.proposal` is assigned to be the new current value of α . Otherwise, the chain stays at the old value.

The next two blocks of code just perform the same steps in the same order, but for the other two parameters in the model, β_P (`bp`) and β_C (`bc`). Finally, the last line before the loop ends just updates a progress bar, to keep you informed of how much longer you'll likely need to wait for the chain to finish sampling. In this case, unless your computer is very old indeed, it'll take much less than a minute. On my aging 2.66 GHz desktop, it took only 25 seconds to complete all 100-thousand samples.

Now, let's bundle up the samples into a common data frame, for convenience:

R code
11.7

```
post <- data.frame( a=samples.a , bp=samples.bp , bc=samples.bc )
```

You can now think of the contents of this data frame as being inferentially the same as the samples you drew from the naive posterior in earlier chapters, using `sample.naive.posterior`. In those cases, the posterior was always assumed to be multivariate Gaussian. Now, we've made no such assumption. But we have had to do more work to get to this point.

It'll be useful to compare these samples to the Gaussian assumption (quadratic approximation). So quickly fit the model the old-fashioned way (as in Chapter 8):

R code
11.8

```
m <- mle2( Total.Tools ~ dpois( lambda=exp(a + bp*log.pop + bc*contact.hi) ) ,
```

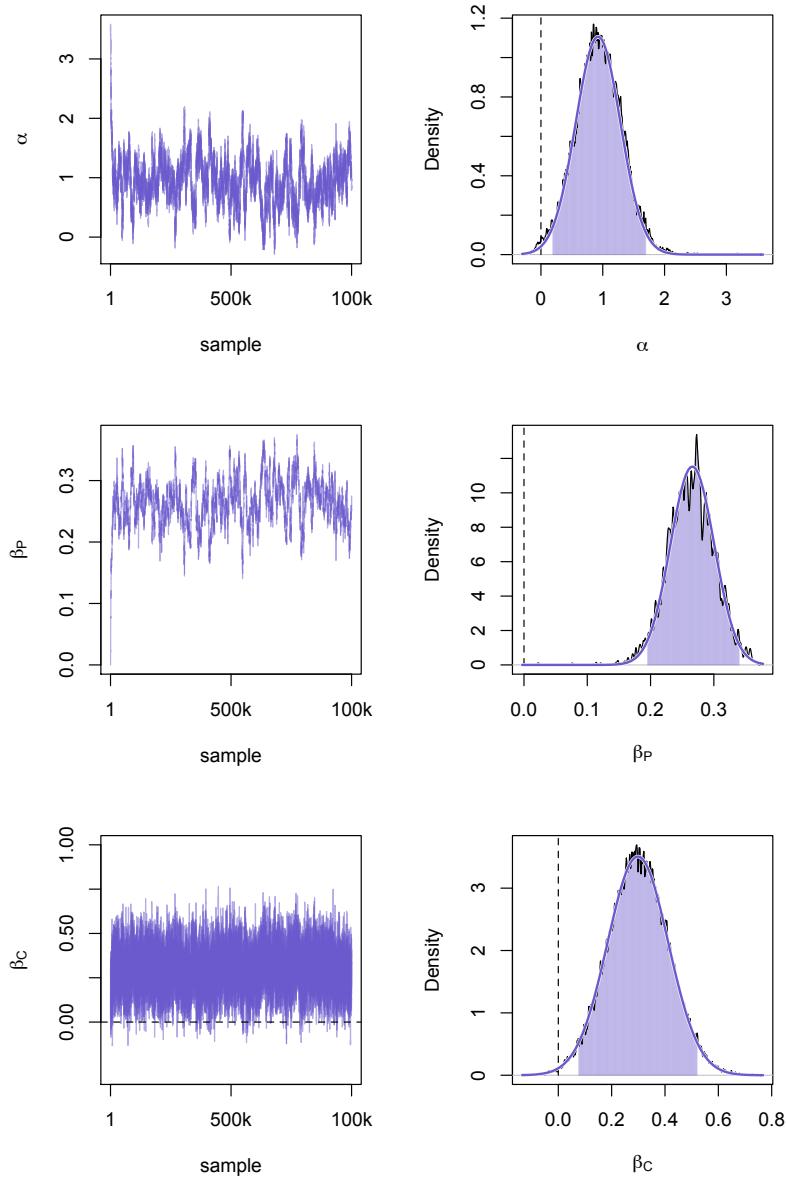


FIGURE 11.3. Samples from the posterior density of the Polynesian islands model, produced by Metropolis algorithm.

```
data=d , start=list(a=log( mean(d$Total.Tools) ),bp=0,bc=0) )
```

Now in FIGURE 11.3, I display the samples from the Markov chain, with the time trend of samples shown in the lefthand column and the densities shown in the righthand column. Each row shows samples for only one parameter, α , β_P , or β_C . In the time trends for both α and β_P , it is easy to see the initial run towards the high-density region. Both of these parameters began far from the center of their posterior distributions, and so both traveled quickly away from their starting points. Both then settled down into regions that never returned to their starting points. The parameter β_C started at zero and always quickly wandered away from it, but the posterior for this parameter came back to zero many times. The righthand column shows the same samples, but viewed as densities, with the quadratic approximation superimposed with a thin blue curve. In all three cases, the quadratic approximation the MLE gives us is extremely similar to the Markov chain samples.

Not only does the Markov chain give us these posterior shapes for each parameter, but it also gives us the right covariances among parameters as well. You can quickly compute the variance-covariance matrix for your samples with:

R code
11.9

```
cov( post )
```

	a	bp	bc
a	0.15046964	-0.014224836	-0.020849826
bp	-0.01422484	0.001403763	0.001439318
bc	-0.02084983	0.001439318	0.012972229

If you compare that output to what you get from `vcov(m)`, you'll see that the estimates are extremely similar. This is a case in which the model is very simple and has only one stochastic node. So maximum likelihood and the quadratic approximation do a very good job, only disagreeing with the Markov chain in the extreme tails of the posterior. It's a testament to the power of the theory that the posterior converges to multivariate near-normality, even with only 10 cases in the data. In this case, using the Markov chain has bought us very little, compared to MLE and quadratic approximation. Later in the chapter, I'll show you how to estimate a multilevel model with varying effects, and then you'll see that a Markov chain is much more capable, in general. For now, you can understand how the samples from the Markov chain relate to the MLE inferences you've been using throughout most of the book.

11.2.2. Metropolis-Hastings. The Metropolis algorithm works whenever the probability of proposing a jump to B from A is equal to the probability of proposing A from B, when the proposal distribution is symmetric. We ensured the proposals were always symmetric above, by using a Gaussian distribution to generate the proposals for each parameter. Since the Gaussian is symmetric, Metropolis worked.

But what about when the proposals cannot be symmetric? Suppose the model has a Gaussian outcome variable, and so we must estimate the posterior for σ ? As long as the current value of σ stays far away from zero, we might still get away with using `rnorm` to generate proposals. But if σ ever wanders near the zero boundary of its valid values, we might propose a negative standard deviation. No bueno. You might just clip off the negative proposals, moving them to 0.0001 or even just trying another proposal, until you get a positive one. But then the Metropolis algorithm no longer applies, and you will get a distorted estimate for the posterior of σ . The distortion will be quite small, as long as values near zero aren't too common. But wouldn't it be nice to have a way to estimate the posterior for parameters with bounds, like σ , without accepting such distortions?

We can do a good accurate job for σ , but only if we use a more general algorithm, the one that the Metropolis is itself just a special case of. The so-called *Metropolis-Hastings* algorithm¹¹³ provides a solution, one that will lead us not only to a solution to our σ vexation, but also to an extremely powerful way to accelerate our Markov chains, a technique known as *Gibbs sampling*. So for this section, I'll show you how to sample from a Gaussian model with a σ to estimate. Then in the next, I'll link our work here to Gibbs sampling.

Let's take a very easy example with a σ parameter, the adult !Kung height data from Chapter 4. The model is:

$$y_i \sim \text{Normal}(\mu, \sigma),$$

Where y is a vector of adult heights and we want to use the Markov chain to sample from the joint posterior of μ and σ . Load the data and pull out the adult heights into a vector y :

```
library(rethinking)
data(Howell1)
d <- Howell1
y <- d$height[ d$age > 18 ]
LL.Howell1 <- function( mu , sigma )
  sum( dnorm( y , mu , sigma , log=TRUE ) )
```

R code
11.10

The last line above defines a convenient log-likelihood function for the model.

Now our strategy for generating proposal for σ will be to draw them from a gamma distribution. The reason to use a gamma is because values drawn from a gamma distribution are always positive. Now, since the gamma distribution isn't symmetric, we can no longer use the plain Metropolis algorithm, but must now use some more general, the Metropolis-Hastings. What this means is changing the way we compute the probability of accepting the proposal. In the plain Metropolis algorithm, the ratio we computed was (still ignoring priors for the now):

$$\Pr(\text{accept}) = \frac{\Pr(D|\sigma')}{\Pr(D|\sigma)},$$

where σ is the current value of the parameter and σ' is the proposed value. But the more general formula is actually:

$$\Pr(\text{accept}) = \frac{\Pr(D|\sigma') \Pr_P(\sigma|\sigma')}{\Pr(D|\sigma) \Pr_P(\sigma'|\sigma)},$$

where $\Pr_P(a|b)$ indicates the probability of proposing parameter value a when the current value is b . Now, this new formula was always actually in effect. But when the proposal distribution is symmetric, then $\Pr_P(a|b) = \Pr(b|a)$ and so the formula simplifies to the plainer Metropolis ratio of likelihoods (or posterior probabilities, if you include priors).

But we're going to use an asymmetric proposal distribution, and so we'll need to incorporate the gamma density into our ratio now. Here's the rest of the code to complete the example. I'll explain the new pieces afterwards.

R code
11.11

```
mu <- mean(y)
sigma <- sd(y)
num.samples <- 100*1000
samples.mu <- rep(0,num.samples)
samples.sigma <- rep(0,num.samples)
st <- Sys.time()
for ( i in 1:num.samples ) {

  # record samples
  samples.mu[i] <- mu
  samples.sigma[i] <- sigma

  # sample mu
  mu.proposal <- rnorm( 1 , mu , step )
```

```

LL.current <- LL.Howell1( mu , sigma )
LL.proposal <- LL.Howell1( mu.proposal , sigma )
pr.accept <- exp( LL.proposal - LL.current )
mu <- ifelse( runif(1) < pr.accept , mu.proposal , mu )

# sample sigma
sigma.proposal <- rgamma2( 1 , sigma , 0.01 )
LL.current <- LL.Howell1( mu , sigma )
  + dgamma2(sigma.proposal,sigma,0.01,log=TRUE)
LL.proposal <- LL.Howell1( mu , sigma.proposal )
  + dgamma2(sigma,sigma.proposal,0.01,log=TRUE)
pr.accept <- exp( LL.proposal - LL.current )
sigma <- ifelse( runif(1)<pr.accept , sigma.proposal , sigma )

progbar( i , min=1 , max=num.samples , starttime=st )
}
post <- data.frame( mu=samples.mu , sigma=samples.sigma )

```

The only new technique here is the change in the block that samples σ . The parameter μ can still get its proposals from the Gaussian, and so that step is still essentially plain Metropolis. Indeed, you can always mix plain Metropolis, Metropolis-Hastings, and even Gibbs sampling steps together in the same Markov chain, using each technique where you can get the most from it. For σ in this case, we need a Metropolis-Hastings step, and so the first line of that block pulls a random gamma-distributed value with mean σ and scale parameter 0.01. (The function `rgamma2` is part of the `rethinking` package, version 1.03 or higher.) Then when each log-likelihood is computed, we add (remember, adding logs is the same as multiplying) the gamma log-likelihoods of the proposal or the current value. Then the rest of the code proceeds the same as before.

Now if σ ever wanders close to zero, we won't have to mess around with clipping it or suffering a distortion. In the data example above, you can run the chain yourself and see that σ doesn't really get close enough to worry about the issue. But I've set up the code so you can simulate any simple Gaussian values you like, with any arbitrary mean and standard deviation, to see how the chain does with a very small σ . Here's one line of code to simulate Gaussian outcomes to estimate the posterior for, using the above chain:

```
y <- rnorm( 20 , mean=5 , sd=0.05 )
```

R code
11.12

Estimate the posterior for those values, and then play around with both the mean and standard deviation, to get a feel for how the posterior of σ changes shape as it gets close to the boundary.

11.2.3. Gibbs Sampling. Okay, so far you've seen how to write simple Markov chains to draw samples from the posterior distribution of the parameters. The last major conceptual topic, before turning to how to write chains that incorporate varying effects, is Gibbs sampling. Gibbs sampling¹¹⁴ is a special case of the Metropolis-Hastings algorithm that is very efficient. By "efficient," I mean that you can get a good estimate of the posterior from Gibbs sampling with many fewer samples than a comparable Metropolis approach. The improvement arises from using a form of *adaptive proposals*, in which the distribution of proposed parameter values really adjusts itself intelligently, depending upon where the chain is at the moment.

First, let's describe the problem a little better. Then I'll explain how Gibbs sampling works in detail. A major problem with Metropolis-Hastings is that it can take it a long time to adequately explore the posterior density. The primary reason it can take so long is that the proposal distribution we use knows nothing about the target distribution. Let's focus on the standard deviation of a Gaussian proposal distribution, for example. This is the value that I've called `step`, earlier in this chapter. You can think of it as the step size of proposals. If this value is too large, then many proposals will be rejected, because they will jump too far outside the target region we wish to explore. If they are instead too small, then most will be accepted, but it will take a very long time for the chain to explore the tails of the target density. If you set the step size just right, then you can tradeoff one of the concerns for the other.

But the algorithm will still be rather inefficient, because what we really want is to take big steps when we are far from the center of the target distribution and small steps when we are close to the center. In other words, we'd like the proposals to be a function of how close we are to the target. Metropolis-Hastings, in its usual form, has no way to do this. If we could make the proposal distribution a function of the current parameter values, then we could improve the efficiency of the Markov chain.

To work towards one solution to this problem, let's consider the final samples from a Metropolis chain, one that has given us a good picture of the target posterior density. Suppose for example we are estimating the mean μ and standard deviation σ for 20 adult heights from the !Kung data you met in Chapter 4. I display such a posterior in FIGURE 11.4. The lefthand plot shows samples from the posterior for both parameters,

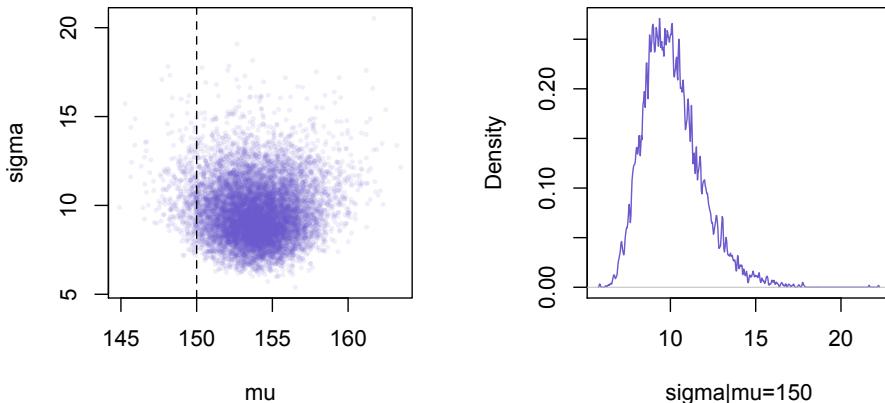


FIGURE 11.4. Samples from the posterior for 20 adult !Kung heights. The vertical dashed line on the left shows the location of the slice through the posterior of σ shown on the right. These slices are posterior densities conditional on the value of the other parameter, μ in this case. Writing analytical expressions for these slices makes Gibbs sampling possible.

or $\Pr(\mu, \sigma | D)$. The vertical dashed line locates a particular slice through the posterior, at $\mu = 150$. The righthand plot shows the profile of the posterior at this slice. What you are seeing is the posterior density of σ , where $\mu = 150$. Another way to say this is you are viewing $\Pr(\sigma | D, \mu)$.

Now, the interesting thing about this kind of slice through the full posterior is that, even when we cannot compute an analytical expression for the full density $\Pr(\mu, \sigma | D)$, we often can compute an analytical expression for $\Pr(\sigma | \mu, D)$ (or for $\Pr(\mu | \sigma, D)$). By treating one of the parameters as a constant, like the data D , it makes the math easier. In the case of a Gaussian outcome, it is possible to write down analytical expressions for $\Pr(\sigma | \mu, D)$ and for $\Pr(\mu | \sigma, D)$.

Why might we want to do this? In order to generate intelligent proposals, so that the Markov chain will be more efficient. Think of it this way. If the Markov chain is at $\mu = 150$, then the target for σ is the righthand plot in FIGURE 11.4. So if we have a formula to give us the shape of the slice in FIGURE 11.4, then we can use that function to produce random proposal steps that will automatically be smaller, when

the chain is close to the center of the target, and automatically be larger, when the chain is far from the center of the target. So if we sample out proposal values for both σ and μ from $\Pr(\sigma|\mu, D)$ and $\Pr(\mu|\sigma, D)$, respectively, then the Markov chain will converge much faster and we'll need fewer samples overall to get a good picture of the full target, $\Pr(\mu, \sigma|D)$, the joint posterior.

The catch is that, in order to get an analytical expression for these slices, we'll have to make an explicit choice of the prior distribution for each parameter. Flat priors will no longer do. Indeed, we must choose a prior distribution that is *conjugate* with the likelihood function. So called *conjugate pairs* are matching distributions for priors and likelihoods that produce a posterior with the same family as the prior. For example, the likelihood function in this case is Gaussian. For σ , the standard deviation of the Gaussian, it turns out that we want to use a gamma distribution. Or rather, the natural parameter of the Gaussian is $\tau = 1/\sigma^2$, and if you use a gamma shaped prior for τ , then the posterior will also be a gamma distribution. The analogous conjugate prior for the mean μ is a normal distribution.

All of this means we can rewrite our Markov chain to generate proposals from the analytical expressions for the slices through the joint posterior. Now the probability of accepting a proposed new value σ' drawn at random from the slice is:

$$\Pr(\text{accept}) = \frac{\Pr(\sigma'| \mu, D)}{\Pr(\sigma | \mu, D)} \frac{\Pr(\sigma | \mu, D)}{\Pr(\sigma' | \mu, D)} = 1.$$

Notice that every term on the top is matched by the same term on the bottom, and so the entire probability is now exactly 1, always, regardless of the values of σ and σ' . This means that Gibbs sampling will always accept any proposal. No rejected proposals means more efficient search of the target distribution.

Okay, so how do you actually implement this strategy, in code form? Because proposals will always be accepted, all you need to do is use a single line for each parameter, drawing a random value from its current “slice” posterior, conditional on the other parameters and the data. Here’s what it looks like, for a simple Gaussian model.

R code
11.13

```
library(rethinking)
data(Howell1)
d <- Howell1
y <- d$height[ d$age > 18 ]
```

```

# priors; these are "weak"
mu0 <- 155
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

# offset initial guess, so we can see rate of approach to target
mu <- mean(y) - 10
sigma <- sd(y) * 2

num.samples <- 100*1000
samples.mu <- rep(0,num.samples)
samples.sigma <- rep(0,num.samples)
st <- Sys.time()
for ( i in 1:num.samples ) {

  # record samples
  samples.mu[i] <- mu
  samples.sigma[i] <- sigma

  # sample sigma, conditioned on mu
  sigma <- sqrt( 1/rgamma( 1 , shape=shape0 + length(y)/2 ,
    rate=rate0 + sum( (y-mu)^2 )/2 ) )

  # sample mu, conditioned on sigma
  mu <- rnorm( 1 ,
    mean=( mu0/sigma0^2 + sum(y)/sigma^2 ) /
      (1/sigma0^2 + length(y)/sigma^2) ,
    sd=sqrt(1/(1/sigma0^2 + length(y)/sigma^2)) )

  progbar( i , min=1 , max=num.samples , starttime=st )
}
post <- data.frame( mu=samples.mu , sigma=samples.sigma )

```

The parameter σ is sampled from its posterior slice and μ from its posterior slice. These posterior density formulas can appear a little monstrous, especially in code form. But all they actually do is analytically describe the shape of the slices through the target joint posterior density.

The most confusing thing about the above code is that we sample σ from the square root of an inverse-gamma distribution. This is because the natural parameter for the spread of a Gaussian distribution is $\tau = 1/\sigma^2$, and so we actually sample τ and then convert it to σ . I did both the sampling and conversion in the same line, in the code above, which

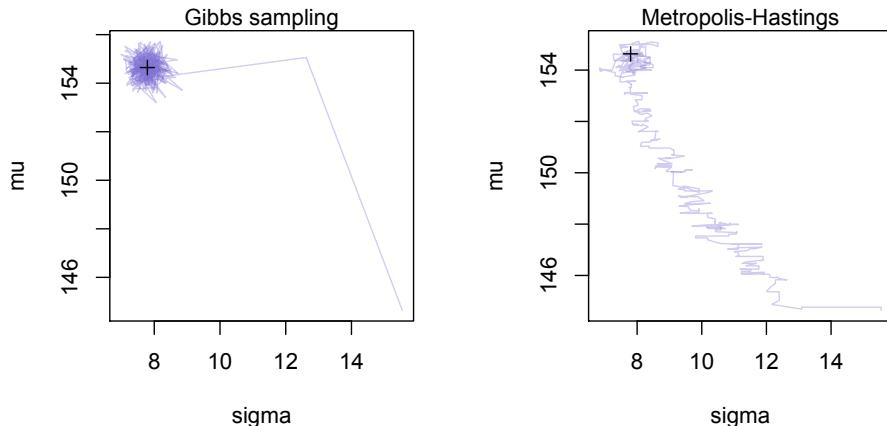


FIGURE 11.5. The first 500 samples from Gibbs sampling (left) and fixed-step Metropolis-Hastings (right). The black plus signs in each plot locate the target high density region. Gibbs sampling gets nearby in exactly two steps.

is why the value produced by `rgamma` divides 1 and then is wrapped in `sqrt`. Those two steps make τ into σ .

Alright, so let's compare the efficiency of the Gibbs algorithm above to the analogous Metropolis-Hastings algorithm that uses fixed proposal distributions. You have the Metropolis-Hastings code, in the previous section. The first thing to compare is how long each approach takes to finish 100-thousand samples. On my 2.66 GHz desktop, the Gibbs chain took 5 seconds, while the plain Metropolis-Hastings took about 17 seconds. That's about 19956 samples per second for the Gibbs chain, and 5974 for the Metropolis-Hastings chain. Now, these models are so simple that the improvement hardly matters. But at this relative rate of improvement, a Metropolis chain that takes an hour to finish can be made to finish in about 18 minutes, once converted to Gibbs sampling.

Not only will the Gibbs chain execute faster, but it'll also explore the parameter space better. We can compare how quickly the two approaches get into the high probability region of the target distribution. In FIGURE 11.5, I display the first 500 samples from both chains, with Gibbs sampling on the left and fixed-step Metropolis-Hastings on the

right. Both chains started at the same parameter values, but Gibbs sampling approached the target region in exactly two giant steps. Metropolis-Hastings, in contrast, took hundreds of steps to arrive in the same region. In the end, both algorithms produced the same approximate estimate of the joint posterior. But Gibbs did so much more efficiently.

The major limitation to Gibbs sampling is that it needs more information up front. Often, you can't choose a conjugate prior, or you just don't know one. In that case, you need to fall back on Metropolis-Hastings. It's fine to write Markov chains in which some parameters are sampled with Metropolis-Hastings and others with Gibbs sampling. The more Gibbs steps you can include, the more efficient the entire chain is likely to be.

11.2.4. Simulated annealing. There are yet other ways to tune Markov chain Monte Carlo. Another common technique is to use something called *simulated annealing* to improve how the chain explores the parameter space. I won't go into detail on implementing simulated annealing here, but the notion is that we often want to take big steps early on, when we don't know how far we are from the target region, but then take smaller steps later, once we're near it. Simulated annealing makes the step size of your Metropolis-Hastings proposals a function of how many samples you've taken. You want to decrease the width of the proposal distribution, as time goes on. But if you decrease the width too quickly, you can actually make things worse. So care is needed.

11.2.5. Hamiltonian MCMC. Describe Hamiltonian strategy? Could be confusing, but likely to be a popular method soon, so worth explaining.

11.3. Multilevel Markov chains

As I keep saying, the true value of Markov chain Monte Carlo estimation arises when you have a multilevel model. In that case, it is commonplace to find yourself without a way to compute the likelihood of the data, conditional on all of the parameters. The major obstacle arises from having to average over uncertainty at lower levels of the model (lower stochastic nodes), when consider uncertainty at the top level. In special cases, and with strategic assumptions, progress can be made. But in general, there's no guarantee that it's possible to produce a single computable likelihood function for a multilevel model. Markov chain Monte Carlo obviates the need for a single likelihood function,

because all you need at the moment you update a parameter is the likelihood that is relevant to it. All of the other parameters are treated as fixed and given.

In this section, I show you some simple examples of how to sample varying effects in a Markov chain. I stick to simple Metropolis-Hastings chains, for clarity. But the basic idea remains the same, whether you use Gibbs sampling or even some other MCMC algorithm (there are many others). I'll explain the motivation behind the sampling strategy, discussing varying intercepts and varying slopes separately. Then I'll provide an extended example, presenting the working code.

11.3.1. Varying intercepts. Maybe the weirdest fact about multilevel models and Markov chains is that a “parameter” at one level of the model is “data” at another level. It all depends upon where in the chain you are. It might help you grasp what is going on if you actually forget about the data-parameter distinction in this context, at least as far as designing the chain goes. Everything in the Markov chain is potentially generated by a stochastic process. We sample parameters just as we might sample data.

To see how this helps us, consider perhaps the simplest multilevel Gaussian model, one with varying intercepts:

$$\begin{aligned} y_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma), \\ \mu_{ij} &= \alpha + \alpha_j, \\ \alpha_j &\sim \text{Normal}(0, \sigma_\alpha). \end{aligned}$$

How can we produce a Markov chain sampling strategy for such a model? The key insight is to realize that each stochastic node implies its own likelihood function, one that applies to the “data” on the left side of the same line. At the top level, the data are y_{ij} values, and the likelihood function is Gaussian, with parameters μ_{ij} and σ . The second stochastic node is also Gaussian, but its data are the varying intercepts, α_j , and its parameter is just σ_α .

What this distinction buys us is that when we consider a proposed jump for, say, the parameter α , we don't need to concern ourselves with the second stochastic node. All that matters for accepting or rejecting a proposed value of α is the likelihood function implied by the top line of the model, because that's the only place α appears. Likewise, if we are considering a proposal for σ_α , all we need is the bottom likelihood function, the one that “predicts” the varying intercepts.

And when a parameter appears in more than one stochastic node, whether as “data” or as a parameter, then we need all of the stochastic

nodes it appears. And so when we consider a proposal for any α_j , we need both stochastic nodes. However, we don't have to average over uncertainty in any of the other parameters, because the Markov chain let's us treat them as fixed.

What all of this hand waving means is that different parameters get updated with different log-likelihood functions. In the case of the simple model at the start of this section, we could match up each kind of parameter with the necessary log-likelihood as follows.

- (1) α : $\sum_{ij} \log \Pr(y_{ij}|\alpha, \alpha_j, \sigma)$
- (2) σ : $\sum_{ij} \log \Pr(y_{ij}|\alpha, \alpha_j, \sigma)$
- (3) σ_α : $\sum_j \log \Pr(\alpha_j|\sigma_\alpha)$
- (4) α_j : $\sum_{ij} \log \Pr(y_{ij}|\alpha, \alpha_j, \sigma) + \sum_j \log \Pr(\alpha_j|\sigma_\alpha)$

So for example to decide whether or not to accept a proposed varying intercept α'_j , we'd need to compute the ratio:

$$\frac{\exp(\sum_{ij} \log \Pr(y_{ij}|\alpha, \alpha'_j, \sigma) + \sum_j \log \Pr(\alpha'_j|\sigma_\alpha))}{\exp(\sum_{ij} \log \Pr(y_{ij}|\alpha, \alpha_j, \sigma) + \sum_j \log \Pr(\alpha_j|\sigma_\alpha))}.$$

You'll see this in code form, a bit later.

11.3.2. Varying slopes. Writing a chain to sample varying slopes works the same way as with varying intercepts. Pay attention to each stochastic node in which the particular slope parameter you are updating appears. The relevant log-likelihood for updating the parameter will be the sum of the log-likelihoods from the separate stochastic nodes. This can appear tricky, when you have correlated varying intercepts and slopes, but the basic strategy remains the same. You just need to remember that the likelihood of either a varying intercept or slope will depend upon its correlated partner. If you're not comfortable with multivariate normal distributions, then best leave this bother to software like BUGS and JAGS that is designed to handle such problems efficiently.

11.3.3. Example: Multilevel logistic regression. It's often hard to appreciate what the sampling strategy really is, until you see the algorithm in full code glory. So let's return to the chimpanzee prosociality data from Chapter 8 and Chapter 10. In Chapter 8, we estimated a simple logistic regression. In Chapter 10, we used `lme4` to estimate varying intercepts logistic regression. Now we'll do the same model as in Chapter 10, but with a home-grown Markov chain. This will serve two purposes. First, you'll really see how a chain samples varying intercepts, in the simplest implementation I can think of. Second, you'll be able to compare your MCMC estimate of the posterior to the information

`lme4` gave us. As you'll see, `lme4` is amazing, but there are some things it just can't be expected to do. And so at the end of this section, we'll plot model predictions over the data, now having a complete posterior density, across all parameters, something we could not really do with a fit from `lme4`.

Here's the model to estimate:

$$\begin{aligned} L_{ij} &\sim \text{Binomial}(p_{ij}, 1), \\ \log \frac{p_{ij}}{1 - p_{ij}} &= \alpha + \alpha_j + \beta_P P_{ij} + \beta_{PC} P_{ij} C_{ij}, \\ \alpha_j &\sim \text{Normal}(0, \sigma_\alpha). \end{aligned}$$

So we have four non-varying parameters— α , β_P , β_{PC} , and σ_α —and seven varying intercepts α_j , one for each chimpanzee in the sample. First thing, let's get the data back into R:

R code
11.14

```
library(rethinking)
data(chimpanzees)
d <- chimpanzees
```

Now we're going to need two log-likelihood functions, one for each stochastic node. I'll call the top-level stochastic node's function `LL.chimp.Li` and the second node's function `LL.chimp.aj`, since it applies to the varying intercepts.

R code
11.15

```
LL.chimp.Lij <- function( a , aj , bp , bpc ) {
  ai <- a + aj[d$actor]
  sum( dbinom( d$pulled.left ,
    prob=logistic( ai + bp*d$prosoc.left + bpc*d$condition ) ,
    size=1 , log=TRUE ) )
}
LL.chimp.aj <- function( aj , a , sigma.a ) {
  sum( dnorm( aj , mean=a , sd=sigma.a , log=TRUE ) )
}
```

Now we're ready to set starting values for the parameters and initialize the empty vectors to hold the samples.

R code
11.16

```
# initialize parameters
nj <- length( unique( d$actor ) )
a <- 0
sigma.a <- 1
```

```

aj <- rep(0,nj)
bp <- 0
bpc <- 0
# initialize chain
nsamp <- 1000*200
samples.a <- rep(0,nsamp)
samples.sigma.a <- rep(0,nsamp)
samples.aj <- matrix(0,nrow=nj,ncol=nsamp) # rows actors, cols samples
samples.bp <- rep(0,nsamp)
samples.bpc <- rep(0,nsamp)
step <- 1/10

```

One trick to note above. The varying intercepts α_j (`samples.aj`) will be stored in a matrix that has 7 rows and 200-thousand columns. Each row is a unique j value, a unique chimpanzee. Each column is a sample. So on any step i in the chain, the varying intercept for actor j is retrieved with `samples.aj[j,i]`. There's nothing special about using a matrix here, but it will make it easier to loop over the varying intercepts, rather than writing a unique code block for each. Trust me, if you had dozens of clusters, or even hundreds, you'd want to loop over them in this way.

Time to launch. When you paste the code below into R, it'll begin sampling 200-thousand times for each parameter.

```

st <- Sys.time()
for ( k in 1:nsamp ) {
  # record samples
  samples.a[k] <- a
  samples.sigma.a[k] <- sigma.a
  samples.aj[,k] <- aj
  samples.bp[k] <- bp
  samples.bpc[k] <- bpc

  # sample a
  prop <- rnorm( 1 , a , step )
  ll.here <- LL.chimp.Lij( a , aj , bp , bpc )
  ll.prop <- LL.chimp.Lij( prop , aj , bp , bpc )
  pr <- exp( ll.prop - ll.here )
  a <- ifelse( runif(1) < pr , prop , a )

  # sample sigma.a
  prop <- rnorm( 1 , sigma.a , step )
  ll.here <- LL.chimp.aj( aj , 0 , sigma.a )

```

R code
11.17

```

ll.prop <- LL.chimp.aj( aj , 0 , prop )
pr <- exp( ll.prop - ll.here )
sigma.a <- ifelse( runif(1) < pr , prop , sigma.a )

# sample bp
prop <- rnorm( 1 , bp , step )
ll.here <- LL.chimp.Lij( a , aj , bp , bpc )
ll.prop <- LL.chimp.Lij( a , aj , prop , bpc )
pr <- exp( ll.prop - ll.here )
bp <- ifelse( runif(1) < pr , prop , bp )

# sample bpc
prop <- rnorm( 1 , bpc , step )
ll.here <- LL.chimp.Lij( a , aj , bp , bpc )
ll.prop <- LL.chimp.Lij( a , aj , bp , prop )
pr <- exp( ll.prop - ll.here )
bpc <- ifelse( runif(1) < pr , prop , bpc )

# sample actor intercepts
for ( j in 1:nj ) {
    prop <- aj
    prop[j] <- rnorm( 1 , aj[j] , step )
    ll.here <- LL.chimp.Lij( a , aj , bp , bpc )
        + LL.chimp.aj( aj , 0 , sigma.a )
    ll.prop <- LL.chimp.Lij( a , prop , bp , bpc )
        + LL.chimp.aj( prop , 0 , sigma.a )
    pr <- exp( ll.prop - ll.here )
    aj[j] <- ifelse( runif(1) < pr , prop[j] , aj[j] )
}

progbar( k , min=1 , max=nsamp , starttime=st )
}
post <- data.frame( a=samples.a , bp=samples.bp , bpc=samples.bpc ,
    sigma.a=samples.sigma.a , t(samples.aj) )
colnames(post)[5:11] <- paste( "a" , 1:7 , sep="" )

```

It took my aging 2.66 GHz desktop computer about 18 minutes to complete all 200-thousand samples. We'll want to compare the samples from this chain to the estimates provided by `lme4`, for the same model:

R code
11.18

```
m <- glmm( pulled.left ~ (1|actor) + prosoc.left * condition
    - condition , data=d , family=binomial )
```

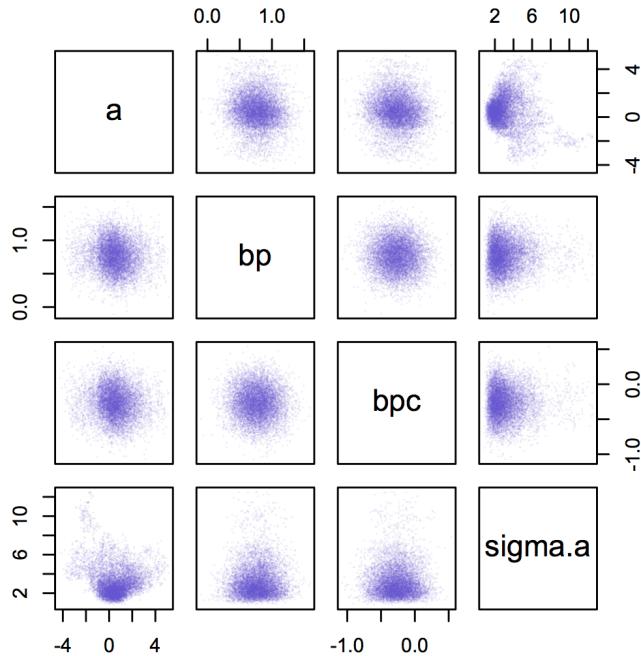


FIGURE 11.6. 10-thousand samples from the posterior of the non-varying parameters.

Let's compare the crude `precis` summaries of the estimates. First, the `lme4` estimates:

```
precis( m )
```

R code
11.19

	Estimate	Std. Error	2.5%	97.5%
(Intercept)	0.3919602	0.7240576	-1.0271666	1.8110871
prosoc.left	0.8184636	0.2593128	0.3102199	1.3267072
prosoc.left:condition	-0.1327258	0.2955735	-0.7120391	0.4465876
((Intercept) actor)	1.8488511	NA	NA	NA

And now the Markov chain estimates (note that you need `rethinking` version 1.03 or higher, for `precis` to work with the samples in this way):

```
precis( post )
```

R code
11.20

	Expectation	Std. Error	lower 0.95	upper 0.95
a	0.5315849	1.2815649	-2.2558735	3.2906205
bp	0.7677212	0.2185655	0.3440187	1.2015206

	bpc	-0.2687117	0.2131819	-0.6895719	0.1456936
sigma.a	3.1147005	1.5171106	1.0599618	6.0977366	
a1	-1.1284176	1.2867367	-3.9670082	1.5625217	
a2	5.1107027	2.5527949	0.9228596	10.9848733	
a3	-1.4319598	1.2956506	-4.3312945	1.2260725	
a4	-1.4331753	1.2929795	-4.2526440	1.2374504	
a5	-1.1231761	1.2923878	-3.9848697	1.5496656	
a6	-0.1770750	1.2857664	-2.9443229	2.5582048	
a7	1.3849161	1.3088232	-1.5188450	4.0712650	

I display 10-thousand samples from the posterior of the non-varying parameters in FIGURE 11.6. It is not unusual to get somewhat different estimates from a Markov chain and `lme4`, when the model is not Gaussian at every level. Still, the qualitative results are the same. But the Markov chain estimates more uncertainty about every parameter aside from the two slopes.

Now to compare the varying intercept estimates. I'm going to manufacture a quick table to hold both the `lme4` estimates of the varying intercepts and the mean and median of the samples from the Markov chain. I'll also tack on the end the estimated standard errors from `lme4` and the standard deviations from the chain.

The tricky aspect will be that `lme4` doesn't actually provide standard errors for the varying effects α_j , but rather for the sum $\alpha + \alpha_j$. But we can compute these from the MCMC samples easily:

R code
11.21

```
A1 <- post$a + post$a1
A2 <- post$a + post$a2
A3 <- post$a + post$a3
A4 <- post$a + post$a4
A5 <- post$a + post$a5
A6 <- post$a + post$a6
A7 <- post$a + post$a7
A <- data.frame( A1 , A2 , A3 , A4 , A5 , A6 , A7 )
library(arm)
x1 <- ranef(m)$actor[,1] + fixef(m)[1]
x1se <- se.ranef(m)$actor[,1]
x2 <- apply( A , 2 , mean )
x3 <- apply( A , 2 , median )
x4 <- apply( A , 2 , sd )
z <- cbind( lme4=x1 , mean=x2 , median=x3 , lme4se=x1se , sd=x4 )
rownames(z) <- colnames(A)
z
```

	lme4	mean	median	lme4se	sd
--	------	------	--------	--------	----

```
A1 -0.7054531 -0.5968327 -0.5930480 0.2408083 0.2891478
A2 3.9219493 5.6422875 5.0735601 0.8741857 2.2009549
A3 -1.0035361 -0.9003749 -0.8972118 0.2483834 0.2960965
A4 -1.0035361 -0.9015904 -0.8975043 0.2483834 0.2981650
A5 -0.7054531 -0.5915912 -0.5894904 0.2408083 0.2891683
A6 0.2179212 0.3545099 0.3545709 0.2472422 0.2906420
A7 1.7048892 1.9165010 1.9004395 0.3627645 0.4009529
```

The first column are the varying intercept estimates provided by `lme4`, and the second and third are the mean and median of the corresponding samples from the Markov chain. Despite it's heroic approximations, `lme4` has done an amazing job of estimating the varying intercepts.

But if you now look at the last two columns above, you'll see that `lme4` has underestimated the width of these densities. The penultimate column contains the estimated standard errors (standard deviation of the posterior) provided by `lme4`. The last column shows the corresponding standard deviations of the samples from the Markov chain. While the relative magnitudes of the standard errors are very similar in both columns, the absolute magnitudes differ, by quite a lot in the case of actor number 2. The estimates from `lme4` are overconfident. This is probably the reason `lme4` does not provide these standard errors by default. In many cases, we shouldn't put much faith in them. Now, I have fit models with both `lme4` and my own Markov chains before and seen incredible agreement between them, as well. So it isn't that `lme4` always produces the wrong standard errors. It's just that if you really care about the uncertainty around your varying effects, you probably want to use a Markov chain. Given that the model itself is certainly not a perfect description of reality, we shouldn't get too excited about disagreement of this sort. But my preference is to use the conservative estimation tactic, which in this case is MCMC, rather than the confident one.

Plotting predictions. Now that we have samples for every parameter, fixed and varying, we can use our old methods to plot predictions that incorporate all of the uncertainty embodied in the posterior.

```
prosoc <- rep( c(0,1,0,1) , 7 )
condit <- rep( c(0,0,1,1) , 7 )
actor <- rep( 1:7 , each=4 )
i <- sample( 1:nrow(post) , size=10000 )
post2 <- post[i,]
pred.propleft <- sapply( 1:length(prosoc) , function(i)
  mean( logistic( post2$a + post2[, (4+actor[i])] +
    post2$bp*prosoc[i] + post2$bpc*prosoc[i]*condit[i] ) )
```

R code
11.22

```

pred.propleft.ci <- sapply( 1:length(prosoc) , function(i)
    PCI( logistic( post2$a + post2[, (4+actor[i])]
        + post2$bp*prosoc[i] + post2$bpc*prosoc[i]*condit[i] ) )
p <- by( d$pulled.left ,
    list(d$prosoc.left,d$condition,d$actor) , mean )
p <- as.vector(p)
plot( p , xlab="" , ylab="proportion pulled left" , ylim=c(0,1) ,
    col="slateblue" , pch=rep(c(1,2,21,24),7) , xaxt="n" ,
    bg=col.alpha("slateblue",0.8) )
lines( c(-1,length(p)+1) , c(0.5,0.5) , lty=2 )
for ( i in 1:7 ) {
    r <- (i-1)*4 + 1
    x <- r:(r+3)
    y <- p[x]
    lines( x , y , col="slateblue" )
    lines( x , pred.propleft[x] )
    lines( x , pred.propleft.ci[1,x] , lty=2 )
    lines( x , pred.propleft.ci[2,x] , lty=2 )
}

```

The results are shown in FIGURE 11.7.

11.4. BUGS, JAGS and DAGs

Simple advice on specifying these models in BUGS/JAGS syntax. So many books teach these specifications, best to just explain the strategy and then cite Gelman and Hill or such.

Fitting the chimpanzee varying intercepts model with JAGS. Here's the model file:

```

model{
    for ( i in 1:N ) {
        logit(p[i]) <- mu.alpha + alpha[actor[i]] + betaP*x1[i] + betaPC*x2[i]
        y[i] ~ dbin( p[i] , 1 )
    }
    for ( j in 1:N.actors ) {
        alpha[j] ~ dnorm( 0 , tau.alpha )
    }
    mu.alpha ~ dnorm( 0 , 1e-4 )
    betaP ~ dnorm( 0 , 1e-4 )
    betaPC ~ dnorm( 0 , 1e-4 )
    tau.alpha ~ dgamma( 1e-3 , 1e-3 )
}

```

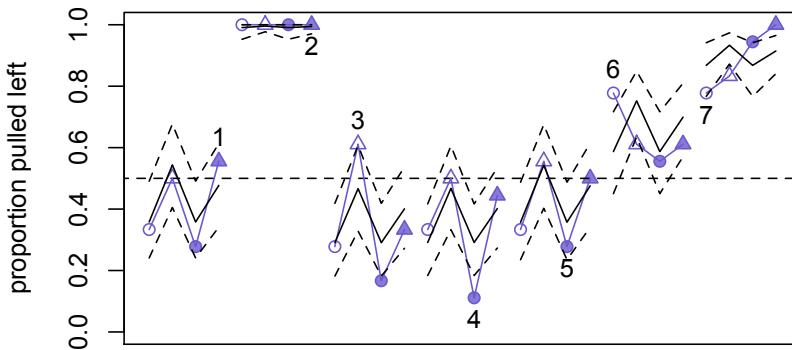


FIGURE 11.7. Predictions for the chimpanzee data and model, using the joint posterior samples from the Markov chain.

Put that in a file called `chimpanzees.jags`. Then if you have JAGS and the R library `rjags` installed, you can run in R:

```
library(rjags)
library(rethinking)
data(chimpanzees)
d <- chimpanzees

# tell JAGS about the model
m.jags <- jags.model("chimpanzees.jags",
  list( y=d$pulled.left , x1=d$prosoc.left , x2=d$condition ,
    actor=d$actor , N=nrow(d) , N.actors=7 ) ,
  n.chains=1 )

# first 4-thousand samples to burn in
update( m.jags , n.iter=4000 )

# 10-thousand samples for first 4 parameters
post.jags <- coda.samples( m.jags ,
  c("mu.alpha","betaP","betaPC","tau.alpha","alpha") ,
```

R code
11.23

```

n.iter=10000 )

# print summary
summary(post.jags)

# plot chains and density estimates
plot(post.jags)

```

You should get a summary table of the chain that looks something like:

	Mean	SD	Naive SE	Time-series SE
alpha[1]	-1.2624	1.0448	0.007388	0.125544
alpha[2]	4.3886	2.1045	0.014881	0.070315
alpha[3]	-1.5688	1.0463	0.007398	0.127379
alpha[4]	-1.5669	1.0456	0.007394	0.126502
alpha[5]	-1.2597	1.0387	0.007345	0.127269
alpha[6]	-0.3163	1.0422	0.007370	0.126337
alpha[7]	1.2345	1.0640	0.007523	0.124756
betaP	0.7572	0.2145	0.001517	0.003225
betaPC	-0.2720	0.2153	0.001522	0.003491
mu.alpha	0.6795	1.0315	0.007294	0.128149
tau.alpha	0.2463	0.1916	0.001355	0.006772

These are very similar to what we got with our own clunky Metropolis-Hastings chain. JAGS has done it much more quickly, and with less fuss. But you have to learn to specify its models, and sometimes that is tricky.

12 Missing Values and Other Errors

(1) Simultaneous estimation of missing values. (2) Incorporating measurement error by assigning probability distribution to each predictor value.

Endnotes

1. McElreath and Boyd 2007. [8]
2. Pierre-Simon Laplace (1749–1827) advocated the use of inverse probability, actually applying it to important scientific calculations. Thomas Bayes, in contrast, didn't even publish his ideas on inverse probability in his own lifetime. [8]
3. See Gigerenzer et al. 2004 for examples. [8]
4. This isn't really a new phenomenon. See for examples Jaynes 1976. That paper also contains a very entertaining and educational exchange between Jaynes and a defender of "orthodox" non-Bayesian statistics. [9]
5. See for example studies reviewed in Gigerenzer et al. 2004. [15]
6. A wide-ranging place to start is Gigerenzer et al. 1990. This book has a detailed discussion of some controversies surrounding Fisher's significance tests, at the time he first proposed them. Jaynes 2003 also discusses the historical accidents that factored into the growth of statistics, including the distinct personalities of men like Ronald Fisher and Harold Jeffreys, among others. See Chapters 16 and 17 of that book. [16]
7. I recommend interested readers to go straight to a modern translation of Popper's *Logic of Scientific Discovery*. Chapters 6, 8, 9 and 10 in particular demonstrate that Popper appreciated these issues.
8. See Cho 2011 for a December 2011 summary focusing on debates about measurement. [20]
9. See Mulkay and Gilbert 1981 for many examples of "Popperism" from practicing scientists, including famous ones. [20]
10. Collins and Pinch 2011. Some scientists have read this book as an attack on science. However, as the authors clarify in the second edition, this was not their intention. Science makes myths, like all cultures do. That doesn't necessarily imply that science does not work. [20]
11. Popper himself had to deal with this kind of theory, because the rise of quantum mechanics in his lifetime presented rather serious challenges to the notion that measurement was unproblematic. See chapter 9 in his *Logic of Scientific Discovery*, for example. [21]
12. George E. P. Box is famous for this dictum. As far as I can tell, his first published use of it was as a section heading in a 1979 paper (Box, 1979). Population biologists

like myself are more familiar with a philosophically similar essay about modeling in general by Richard Levins, “The Strategy of Model Building in Population Biology” (Levins, 1966). [22]

13. This phrase is sometimes attributed to Carl Sagan. You can find it, for example, in Chapter 12 of his book *Demon Haunted World*. Surely the phrase, intended to mean various things, predates this usage. [22]

14. It's not clear when he said it, but many people who knew Haldane credit him with the statement that the discovery of fossil rabbits from the Precambrian would be sufficient to shake his faith in evolutionary theory. The Precambrian Period spans from the beginning of the Earth to about 540 million years ago.

15. Cohen 1994; Gigerenzer et al. 2004. [23]

16. Fisher was confronted with this problem in his lifetime. He never satisfactorily resolved it, and anyway, he was much more invested in his likelihood theory, which stands in direct opposition to significance testing. See Chapter 3 of Gigerenzer et al. 1990 for an historical discussion of the controversy. [24]

17. I first encountered this globe tossing strategy in Gelman & Nolan. Since I've been using it in classrooms, several people have told me that they have seen it in others places, but I've been unable to find a primeval citation, if there is one.

18. In Chapter ??, this term *uncertainty* will be put on a firmer foundation, by relating the shape of a distribution to information theory. For now, it's sufficient to acquire a heuristic appreciation for how narrow distributions reflect low uncertainty.

19. It's peculiar that this theorem is named after Thomas Bayes, as it predates him. In addition, his analysis of it was published only after his death. It was Laplace who outlined the major features of what we now call Bayesian inference. There are many places to read this history, but a catholic and literate presentation germane to the interests of readers of this book is Gigerenzer et al's *The Empire of Chance*.

20. Jaynes 2003 makes this point central to his introduction of Bayes' theorem.

21. Lindley for example wrote: “Scientific inference is essentially the passage from observed, past data to unobserved, future data. The roles of models and theories in doing this are explored. The Bayesian view is that all this should be accomplished entirely within the calculus of probability...” (1990) *Statist. Sci.* 5(1):44-65. It's not hard to find “Bayesian” statisticians who disagree. See Gelman 2011 for example, or this very book. Lindley's paper is very much worth reading, though. It covers a lot of important history, touches upon the small/large world problem, and briefly addresses George Box's (1980) important argument that Bayesian inference from the posterior is mainly useful within a model, but that other approaches are needed to check models. See also Lindley's concerns and Box's reply, at the end of Box's paper. Later chapters in this book end up taking a stance similar to Box's. Lindley's comments about anything-but-Bayes approaches to statistics are very similar to what one finds from Jaynes. See for example Jaynes 1976, his most polemic and entertaining essay.

22. See for example Andrew Gelman's observation that what he does is not well described by typical definitions of "Bayesian" statistical inference. See Gelman 2009 and Gelman 2011.
23. Indeed, philosophers tend to identify Bayesian inference with subjectivity. See for recent examples Mayo 2004 and Casella's reply in the same chapter. Casella in particular, who is a statistician instead of a philosopher, actually waves off objective approaches to priors, without citation. Objective approaches to selecting priors were developed in the early 20th century and have been in continuous use in experimental physics ever since. See papers in Rosenkrantz 1989 and the many examples and links to early papers in Jaynes 2003. It is worth noting that economists and some other social scientists do sometimes use subjective priors of agents in models of human decision making. These scholars rarely use subjective priors in analysis of data, however.
24. See also Box and Tiao 1973 page XX on using the prior to constrain parameter values. [39]
25. This concise example is from Box and Tiao 1973, page XX.
26. Technically, likelihoods are not necessarily proper probabilities, as they can be greater than 1, and often are when the distribution in question is continuous, rather than discrete. This point does matter sometimes in application, but it won't matter for the discussion here.
27. An interesting note about likelihood is that which variables in the formula are considered "the model" depends upon our purpose. For example, a common problem in field biology is to estimate population size, using mark-recapture designs. This can be usefully thought of as a kind of binomial sampling problem, in which both the probability of a success p_W and the sample size n are unknown. Instead, both must be estimated from the data. In this case, the likelihood is better indicated by something like $L(p_W, n|n_W)$, even though the formula does not change. Of course, mark-recapture models are usually more complex than a simple binomial. But the point is that one person's data is another's parameter.
28. Add citations to Gauss and Laplace using flat priors and ending up at maximum likelihood. [50]
29. Fisher is responsible for much of the modern appreciation of maximum likelihood estimates and their properties. A good place to start into this literature is Stigler's (2007) "The Epic Story of Maximum Likelihood."
30. In particular, only some priors are uninformative across transformation, but that approach is a bit beyond the scope of this book. The mathematically confident reader might start with Box and Tiao's 1973 discussion of so-called Jeffreys priors. The bible for this stuff is Jaynes 2003.
31. In this regard, I follow Box 1980. [58]
32. It has nothing to do with naive Bayes classifiers.
33. In particular, you should never use Microsoft Excel to compute anything of scientific importance. Excel has had well-documented errors in both precision and approach

since the 1990's. See Yalta 2008, for some important examples.

34. Simulated annealing is available from within `optim` and `mle2` via changing the `method` of optimization. See `?optim` for details. For reading on simulated annealing, being an old-fashioned geek, I like the presentation in *Numerical Recipes: The Art of Scientific Computing*. The *Numerical Recipes* books are well-written, entertaining, and extremely informative.

35. Ben Bolker and R Development Core Team 2011.

36. Bolker 2008. [70]

37. Some readers will already know that you could instead sample models directly from a Beta density, which has a special relationship to the binomial density. But having tried to teach this material using the Beta density, I have learned that it introduces too many new concepts at this point. For those aware of the Beta density already and interested in the code, you can get functionally identical samples from the naive posterior by using `sim.models <- rbeta(10000, 7, 5)`.

38. Box and Tiao 1973, page 83.

39. See Box and Tiao 1973, page 84 and then page 122 for a general discussion. [84]

40. When the posterior has more than one peak, which does happen, then there will be more than one lower and upper bound, defining a dis-continuous interval. Not all algorithms for computing HPDI's actually pay attention to this possibility. So some caution is necessary. Always view the entire posterior, when in doubt. [84]

41. Gelman et al. 2004 page 38 argue for the pragmatic superiority of simple symmetrical intervals over HPDI's. But they also note cases in which an HPDI is superior. There's no single best interval for all applications, is the important lesson. As I argue a bit later in the main text, if different intervals provide rather different boundaries, then you probably shouldn't be reporting intervals, but plotting the entire posterior. [85]

42. See Adler and XX 2009. For a recent example of mistakenly assuming that there can be only one "neutral" model for a process, see Lansing et al 2010 and commentary that follows it. [127]

43. Gelman and Sterns. Nieuwenhuis et al 2011.

44. Howell 2010 and Howell 2000. See also Lee and DeVore 1976. Much more raw data is available for download from <https://tspace.library.utoronto.ca/handle/1807/10395>. [144]

45. This is an important thing to remember about the sampling approach. But in practice the approximation error can be reduced to a very reasonable level just by boosting the number of samples. [152]

46. Gelman and Hill 2007, page 59. [180]

47. These data and this example is extracted from Grafen and Hails 199?.
48. Data from Table 2 of Hinde and Milligan 2011.
49. The issue of considering the influence of shared ancestry on patterns across species is a big one, well beyond the scope of this book. A serious problem is that phylogeny is usually uncertain, and so that uncertainty must be incorporated into the analysis. The most common approaches, like phylogenetically independent contrasts, ignore this problem. More recent approaches are substantially more powerful, but also substantially harder to use. See Felsenstein 2001 for an overview. See Nunn 2011 for a more recent treatment. ?
50. Or *parameter* \times *function(data)*.
51. See Palais 2001. See also <http://tauday.com/>. For example, a much-celebrated theorem in mathematics is Euler's identity $e^{i\pi} = -1$. This is indeed a beautiful and useful result. But if we replace π with 2π it becomes even more elegant: $e^{i2\pi} = 1$. All that remains is to adopt a unique symbol for 2π , and a bunch of theorems suddenly become simpler and more natural. One suggestion is to use τ , actually. Let $\tau = 2\pi$. Now $e^{i\tau} = 1$. Now that's an elegant identity. [228]
52. The mathematically inclined might look at Box and Tiao's derivation, which is nice because it explicitly compares the Fisherian and Bayesian approaches to understanding uncertainty in σ . See pages xx-yy in Box and Tiao 1973. Like many derivations, it also shows that τ is a more natural parameter than is σ . [229]
53. It follows that the variance, σ^2 , is expected to have a squared-inverse-Gaussian density. You might know that a squared-Gaussian density is usually known as a chi-square (χ^2) density. So we expect the naive posterior of σ^2 to have an approximately inverse- χ^2 distribution, sometimes written $1/\chi^2$ or inv- χ^2 . Likewise, τ^2 should be approximately χ^2 . [230]
54. I first encountered this kind of example in Jaynes 1976, page 246. Jaynes himself credits G. David Forney's 1972 information theory course notes. Unfortunately, I have never been able to locate a copy of these notes. [234]
55. People sometimes like to use R^2 as an absolute measure of model fit. That is, if R^2 is close to 1, then the inference is that the model does a very good job of prediction. However, even this seemingly obvious inference can be suspect. For example, the easiest way to get a high R^2 is to bin and average the outcome variable before fitting the model. Averaging reduces the variation and increases R^2 . This is known as *ecological regression*. Not only is the R^2 value from such a regression suspect, but so too are inferences about the factors.
56. There are many discussions of bias and variance in the literature, some much more mathematical than others. A gentle place to start might be Forster and Sober's 1994 paper, which takes a general philosophy of science perspective on the problem of choosing a model family. Forster has a number of papers and chapters on similar issues. For a more technical treatment, I recommend Chapter 7 of Hastie, Tibshirani and Friedman's 2009 book, which explores and compares BIC, AIC, cross-validation and other

measures, all in the context of the bias-variance tradeoff. Importantly, HTF take a broader perspective on the tradeoff than is typical, because they have in mind a much broader array of model fitting approaches than is typical.

57. I do not mean to suggest that this is how real animals, including real people, actually learn. So that there is no doubt in the reader's mind about my attitudes in this area, I recommend Herbert Simon's bounded rationality tradition as the most fruitful approach to the study of how real organisms learn.

58. See examples in Hastie, Tibshirani and Friedman 2009 Chapter 7, for example.

59. Mean squared error (MSE) is just the average squared residual. In this context, the residual is measured as the difference between the actual value of the outcome and the prediction based upon the training set. MSE isn't the only way to measure accuracy, though. Again, see Chapter 7 of Hastie, Tibshirani and Friedman 2009.

60. Stone 1977 provides one proof of the convergence of AIC and leave-one-out cross-validation, under certain assumptions. Forward search on this paper to flush out more recent work. Cross-validation is actually a rather complex topic. In applied statistics, particularly in engineering and computer science, it is used extensively. But many academic statisticians have never used it. There are many different ways to conduct cross-validation, so if you end up wanting to try cross-validation, I recommend taking some time to compare algorithms, before you dive in.

61. Kass and Raftery 1995 is a good entry point into the literature. Note however that the end of this paper contains a section, 8.3, on AIC that is now widely recognized as misleading at best and plain wrong at worst. Best to ignore it. [255]

62. This remains a rarely-noted issue. I've seen this concern in a few places, notably in Malcolm Forster's publications. Andrew Gelman discussed it once on his blog and probably also in print. But in most technical discussions of Bayes Factors and the like, it is hardly mentioned. Note also the subtle and related issue of defining model family probabilities for nested models, also discussed in Forster.

63. Yang, Burnham and Anderson, Forster.

64. The label "BIC" does not appear in Schwartz' paper. [259]

65. Schwarz' 1978 derivation of BIC is brief and a little opaque. But the paper is still worth looking through, if only casually, so you appreciate how the original result was presented. A more patient but still quite technical derivation is given in Chapter 6 of Burnham and Anderson (2002, p XX). You might suspect that Burnham and Anderson are biased against BIC, since they engage in an extended argument against it. So it might help to also refer to the discussion and derivation of BIC in Raftery 1995, which is considerably more promotional. But see also Gelman and Rubin 1995 for an example of Bayesian statisticians who disagree with the BIC or Bayes Factor approach.

66. Although keep in mind the issue discussed in Section 5.4.5.2, page 256. It's not clear that it's easy to bury the prior, once we start comparing model families.

67. Akaike 1973. See also Akaike 1974. Like with BIC, the factor -2 is just there so that AIC is scaled like a deviance. See Burnham and Anderson 2002 for an expanded

treatment. [261]

68. Hurvich and Tsai 1989. AICc is a correction for a particular class of models. Other corrections exist. See Burnham and Anderson 2002, Chapters 6 and 7. [261]

69. Edwin T. Jaynes, a physicist, is usually considered the father of maximum entropy. See Jaynes 2003, which is actually a fantastic and unique book on probability and its relation to science and statistical inference. Jaynes passed away in 1998, not long before the book was completed. His friends, colleagues and former students labored to piece the manuscript together, patch up the missing bits, and publish it posthumously.

70. Shannon 1948. Readers interested in a formal derivation can consult any number of textbooks or papers. The venerable textbook *Elements of Information Theory*, by Cover and Thomas, is a common first course. I also recommend Jaynes' (2003, Chapter 11) presentation, because of the broader statistical discussion it is embedded in.

71. The only trick in computing H is to deal with the inevitable question of what to do when $p_i = 0$. The $\log(0) = -\infty$, which won't do. However, L'Hôpital's rule tells us that $\lim_{p_i \rightarrow 0} p_i \log(p_i) = 0$. So just assume that $0 \log(0) = 0$, when you compute H . In other words, events that never happen drop out. This is not really a trick. It follows from the definition of a limit. But it isn't obvious.

72. Elias citation — “Two Famous Papers” editorial.

73. I really wish I could say there is an accessible introduction to Maxent, at the level of most natural and social scientists' math training. If there is, I haven't found it yet. Jaynes 2003 is an essential source, but if your integral calculus is rusty, progress will be very slow. Better might be Stephen Frank's papers that explain the approach and relate it to common distributions in nature and therefore our data. Frank 2010, 2011a and 2011b. You can mainly hum over the maths in these and still get the major concepts.

74. Kullback and Leibler 1951. Note however that Kullback and Leibler did not name this measure after themselves. See Kullback 1987 for Solomon Kullback's reflections on the nomenclature. For what it's worth, Kullback and Leibler make it clear in their 1951 paper that Jeffreys had used this measure already in the development of Bayesian statistics.

75. See detailed discussion in Burnham and Anderson 2002, Chapter 7. They consider the case of gamma distributed error and find that as long as the error isn't too skewed, AICc performs very well. They have also simulated use of AICc with multinomial models, with good results. A lot more work needs to be done both evaluating the useful limits of AICc across kinds of models, as well as deriving AICc-like second order approximations for non-Gaussian models.

76. Yang 2005. [268]

77. I first encountered this approach in Burnham and Anderson 2002, Chapter 7. As they note, it is somewhat heuristic, because BIC and AICc are estimated to different accuracies. Also, the lurking problem of parameter priors influencing marginal likelihood remains unresolved. Akaike himself, in a rarely-read paper, was the first to note

that AIC differs from BIC in that it explicitly adopts priors designed to optimize prediction. See Akaike 1981. [270]

78. Burnham and Anderson 2002 is the most commonly cited treatment of the approach in the context of AIC-like metrics. See also Buckland et al. 1997. Adrian Raftery and colleagues have done a lot of work on Bayesian model averaging, which uses BIC and other Bayesian metrics instead.

79. cite page numbers in Burnham and Anderson ch7 ch6 for example

80. From Nunn and Puga 2011. [290]

81. Riley et al. 1999 [290]

82. This pattern is often labeled the Curse of Tippecanoe due to Harrison's military history earning him the nickname "Old Tippecanoe." Tippecanoe was the sight of a large battle between Native Americans and Harrison, in 1811. In popular imagination, Harrison was cursed by the Native Americans in the aftermath of the battle.

83. Sidanius & Pratto's book *Social Dominance* provides a nice review, and chasing its citations forward will yield more recent studies.

84. Ayres and Siegelman 1995. [297]

85. The default method for `mle2` at this point is `BFGS`, which has a smaller maximum number of iterations than `Nelder-Mead` does. You can also just pass the parameter `maxit=2000` to `mle2` to change the maximum iterations without changing the search strategy itself. Just change the 2000 to whatever number you like, within reason. [327]

86. Howell 2010 and Howell 2000. See also Lee and DeVore 1976. Much more raw data is available for download from <https://tspace.library.utoronto.ca/handle/1807/10395>. [330]

87. The paper usually credited as leading to Gauss' name being applied to the normal probability density is one in which Gauss carefully develops the density as a model of error, using what we'd now recognize as a Bayesian analysis with a weak prior. [334]

88. Grayson 1984. [334]

89. Faith 2008 is the paper. See Weaver et al. 2011 for discussion of flaws with the approach, in this data context. [334]

90. Some have begun to notice this. See Lyman and Ames 2007. [335]

91. Pearson 1900. By historical accident, this procedure was adopted by population geneticists in the first half of the 20th century, and so many college undergraduates are still forced to learn it to test for something called "Hardy-Weinberg Equilibrium." [337]

92. Perhaps more than anyone else, John Venn is responsible for sidelining Bayesian interpretations of probability. His 1866 book *The Logic of Chance*, which argued for the frequentist interpretation of probability, explicitly rejects Bayesian logic. Fisher may have taken lectures from Venn at Cambridge, acquiring his fierce opposition to inverse probability from him. See also Jaynes' 2003 speculations on Venn and Fisher.

Before Venn, George Boole (lived 1815–1864) also rejected inverse probability, partly based on concerns with uniform priors. Jaynes and others answered those concerns, but there was a long period during which few used or defended Bayesian inference. Note that in 1991, A. W. F. Edwards, a brilliant student of Fisher's and lifelong opponent of Bayesian inference, was still apparently ignorant that Boole's objections had been successfully rebutted decades earlier. [338]

93. Does Jaynes have a clear proof of this? Could cite Frank paper, but I think he just cites Jaynes. [346]

94. Common intro text is Singer and Willett 2003. [346]

95. No one knows precisely why this function has this name, but there are plausible guesses. The term appears to originate with Verhulst's 1838 paper on population growth, in which he called it *logistique*. At the time, the term logarithm and logistic were sometimes used interchangeably. So perhaps Verhulst was stating nothing more than this is a logarithmic curve. The term *logit* is a play on the term *probit*, which is itself an shortening of *probability unit*. Probit usually refers to the Gaussian quantile distribution. [350]

96. Silk et al. 2005. Nature 437:1357-1359. [350]

97. Kline and Boyd 2010. [362]

98. Because these models are built on cumulative densities, rather than regular densities, they are sometimes called *cumulative link* models. Remember, the “link” in a GLM refers to the function that connects the additive model to the outcome. So a “cumulative” link uses a cumulative density function, of log-odds in this case, to do so. [370]

99. Cushman et al. 2006. [375]

100. There are notable exceptions. Find a good introductory citation. [389]

101. R calls \mathcal{A} `shape1` and \mathcal{B} `shape2`. See `?dbeta`, for example. [391]

102. The built-in R function `dnbineg` provides the same likelihoods, but under a different guise. Using `dgammapois` is pedagogically useful, but functionally equivalent to `dnbineg`. [401]

103. Hurlbert (1984) is the typical citation in ecology and population biology. [408]

104. I adopt the terminology of Gelman 2005, who argues that the common term *random effects* hardly aids with understanding, for most people. Indeed, it seems to encourage misunderstanding, partly because the terms *fixed* and *random* mean different things to different statisticians. See pages 20–21 of Gelman's paper. I fully realize, however, that by trying to spread Gelman's alternative jargon, I am essentially spitting into a very strong wind. [411]

105. This fact has been understood much longer than multilevel models have been practical to use. See Stein (1955) for an influential paper. [415]

106. Box and Tiao, Bayesian Inference in Statistical Analysis, Addison-Wesley, (1973), section 5.1.2. [438]
107. Chung et al. (2011) [438]
108. Stone (1977). [445]
109. Fang (2011); Vaida and Blanchard (2005) [446]
110. Claeskens and Hjort (2008) provides examples of the diversity of metrics, including several that account for choice of focus. [446]
111. Spiegelhalter et al. (2002), Plummer (2008) [446]
112. Metropolis et al. (1953). The algorithm has been named after the first author of this paper, however it's not clear how each co-author participated in discovery and implementation of the algorithm. Among the other authors were Edward Teller, most famous as the father of the hydrogen bomb, and Marshall Rosenbluth, a renown physicist in his own right, as well as their wives Augusta and Arianna (respectively), who did much of the computer programming. Nicholas Metropolis lead the research group. [453]
113. Hastings (1970) [461]
114. Geman and Geman (1984) is the original. I recommend Casella and George (1992) as well. Note that Gibbs sampling is named after physicist and mathematician J. W. Gibbs, one of the founders of statistical physics. However, Gibbs died in the year 1903, long before even the Metropolis algorithm was discovered. Instead the strategy is named after Gibbs, both to honor him and in light of the algorithms connections to statistical physics. [464]

Bibliography

- Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In Petrov, B. N. and Csaki, F., editors, *Second International Symposium on Information Theory*, pages 267–281.
- Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.
- Akaike, H. (1981). Likelihood of a model and information criteria. *Journal of Econometrics*, 16:3–14.
- Ayres, I. and Siegelman, P. (1995). Race and gender discrimination in bargaining for a new car. *The American Economic Review*, 85(3):304–321.
- Bolker, B. (2008). *Ecological Models and Data in R*. Princeton University Press.
- Box, G. E. P. (1979). Robustness in the strategy of scientific model building. In Launer, R. and Wilkinson, G., editors, *Robustness in Statistics*. Academic Press, New York.
- Box, G. E. P. (1980). Sampling and Bayes' inference in scientific modelling and robustness. *Journal of the Royal Statistical Society A*, 143:383–430.
- Box, G. E. P. and Tiao, G. C. (1973). *Bayesian Inference in Statistical Analysis*. Addison-Wesley Pub. Co., Reading, Mass.
- Burnham, K. and Anderson, D. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer-Verlag, 2nd edition.
- Casella, G. and George, E. I. (1992). Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174.
- Cho, A. (2011). Superluminal neutrinos: Where does the time go? *Science*, 334(6060):1200–1201.
- Chung, Y., Rabe-Hesketh, S., Gelman, A., Liu, J., and Dorie, V. (2011). Avoiding boundary estimates in linear mixed models through weakly informative priors. U.C. Berkeley Division of Biostatistics Working Paper Series, Working Paper 284.
- Claeskens, G. and Hjort, N. (2008). *Model Selection and Model Averaging*. Cambridge University Press.

- Cohen, J. (1994). The Earth is round ($p < .05$). *American Psychologist*, 49(12):997–1003.
- Collins, H. M. and Pinch, T. (2011). *The Golem: What You Should Know about Science*. Cambridge University Press, 2nd edition.
- Cushman, F., Young, L., and Hauser, M. (2006). The role of conscious reasoning and intuition in moral judgment: Testing three principles of harm. *Psychological Science*, 17(12):1082–1089.
- Edwards, A. W. F. (1991). Bayesian reasoning in science. *Nature*, 352:386–387.
- Faith, J. T. (2008). Eland, buffalo, and wild pigs: were middle stone age humans ineffective hunters? *Journal of Human Evolution*, 55:24–36.
- Fang, Y. (2011). Asymptotic equivalence between cross-validations and akaike information criteria in mixed-effects models. *Journal of Data Science*, 9:15–21.
- Gelman, A. (2005). Analysis of variance: Why it is more important than ever. *The Annals of Statistics*, 33(1):1–53.
- Gelman, A., Carlin, J. C., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis*. Chapman & Hall/CRC, 2nd edition.
- Gelman, A. and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.
- Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741.
- Gigerenzer, G., Krauss, S., and Vitouch, O. (2004). The null ritual: What you always wanted to know about significance testing but were afraid to ask. In Kaplan, D., editor, *The Sage handbook of quantitative methodology for the social sciences*, pages 391–408. Sage Publications, Inc., Thousand Oaks.
- Gigerenzer, G., Swijtink, Z., Porter, T., Daston, L., Beatty, J., and Kruger, L. (1990). *The Empire of Chance: How Probability Changed Science and Everyday Life*. Cambridge University Press.
- Grayson, D. K. (1984). *Quantitative Zooarchaeology*. Academic Press, New York.
- Hastings, W. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.
- Howell, N. (2000). *Demography of the Dobe !Kung*. Aldine de Gruyter, New York.
- Howell, N. (2010). *Life Histories of the Dobe !Kung: Food, Fatness, and Well-being over the Life-span*. Origins of Human Behavior and Culture. University of California Press.

- Hurlbert, S. H. (1984). Pseudoreplication and the design of ecological field experiments. *Ecological Monographs*, 54:187–211.
- Jaynes, E. T. (1976). Confidence intervals vs bayesian intervals. In Harper, W. L. and Hooker, C. A., editors, *Foundations of Probability Theory, Statistical Inference, and Statistical Theories of Science*, page 175.
- Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press.
- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86.
- Lee, R. B. and DeVore, I., editors (1976). *Kalahari Hunter-Gatherers: Studies of the !Kung San and Their Neighbors*. Harvard University Press, Cambridge.
- Levins, R. (1966). The strategy of model building in population biology. *American Scientist*, 54.
- Lyman, R. L. and Ames, K. M. (2007). On the use of species-area curves to detect the effects of sample size. *Journal of Archaeological Science*, 34:1985–1990.
- McElreath, R. and Boyd, R. (2007). *Mathematical Models of Social Evolution: A guide for the perplexed*. University of Chicago Press.
- Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092.
- Mulkay, M. and Gilbert, G. N. (1981). Putting philosophy to work: Karl popper's influence on scientific practice. *Philosophy of the Social Sciences*, 11:389–407.
- Nunn, N. and Puga, D. (2011). Ruggedness: The blessing of bad geography in Africa. *Review of Economics and Statistics*.
- Palais, R. (2001). π is wrong! *The Mathematical Intelligencer*, 23(3):7–8.
- Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, Series 5 50(302):157–175.
- Plummer, M. (2008). Penalized loss functions for bayesian model comparison. *Biostatistics*, 9(3):523–539.
- Riley, S. J., DeGloria, S. D., and Elliot, R. (1999). A terrain ruggedness index that quantifies topographic heterogeneity. *Intermountain Journal of Sciences*, 5:23–27.
- Spiegelhalter, D. J., Best, N. G., Carlin, B. P., and va der Linde, A. (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society B*, 64:583–639.

- Stein, C. (1955). Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley Symposium of Mathematical Statistics and Probability*, volume 1, pages 197–206, Berkeley. University of California Press.
- Stone, M. (1977). An asymptotic equivalence of choice of model by cross-validation and Akaike's criterion. *Journal of the Royal Statistical Society B*, 39(1):44–47.
- Vaida, F. and Blanchard, S. (2005). Conditional akaike information for mixed-effects models. *Biometrika*, 92(2):351–370.
- Weaver, T. D., Steele, T. E., and Klein, R. G. (2011). The abundance of eland, buffalo, and wild pigs in middle and later stone age sites. *Journal of Human Evolution*, 60:309–314.
- Yang, Y. (2005). Can the strengths of aic and bic be shared? a conflict between model identification and regression estimation. *Biometrika*, 92(4):937–950.

Index

- Akaike (1973), 492, 497
Akaike (1974), 492, 497
Akaike (1981), 494, 497
Ayres and Siegelman (1995), 494, 497
Bolker (2008), 490, 497
Box and Tiao (1973), 489, 490, 497
Box (1979), 487, 497
Box (1980), 489, 497
Burnham and Anderson (2002), 492, 493, 497
Casella and George (1992), 496, 497
Cho (2011), 487, 497
Chung et al. (2011), 496, 497
Claeskens and Hjort (2008), 496, 497
Cohen (1994), 488, 497
Collins and Pinch (2011), 487, 498
Cushman et al. (2006), 495, 498
Edwards (1991), 495, 498
Faith (2008), 494, 498
Fang (2011), 496, 498
Gelman and Hill (2007), 490, 498
Gelman et al. (2004), 490, 498
Gelman (2005), 495, 498
Geman and Geman (1984), 496, 498
Gigerenzer et al. (1990), 487, 488, 498
Gigerenzer et al. (2004), 487, 488, 498
Grayson (1984), 494, 498
Hastings (1970), 496, 498
Howell (2000), 490, 494, 498
Howell (2010), 490, 494, 498
Hurlbert (1984), 495, 498
Jaynes (1976), 487, 499
Jaynes (2003), 487, 489, 494, 499
Kullback and Leibler (1951), 493, 499
Lee and DeVore (1976), 490, 494, 499
Levins (1966), 488, 499
Lyman and Ames (2007), 494, 499
McElreath and Boyd (2007), 487, 499
Metropolis et al. (1953), 496, 499
Mulkay and Gilbert (1981), 487, 499
Nunn and Puga (2011), 494, 499
Palais (2001), 491, 499
Pearson (1900), 494, 499
Plummer (2008), 496, 499
Riley et al. (1999), 494, 499
Spiegelhalter et al. (2002), 496, 499
Stein (1955), 495, 499
Stone (1977), 496, 500
Vaida and Blanchard (2005), 496, 500
Weaver et al. (2011), 494, 500
Yang (2005), 493, 500
- AIC, 260, 267
AICc, 260, 267
Akaike Information Criterion, *see also* AIC
apply, 282
- Bayes factor, 254
Bayes' theorem, 34, 36
bbmle, 70
bias-variance tradeoff, 238, 257, 265
BIC, 258, 267
- centering, 315
confidence interval, 81
cross-validation, 250
- highest posterior density interval, 84
- information theory, 261
Kullback-Leibler divergence, 264
- likelihood, 40, 49

function, 40
marginal, 43, 253
maximum likelihood estimate, 58
logistic, 354
logit, 354

Maxent, *see also* maximum entropy
maximum entropy, 263
maximum likelihood estimate, 58
`mle2`, 70
models, 32
model families, 236
averaging, 277
posterior probability of, 252
ranking, 273
nested, 238

Normal distribution
 τ , 227

overfitting, *see also* bias-variance
tradeoff

`pairs`, 211
posterior probability, 33
naive, 75, 78
sampling from, 78
prior probability, 38
as tuning, 39
objective, 39
subjective, 39, 40
uninformative, 54

`sample.naive.posterior`, 279
Schwartz criterion, *see also* BIC

underfitting, *see also* bias-variance
tradeoff

`update`, 292