DAVID ROBINSON

INTRODUCTION TO EMPIRICAL BAYES

Examples from Baseball Statistics

To my wife Dana, who by my estimation has always batted 1000.

Contents

1
Introduction

Have you ever tried to learn math from Wikipedia? As someone who uses a great deal of math in my work but doesn't consider himself a mathematician, I've always found it frustrating. Most math Wikipedia articles read to me like:

> The **eigentensors** are a family of parametric subspaces that are *diagonalizable* but not *orthogonal*. Their densities are a ring of consecutive Hilbert fields…

There are people who can learn math from descriptions like that, but I'm not one of them.

When I was learning mathematical statistics, what I found most useful weren't proofs and definitions, but rather intuitive explanations applied to simple examples. I was lucky to have great teachers who showed me the way, and helped guide me towards a thorough appreciation of statistical theory. I've thus endeavoured to make my own contribution to this educational philosophy: an intuitive explanation for a statistical concept that I feel is overdue for one.

This book introduces **empirical Bayes methods**, which are powerful tools for handling uncertainty across many observations. The methods introduced include estimation, credible intervals, A/B testing, hierarchical modeling, and other components of the philosophy. It teaches both the mathematical principles behind these and the code that you can adapt to explore your own data. It does so through the detailed extended of a single case study: that of **batting averages in baseball statistics**. I wrote it for people (like me) who need to understand and apply mathematical methods, but would rather work with real data than face down pages of equations.

1.1 Why this book?

This book originated as an answer to a Stack Exchange question, which asked for an intuitive explanation of the beta distribution. I

followed the answer with a series of posts on my blog Variance Explained, starting with the post Understanding empirical Bayes estimation (using baseball statistics).

As the blog series progressed, I realized I was building a narrative rather than a series of individual posts. Adapting it into a book allowed me to bring all the material into a consistent style and a cohesive order.[1] Among the changes I've made from the blog version is to add a brand new chapter (Chapter 10) about the Dirichlet and the multinomial, and to expand and improve material in several other chapters, including a new explanation of the conjugate prior in Section 2.3.

[1] For example, by choosing the Tufte book style I've been able to move some of the more extraneous material into margin notes like this one.

1.1.1 Why empirical Bayes?

Empirical Bayesian methods are an approximation to more exact methods, and they come with some controversy in the statistical community.[2] So why are they worth learning? Because in my experience, *empirical Bayes is especially well suited to the modern field of data science.*

[2] The name "empirical Bayes" is perhaps the most controversial element, since some have noted that it falsely implies other methods are "not empirical". I use this name throughout the book only for lack of an alternative.

First, one of the limitations of empirical Bayes is that its approximations become inaccurate when you have only a few observations. But modern datasets often offer thousands or millions of observations, such as purchases of a product, visits to a page, or clicks on an ad. Thus, there's often little difference between the solutions offered by traditional Bayesian methods and the approximations of empirical Bayes. (Chapter 12 offers evidence for this proposition, by simulating data and examining how accurate the empirical Bayes approach is.)

Secondly, empirical Bayes offers "shortcuts" that allow easy computation at scale. Full Bayesian methods that use Markov Chain Monte Carlo (MCMC) are useful when performance is less important than accuracy, such as analyzing a scientific study. However, production systems often need to perform estimation in a fraction of a second, and run them thousands or millions of times each day. Empirical Bayesian methods, such as the ones we discuss in this book, can make this process easy.

Empirical Bayes is not only useful, it is *undertaught*. Typical statistical education often jumps from simple explanations of Bayes' Theorem directly to full Bayesian models.[3] This book is not a full statistical treatment of the method, but rather an extended analysis of a single concrete example that will hopefully help you gain the intuition to work with the method yourself.

[3] Bayesian Data Analysis, by Gelman et al, is a classic example. It's a great text, but it focuses almost entirely on full Bayesian methods such as Markov Chain Monte Carlo (MCMC) sampling, while the empirical Bayesian approach is relegated to a few pages in Chapter 5.

1.1.2 Why baseball?

I've been a fan of baseball since long before I worked in statistics, so the example of batting averages came naturally to me, as did the extensions to the statistical topics introduced in the book. However, in truth this book isn't really about baseball.

I originally wanted to write about using empirical Bayes to analyze ad clickthrough rates (CTRs), which is a large part of my job as a data scientist at Stack Overflow. But I realized two things: the data I was analyzing was proprietary and couldn't be shared with readers, and it was very unlikely to be interesting except to programmers in similar positions.

I believe that mathematical explanations should happen alongside analyses of real and interesting data, and the Lahman baseball dataset certainly qualifies. It's thorough and accurate, it's easily accessed from R through the Lahman package (Friendly, 2015), and it allows us to address real sports issues.

In truth, I'm still not sure how accessible these explanations are to readers unfamiliar with baseball. I have friends with little patience for sports who have still gotten a great deal out of the blog series, and others who are alienated by the subject matter. If you're not a baseball fan, I would strongly encourage you to give the book a chance anyway: there is less discussion of baseball than you might expect, and I try to explain the material as I go.[4]

[4] Similarly, sports fans and baseball statisticians will probably find the book's discussions of baseball quite elementary and over-simplified, though they may still learn from the math.

1.2 Organization

This book is divided into four parts.

Part I: Empirical Bayes is an introduction to the beta-binomial model and to the fundamentals of the empirical Bayesian approach.

- Chapter 2 introduces the **beta distribution**, and demonstrates how it relates to the binomial, through the example of batting averages in baseball statistics.
- Chapter 3 describes **empirical Bayes estimation**, which we use to estimate each player's batting average while taking into account that some players have more evidence than others.
- Chapter 12.3 discusses **credible intervals**, which quantify the uncertainty in each estimate.

Part II: Hypothesis Testing discusses two examples of the Bayesian approach to testing specific claims.

- Chapter 11.4 describes the process of **hypothesis testing** in comparing each observation to a fixed point, as well as the Bayesian approach to controlling the **false discovery rate (FDR)**.

- Chapter 6 is a guide to **Bayesian A/B testing**, specifically the problem of comparing two players to determine which is the better batter.

Part III: Extending the Model introduces new complications, expanding the beta-binomial model to allow batting averages to depend on other factors. These kind of extensions show how flexible the empirical Bayes approach is in analyzing real data.

- Chapter 7 uses **beta-binomial regression** to show how the model can control for confounding factors that affect a player's performance
- Chapter 11.3 extends the regression example to take more information into account, such as time and whether a player is left-handed, as a particular case of a **hierarchical Bayesian model**.
- Chapter 11.5 discusses **mixture models**, where each player may come from one of several prior distributions, and shows how to use expectation-maximization algorithms to estimate the mixture.
- Chapter 10 introduces the **Dirichlet and the multinomial** distributions, which unlike the beta-binomial model can handle more than two categories of hits, and uses them to perform empirical Bayes estimation of slugging averages.

Part IV: Computation steps back to discuss practical issues in applying empirical Bayes models with R.

- Chapter 11 introduces **the ebbr package**, an R package developed alongside this book that makes it easy for you to use empirical Bayes on your own binomial data.
- Chapter 12 puts empirical Bayes to the test through **simulation**, by generating data where we know the "right answer" and then evaluating the performance of our statistical methods.
- Chapter 13 extends this simulation approach by **simulating replications** of the data to ensure the methods perform consistently well, and by varying the number of observations to show in what situations empirical Bayes can fail.

1.2.1 How to read this book

You don't need to know how to program to read this book, but R programmers may gain more practical skills from it if they follow along with the code. Every chapter starts with a *Setup* section with the code necessary to follow along with the code examples. This means you can jump into any chapter and replicate its code examples.

I don't generally show all the code necessary to replicate these chapters, only the parts others may want to apply to their own data.

In particular, I almost never show code to generate figures (readers comfortable with ggplot2 will likely be able to reproduce most without trouble). All the code associated with each chapter can be found in the book's GitHub repository.

When discussing mathematical methods, I generally prefer the **inclusive first person plural** (e.g. "Notice that we've improved our estimate…"). I use the first person singular when discussing my own preferences and decisions, such as in most of this introduction.

I tend to adopt a more casual and conversational tone than most mathematical texts. If you already find it unappealing, it's not going to get any better.

1.2.2 Acknowledgments

I thank the many readers who have provided feedback and support while this was being published as a blog series. I thank Sean Lahman for organizing the excellent Lahman baseball dataset, and Yihui Xie and the rest of RStudio for the development of the bookdown package that made this format possible.

I especially thank my family, including my father, grandfather and brother, for reading and commenting on the series of blog posts, and my wife Dana for her unending support and encouragement.

1.2.3 Colophon

The source of this book is published in the repository dgrtwo/empirical-bayes-book. If you find typos or bugs, I would greatly appreciate a pull request. This book is electronic only, so you can expect frequent updates.

You can compile the book with the bookdown package with the line

```
rmarkdown::render_site(encoding = 'UTF-8')
```

Part I

Empirical Bayes

2

The beta distribution

First, let's get to know the beta distribution, which plays an essential role in the methods described in this book. The beta is a probability distribution with two parameters α and β. It can take a couple of shapes (Figure 2.1), all of them constrained to between 0 and 1.

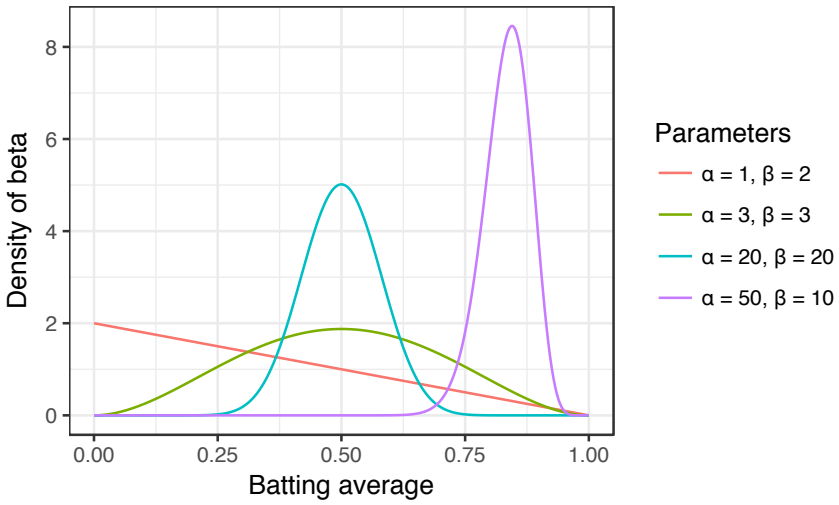

Figure 2.1: The density of the beta distribution for several selected combinations of parameters.

Some distributions, like the normal, the binomial, and the uniform, are described in statistics education alongside their real world interpretations and applications, which means beginner statisticians usually gain a solid understanding of them. But I've found that the beta distribution is rarely explained in these intuitive terms- if its usefulness is addressed at all, it's often with dense terms like "conjugate prior" and "order statistic."[1] This is a shame, because the intuition behind the beta is pretty cool.

In practice, the beta distribution is good at representing a probability distribution *of probabilities*- that is, it represents all the possible values of a probability when we don't know what that probability is. In this chapter, I'll introduce an example that we'll follow through the

[1] For example, mathematicians might start by teaching the probability density function of the beta that's shown in Figure 2.1, which happens to be $\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$, where B is the beta function. But I don't get much out of a definition like that; I like to see how a distribution is useful in practice.

rest of the book.

2.1 Batting averages

The sport of baseball has a long history of tracking and analyzing statistics, a field called sabermetrics. One of the most commonly used statistics in baseball is the batting average, which is calculated as the number of **hits (H)** divided by the number of **at-bats (AB)**:

$$\text{Batting Average} = \frac{H}{AB}$$

A player's batting average is therefore a percentage between 0 and 1. .270 (27%) is considered a typical batting average, while .300 (30%) is considered an excellent one.

Imagine we have a baseball player, and we want to predict what his season-long batting average will be. You might say we can just use his batting average so far- but this will be a very poor measure at the start of a season! If a player goes up to bat once and gets a single, his batting average is briefly 1.000, while if he strikes out or walks, his batting average is 0.000. It doesn't get much better if you go up to bat five or six times- you could get a lucky streak and get an average of 1.000, or an unlucky streak and get an average of 0, neither of which are a remotely good predictor of how you will bat that season.

Why is your batting average in the first few hits not a good predictor of your eventual batting average? When a player's first at-bat is a strikeout, why does no one predict that he'll never get a hit all season? Because we're going in with *prior expectations.* We know that in history, most batting averages over a season have hovered between something like .210 and .360, with some extremely rare exceptions on either side. We know that if a player gets a few strikeouts in a row at the start, that might indicate he'll end up a bit worse than average, but we know he probably won't deviate from that .210-.360 range. Bayesian statistics is a way of modeling these prior successes explicitly.

The number of hits a player gets out of his at-bats is an example of a **binomial distribution,** which models a count of successes out of a total.[2] Since it's a binomial, the best way to represent the prior expectations is with the beta distribution. The prior is representing, before we've seen the player take his first swing, what we roughly expect his batting average to be. The domain of the beta distribution is $(0, 1)$, just like a probability, so we already know we're on the right track- but the appropriateness of the beta for this task goes far beyond that.

[2] For example, we might say that out of 100 at-bats, the number of hits a player gets is distributed according to Binomial$(100, p)$, where p is the probability of each at-bat being a hit (and therefore is the batting average that we'd like to estimate). This is equivalent to flipping 100 coins, each with a p probability of heads.

2.2 Updating

We expect that the player's season-long batting average will be most likely around .27, but that it could reasonably range from .21 to .35. This can be represented with a beta distribution with parameters $\alpha = 81$ and $\beta = 219$. In later chapters we'll go into the details of how we can select parameters for a beta distribution, for now just know that they were chosen so that the mean and variance would be realistic for batting averages.

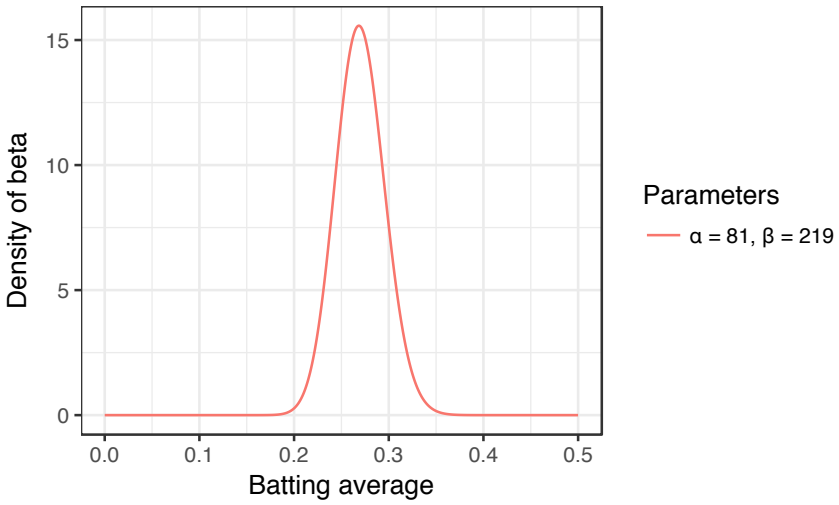

Figure 2.2: The density of the prior distribution: Beta(81, 219). The x-axis represents the distribution of possible batting averages, the y-axis represents the probability density: how likely the batting average is to fall at a particular point.

In Figure 2.2, the x-axis represents the distribution of possible batting averages, and the y-axis represents the probability density of the beta distribution: how likely the batting average is to fall at a particular point. The beta distribution is representing a probability distribution *of probabilities*.

Here's why the beta distribution is so appropriate for modeling the binomial. Imagine the player gets a single hit. His record for the season is now "1 hit; 1 at bat." We have to then **update** our probabilities- we want to shift this entire curve over just a bit to reflect our new information. This is the Bayesian philosophy in a nutshell: we start with a prior distribution, see some evidence, then update to a **posterior** distribution.

The math for proving this is a bit involved (it's shown here), the result is *very simple*. The new beta distribution will be:

$$\text{Beta}(\alpha_0 + \text{hits}, \beta_0 + \text{misses})$$

where α_0 and β_0 are the parameters we started with- that is, 81 and 219. Thus, in this case, α has increased by 1 (his one hit), while β

has not increased at all (no misses yet). That means our new distribution is $\text{Beta}(81+1, 219)$.

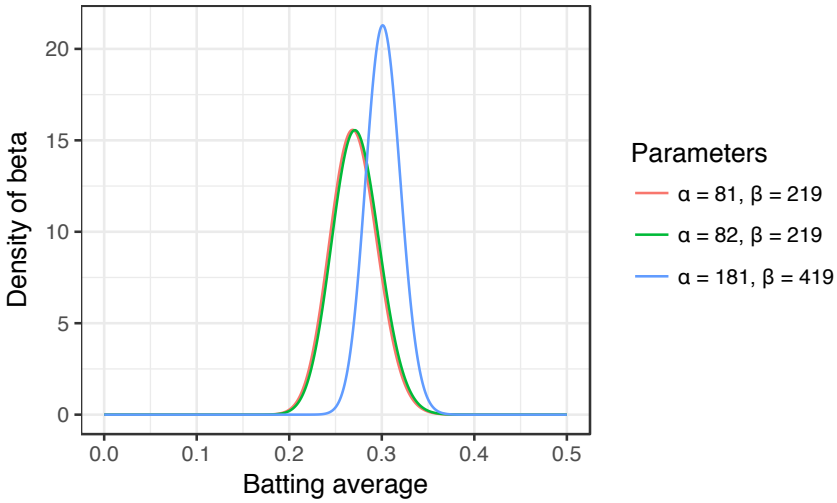

Figure 2.3: The density of the prior beta distribution, alongside the posterior after seeing 1 hit ($\text{Beta}(82, 219)$), or 100 hits out of 300 at-bats ($\text{Beta}(181, 419)$).

Figure 2.3 shows the prior distribution (red) and the posterior after a single hit (green). Notice that it has barely changed at all- the change is almost invisible to the naked eye! That's because one hit doesn't really mean anything. If we were a scout deciding whether to hire this player, we wouldn't have learned anything from the one hit.

However, the more the player hits over the course of the season, the more the curve will shift to accommodate the new evidence, and furthermore the more it will narrow to reflect that we have more proof. Let's say halfway through the season he has been up to bat 300 times, hitting 100 out of those times. The new distribution would be $\text{Beta}(81+100, 219+200) = \text{Beta}(181, 419)$.

Notice in Figure 2.3 that the new curve (in blue) is now both thinner and shifted to the right (higher batting average) than it used to be- we have a better sense of what the player's batting average is.

2.2.1 Posterior mean

One of the most interesting outputs of this formula is the expected value of the resulting beta distribution, which we can use as our new estimate. The expected value (mean) of the beta distribution is

$$\frac{\alpha}{\alpha+\beta}$$

Thus, after 100 hits of 300 at-bats, the expected value of the new beta distribution is

$$\frac{82+100}{82+100+219+200} = .303$$

Notice that it is lower than the raw estimate of $\frac{100}{100+200} = .333$, but higher than the estimate you started the season with $\frac{81}{81+219} = .270$: it is a combination of our prior expectations and our estimates. You might notice that this formula is equivalent to adding a "head start" to the number of hits and non-hits of a player: you're saying "start each player off in the season with 81 hits and 219 non hits on his record").

2.3 Conjugate prior

Why is it so easy to update the beta distribution from $\beta(\alpha_0, \beta_0)$ to $\beta(\alpha_0 + H, \beta_0)$? Because the beta distribution is the **conjugate prior** of the binomial: that just means that it's a particularly convenient distribution. The math for proving this is all available. But how can we get a feel for it?

Imagine you were a talent scout trying to estimate the "true batting average"- the probability of a hit- for a player with a 100/300 record. It would be nice to say "of all the players I've ever seen that batted 100/300, how good did they turn out to be?" This is unrealistic- we haven't seen very many players historically with that exact record. But when we have our **prior distribution**, we can build our own dataset of players, and look at the ones with 100/300.

Let's say we simulated ten million players. According to our prior expectations, they will be distributed according to a $\beta(81, 219)$ distribution (generated with the `rbeta` function in R). From each of them, we'll give them 300 chances at-bat, just like our 100/300 player.[3]

[3] Note that we keep the two vectors, `true_average` and `hits`, in a `data_frame`. This is a useful habit for "tidy" simulation since it can then be used with the dplyr and tidyr packages, and a practice we'll generally continue throughout the book.

```
library(dplyr)

num_trials <- 10e6

simulations <- data_frame(
  true_average = rbeta(num_trials, 81, 219),
  hits = rbinom(num_trials, 300, true_average)
)

simulations
```

```
## # A tibble: 10,000,000 × 2
##   true_average  hits
##          <dbl> <int>
## 1    0.2740295    89
## 2    0.3106993   106
## 3    0.2502260    77
## 4    0.2606805    73
```

```
## 5    0.2683601    73
## 6    0.3083013    85
## # ... with 1e+07 more rows
```

That's a lot of players, and we know the true batting average for every one of them. How many of them got 100/300, so we can compare them to our hypothetical player?

```
hit_100 <- simulations %>%
  filter(hits == 100)

hit_100
```

```
## # A tibble: 80,119 × 2
##   true_average  hits
##          <dbl> <int>
## 1    0.2940795   100
## 2    0.2951697   100
## 3    0.2917700   100
## 4    0.3228789   100
## 5    0.2804953   100
## 6    0.2923887   100
## # ... with 8.011e+04 more rows
```

What distribution of batting averages did these 100/300 players have have? How good was the median player? We can tell with a histogram (Figure 2.4).

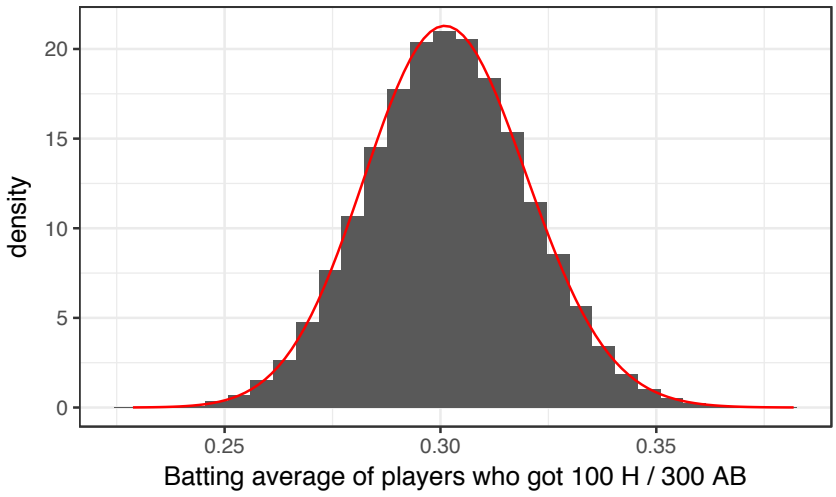

Figure 2.4: Histogram of the true batting average of all the players who got exactly 100 hits. Shown in red is the density of Beta(81+100, 219+200).

Notice the distribution of these batting averages. Our prior distribution may have contained batters with a true batting average of .2,

but they never got 100/300 in our simulations. And while it is easy for a batter with a .330 probability to get 100/300, there weren't many of them in the prior. And the median player who got 100/300 has a true batting average of about .3: that's our posterior estimate.

We can also confirm the math about the conjugate prior: the distribution of players precisely matches our $\beta(81+100, 219+200)$ posterior, shown in red. This shows what Bayesian updating is really doing- it's asking "out of our prior, what kinds of players would end up with evidence like this?"

What if the player had gotten 60 hits, or 80 hits, instead of 100? We could plot the density of each of those subsets of the simulations, as shown in Figure 2.5.[4]

```
simulations %>%
  filter(hits %in% c(60, 80, 100)) %>%
  ggplot(aes(true_average, color = factor(hits))) +
  geom_density() +
  labs(x = "True average of players with H hits / 300 at-bats",
       color = "H")
```

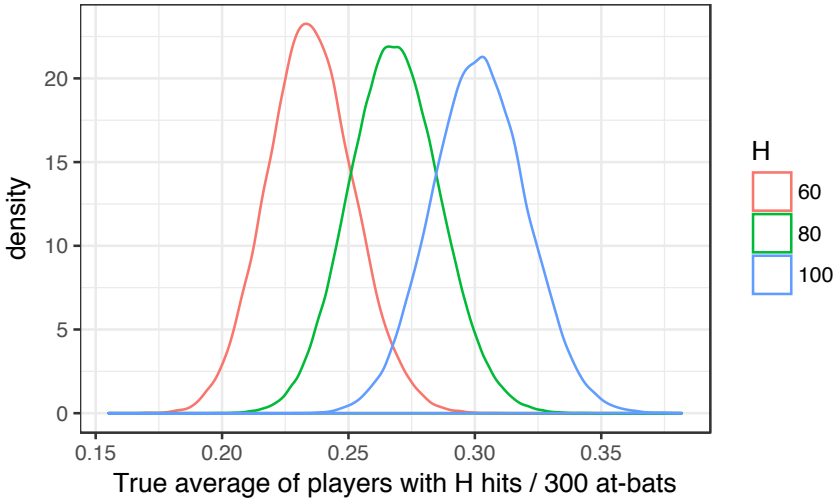

Figure 2.5: The density of the true batting average of subsets of the simulated players, specifically selecting players who had a record of 60/300, 80/300, or 100/300.

We can see that the shape of the posteriors are be similar (following a beta distribution), but that they shift to accomodate the evidence. Thus, the "simulate from the prior, pull out the ones who matched our evidence" approach was able to combine the prior and the evidence.

We won't need to keep generating millions of players in rest of this book: we'll just take the "add hits to α_0, add misses to β_0" approach for granted. We'll revisit the approach of simulating data in Chapter 12, where we'll use it to test and evaluate our methods. Simulation can be useful for more than just checking the math: when

[4] Notice that unlike the previous histogram, I chose to show this code as a way to help understand what is being visualized. Throughout the book I often make such judgment calls about when showing code can help explain a visualization.

Bayesian statisticians work with distributions that don't have a simple conjugate prior, they often use simulation approaches (such as Metropolis-Hastings or Gibbs sampling) that aren't that different from our approach here.

3

Empirical Bayes estimation

Which of these two proportions is higher: **4 out of 10**, or **300 out of 1000**? This sounds like a silly question. Obviously $4/10 = .4$, which is greater than $300/1000 = .3$.

But suppose you were a baseball recruiter, trying to decide which of two potential players is a better batter based on how many hits they get. One has achieved 4 hits in 10 chances, the other 300 hits in 1000 chances. While the first player has a higher proportion of hits, it's not a lot of evidence: a typical player tends to achieve a hit around 27% of the time, and this player's 4/10 could be due to luck. The second player, on the other hand, has a lot of evidence that he's an above-average batter.

This chapter introduces a very useful statistical method for estimating a large number of proportions, called **empirical Bayes estimation**. It's to help you with data that looks like this:

Success	Total
11	104
82	1351
2	26
0	40
1203	7592
5	166

A lot of data takes the form of these success/total counts, where you want to estimate a "proportion of success" for each instance. Each row might represent:

- **An ad you're running**: Which of your ads have higher click-through rates, and which have lower? Note that I'm not talking about running an A/B test comparing two options (which will be discussed in Chapter 6), but rather about ranking and analyzing a large list of choices.
- **A user on your site**: In my work at Stack Overflow, I might look at what fraction of a user's visits are to Javascript questions, to

guess whether they are a web developer. In another application, you might consider how often a user decides to read an article they browse over, or to purchase a product they've clicked on.

When you work with pairs of successes/totals like this, you tend to get tripped up by the uncertainty in low counts. 1/2 does not mean the same thing as 50/100; nor does 0/1 mean the same thing as 0/1000.[1]

In the last chapter, we learned how using the beta distribution to represent your prior expectations, and updating based on the new evidence, can help make your estimate more accurate and practical In this chapter, we'll explore the method we'll be using for the rest of this book: **empirical Bayes estimation**, where a beta distribution fit on all observations is then used to improve each individually. What's great about this method is that as long as you have a lot of examples, *you don't need to bring in prior expectations.*

[1] One common approach is to filter out all cases that don't meet some minimum. But this isn't always an option, since you could be throwing away useful information, and since it's not generally clear where to set a threshold.

3.1 Setup: the Lahman baseball dataset

In Chapter 2, we made some vague guesses about the distribution of batting averages across history. Here we'll instead work with real data, and come up with a quantitative solution, using the `Batting` dataset from the excellent Lahman package. We'll prepare and clean the data a little first, using dplyr and tidyr.[2]

[2] Pitchers tend to be the unusually weak batters and should be analyzed separately, so we remove them using dplyr's `anti_join`. We'll revisit pitchers in depth in Chapter 11.5.

```
library(dplyr)
library(tidyr)
library(Lahman)

# Filter out pitchers
career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(Pitching, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB)) %>%
  mutate(average = H / AB)

# Include names along with the player IDs
career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID") %>%
  dplyr::select(-playerID)
```

We've prepared a dataset of players' career averages, including their number of hits, at-bats, and batting averages.

```
career
```

```
## # A tibble: 9,342 × 4
##              name     H    AB average
##             <chr> <int> <int>   <dbl>
## 1      Hank Aaron  3771 12364  0.3050
## 2   Tommie Aaron   216   944  0.2288
## 3      Andy Abad     2    21  0.0952
## 4    John Abadie    11    49  0.2245
## 5 Ed Abbaticchio   772  3044  0.2536
## 6    Fred Abbott   107   513  0.2086
## # ... with 9,336 more rows
```

I wonder who the best batters in history were. Well, here are the ones with the highest batting average:

name	H	AB	average
Jeff Banister	1	1	1
Doc Bass	1	1	1
Steve Biras	2	2	1
C. B. Burns	1	1	1
Jackie Gallagher	1	1	1

Err, that's not really what I was looking for. These aren't the best batters, they're just the batters who went up once or twice and got lucky. How about the worst batters?

name	H	AB	average
Frank Abercrombie	0	4	0
Lane Adams	0	3	0
Horace Allen	0	7	0
Pete Allen	0	4	0
Walter Alston	0	1	0

Also not what I was looking for. That "average" is a really crummy estimate. **Let's make a better one!**

3.2 Step 1: Estimate a prior from all your data

Consider the distribution of batting averages across players. Any time we want to examine a distribution, we'll generally use a histogram (Figure 3.1).[3]

The first step of empirical Bayes estimation is to estimate a beta prior using this data. Estimating priors from the data you're currently analyzing is not the typical Bayesian approach- usually you decide

[3] For the sake of visualizing and fitting the prior distribution (though not for updating individual players), I've filtered for players with more than 500 at-bats, to reduce the amount of noise. We'll see how this model can be improved in Chapter 7.

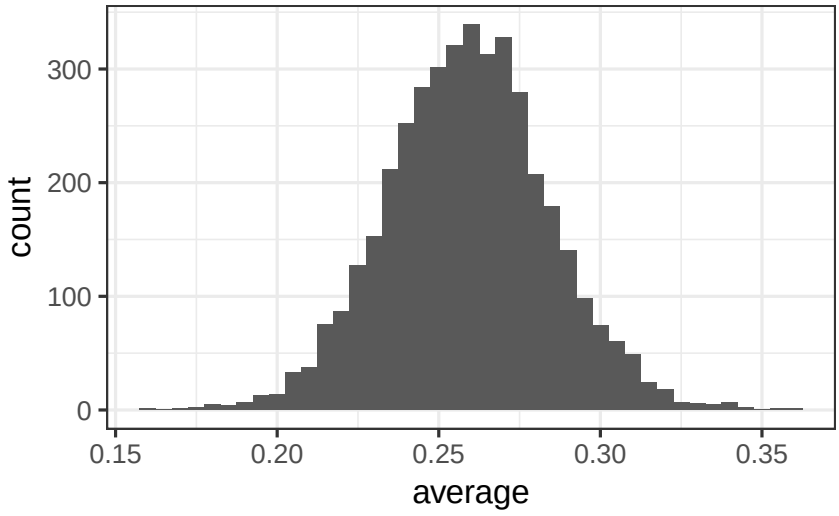

Figure 3.1: A histogram of the batting averages of all players with more than 500 at-bats.

on your priors ahead of time. There's a lot of debate and discussion about when and where it's appropriate to use empirical Bayesian methods, but it basically comes down to how many observations we have: if we have a lot, we can get a good estimate that doesn't depend much on any one individual. Empirical Bayes is an **approximation** to more exact Bayesian methods- and with the amount of data we have, it's a very good approximation.

So far, a beta distribution looks like a pretty appropriate choice based on the above histogram.[4] So we know we want to fit the following model:

$$X \sim \text{Beta}(\alpha_0, \beta_0)$$

We just need to pick α_0 and β_0, which we call "hyper-parameters" of our model. We can fit these using maximum likelihood: to see what parameters would maximize the probability of generating the distribution we see.[5]

```
library(stats4)

career_filtered <- career %>%
  filter(AB > 500)

# log-likelihood function
ll <- function(alpha, beta) {
  x <- career_filtered$H
  total <- career_filtered$AB
  -sum(VGAM::dbetabinom.ab(x, total, alpha, beta, log = TRUE))
}
```

[4] What would have made the beta a bad choice? Well, suppose the histogram had two peaks, or three, instead of one. Then we might have needed a mixture of betas, as will be introduced in Chapter 11.5, or an even more complicated model.

[5] There are many functions in R for fitting a probability distribution to data (`optim`, `mle`, `bbmle`, etc). You don't even have to use maximum likelihood: you could use the mean and variance, called the "method of moments". We'll choose to use the `mle` function, and to use `dbetabinom.ab`.

```
# maximum likelihood estimation
m <- mle(ll, start = list(alpha = 1, beta = 10), method = "L-BFGS-B",
                  lower = c(0.0001, .1))
ab <- coef(m)

alpha0 <- ab[1]
beta0 <- ab[2]
```

This comes up with $\alpha_0 = 101.359$ and $\beta_0 = 287.318$. We can see from the red curve in Figure 3.2 that it fits pretty well!

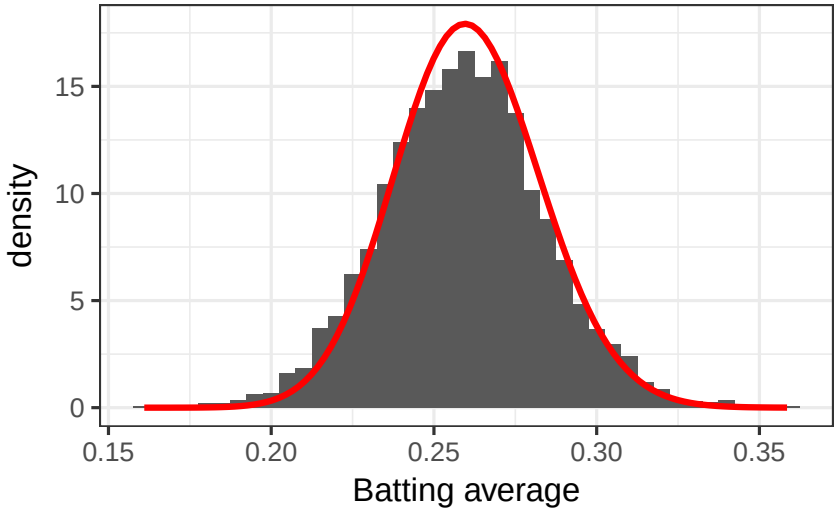

Figure 3.2: A histogram of the batting averages, showing the density of the beta distribution (fit with maximum likelihood) in red.

3.3 Step 2: Use that distribution as a prior for each individual estimate

Now when we look at any individual to estimate their batting average, we'll start with our overall prior, and update based on the individual evidence. I went over this process in detail in Chapter 2: it's as simple as adding α_0 to the number of hits, and $\alpha_0 + \beta_0$ to the total number of at-bats.

For example, consider our hypothetical batter from the introduction that went up 1000 times, and got 300 hits. We would estimate his batting average as:

$$\frac{300 + \alpha_0}{1000 + \alpha_0 + \beta_0} = \frac{300 + 101.4}{1000 + 101.4 + 287.3} = 0.289$$

How about the batter who went up only 10 times, and got 4 hits. We would estimate his batting average as:

$$\frac{4+\alpha_0}{10+\alpha_0+\beta_0} = \frac{4+101.4}{10+101.4+287.3} = 0.264$$

Thus, even though $\frac{4}{10} > \frac{300}{1000}$, we would guess that the $\frac{300}{1000}$ batter is better than the $\frac{4}{10}$ batter!

Performing this calculation for all the batters is simple enough. We'll call this an "empirical Bayes estimate", saving it in the `eb_estimate` column.

```
career_eb <- career %>%
  mutate(eb_estimate = (H + alpha0) / (AB + alpha0 + beta0))
```

3.4 Results

Now that we have a better estimate for each player's batting average, we can ask who the best batters are.

name	H	AB	average	eb_estimate
Rogers Hornsby	2930	8173	0.358	0.354
Shoeless Joe Jackson	1772	4981	0.356	0.349
Ed Delahanty	2596	7505	0.346	0.342
Billy Hamilton	2158	6268	0.344	0.339
Willie Keeler	2932	8591	0.341	0.338

Who are the *worst* batters?

name	H	AB	average	eb_estimate
Bill Bergen	516	3028	0.170	0.181
Ray Oyler	221	1265	0.175	0.195
Henry Easterday	203	1129	0.180	0.201
John Vukovich	90	559	0.161	0.202
George Baker	74	474	0.156	0.203

Notice that in each of these cases, empirical Bayes didn't simply pick the players who had 1 or 2 at-bats and hit on 100% of them. It found players who generally batted well, or poorly, across a long career. What a load off of our minds: we can start using these empirical Bayes estimates in downstream analyses and algorithms, and not worry that we're accidentally letting 0/1 or 1/1 cases ruin everything.

Overall, let's see how empirical Bayes changed all of the batting average estimates using a scatter plot (Figure 3.3).

The horizontal dashed red line marks $y = \frac{\alpha_0}{\alpha_0+\beta_0} = 0.261$. That's what we would guess someone's batting average was if we had *no* evidence at all (if both H and AB were 0). Notice that points above that line tend to move down towards it, while points below it move up.

The diagonal red line marks $x = y$. Points that lie close to it are the ones that didn't get shrunk at all by empirical Bayes. Notice that

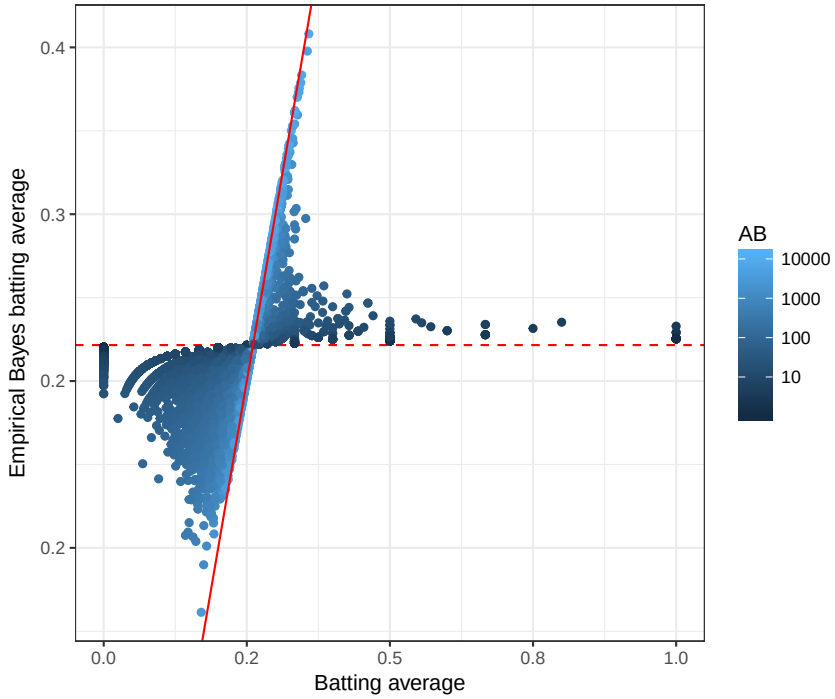

Figure 3.3: A comparison between the raw batting average ($\frac{H}{AB}$) and the empirical Bayes estimate ($\frac{H+\alpha_0}{AB+\alpha_0+\beta_0}$) for all batters.

they're the ones with the highest number of at-bats (the brightest blue): they have enough evidence that we're willing to believe the naive batting average estimate.

This process is often (including in the rest of this book) called **shrinkage**: the process of moving all our estimates towards the average. How much it moves these estimates depends on how much evidence we have: if we have very little evidence (4 hits out of 10) we move it a lot, if we have a lot of evidence (300 hits out of 1000) we move it only a little. That's shrinkage in a nutshell: *Extraordinary outliers require extraordinary evidence.*

3.5 So easy it feels like cheating

There are two steps in empirical Bayes estimation:

1. Estimate the overall distribution of your data.
2. Use that distribution as your prior for estimating each average.

Step 1 can be done once, "offline"- analyze all your data and come up with some estimates of your overall distribution. Step 2 is done for each new observation you're considering. You might be estimating the success of a post or an ad, or classifying the behavior of a user in terms of how often they make a particular choice.

And because we're using the beta and the binomial, consider how *easy* that second step is. All we did was add one number to the successes, and add another number to the total. If you work in a company that puts models into production, you could build that into your system with a single line of code that takes nanoseconds to run.

```
// We hired a Data Scientist to analyze our Big Data
// and all we got was this lousy line of code.
float estimate = (successes + 101.4) / (total + 388.7);
```

That really is so simple that it feels like cheating- like the kind of "fudge factor" you might throw into your code, with the intention of coming back to it later to do some real Machine Learning.

I bring this up to disprove the notion that statistical sophistication necessarily means dealing with complicated, burdensome algorithms. This Bayesian approach is based on sound principles, but it's still easy to implement. Conversely, next time you think "I only have time to implement a dumb hack," remember that you can use methods like these: it's a way to choose your fudge factor. Some dumb hacks are better than others!

But when anyone asks what you did, remember to call it "empirical Bayesian shrinkage towards a Beta prior." We statisticians have to keep up appearances.

4
Credible intervals

In Chapter 3, we explored the method of empirical Bayes estimation to calculate useful proportions out of many pairs of success/total counts (e.g. 0/1, 3/10, 235/1000). If we a batter gets with 0 hits in 2 at-bats, or 1 hit in 1 at-bat, we know we can't trust those proportions, and we can instead use information from the overall distribution (in the form of a prior) to improve our guess. Empirical Bayes gives a single value for each player that can be reliably used as an estimate.

But sometimes we want to know more than just our "best guess," and instead wish to know how much uncertainty is present in our point estimate. In many cases like this, statisticians would use a binomial proportion confidence interval[1], but this doesn't bring in information from our whole dataset the way that empirical Bayesian estimation does. For example, the confidence interval for someone who gets 1 hit out of 3 at-bats would be $(0.008, 0.906)$. We can indeed be quite confident that that interval contains the true batting average... but from our knowledge of batting averages, we could have drawn a much tighter interval than that! There's no way that the player's real batting average is .1 or .8: it probably lies in the .2-.3 region that most other players' do.

Here I'll show how to compute a **credible interval** using the empirical Bayes method. This will have a similar improvement relative to confidence intervals that the empirical Bayes estimate had to a raw batting average.

[1] For example, when the news reports that a political poll has a "plus or minus 3 percent margin of error", they're usually using this kind of confidence interval. You can compute it with the `prop.test` function in R.

4.1 Setup

We'll start with code that sets up the objects used and analyzed in this post.[2]

[2] From now on, each chapter will start with a section like this with code that sets up the chapter. If you're following along in code, this saves you from going back to earlier chapters to keep track of variables.

```
library(dplyr)
library(tidyr)
library(Lahman)
```

```
career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(Pitching, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB)) %>%
  mutate(average = H / AB)

career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID")

# values estimated by maximum likelihood in Chapter 3
alpha0 <- 101.4
beta0 <- 287.3

career_eb <- career %>%
    mutate(eb_estimate = (H + alpha0) / (AB + alpha0 + beta0))
```

The end result of this process is the `eb_estimate` variable in the `career_eb` dataset. This gives us a new estimate for each player's batting average; what statisticians call a **point estimate**. Recall that these new values tend to be pushed towards the overall mean (giving this the name "shrinkage").

This shrunken value is generally more useful than the raw estimate: we can use it to sort our data, or feed it into another analysis, without worrying too much about the noise introduced by low counts. But there's still uncertainty in the empirical Bayes estimate, and *the uncertainty is very different for different players,* with more uncertainty for players with few at-bats, and less uncertainty for players with many. We may want not only a point estimate, but an interval of possible batting averages; one that will be wide for players we know very little about, and narrow for players with more information. Luckily, the Bayesian approach has a method to handle this.

4.2 Posterior distribution

Consider that what we're really doing with empirical Bayes estimation is computing two new values for each player: α_1 and β_1. These are the *posterior* shape parameters for each player's distribution, after the prior (which was estimated from the whole dataset) has been updated based on each player's evidence. They are computed as $\alpha_1 = \alpha_0 + H$ and $\beta_1 = \beta_0 + AB - H$.

```
career_eb <- career_eb %>%
  mutate(alpha1 = alpha0 + H,
         beta1 = beta0 + AB - H)
```

Since we have these two parameters for each player's beta distribution, we can visualize the density of the posterior distribution for each, using the `dbeta` function in R. I'll pick a few of my favorites from the 1998 Yankee lineup (Figure 4.1).

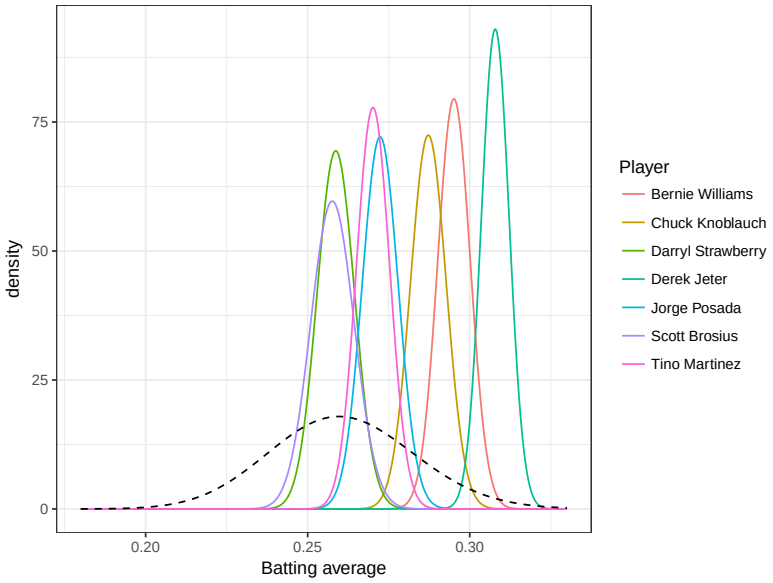

Figure 4.1: The posterior beta distribution for each of seven Yankee batters. The prior distribution is shown as a dashed curve.

Each of these curves is our probability distribution of what the player's batting average could be, after updating based on that player's performance. The empirical Bayes estimate that we reported in Chapter 3 is simply the peak of each[3], but this distribution is what we're really estimating.

[3] Technically, the peak (or **mode**) of the beta distributions is $\frac{\alpha-1}{\alpha+\beta-2}$, not the empirical Bayes estimate (or mean) of $\frac{\alpha}{\alpha+\beta}$, but those two become indistinguishable for large α and β.

4.3 Credible intervals

These density curves are hard to interpret visually, especially as the number of players increases, and it can't be summarized into a table or text. We'd instead prefer to create a credible interval, which says that some percentage (e.g. 95%) of the posterior distribution lies within an particular region. For example, the credible interval for Derek Jeter is shown in Figure 4.2.

You can compute the edges of the credible interval quite easily using the qbeta (quantile of beta) function in R. We just provide it the posterior `alpha1` and `beta1` parameters for each player.

```
yankee_1998_career <- yankee_1998_career %>%
```

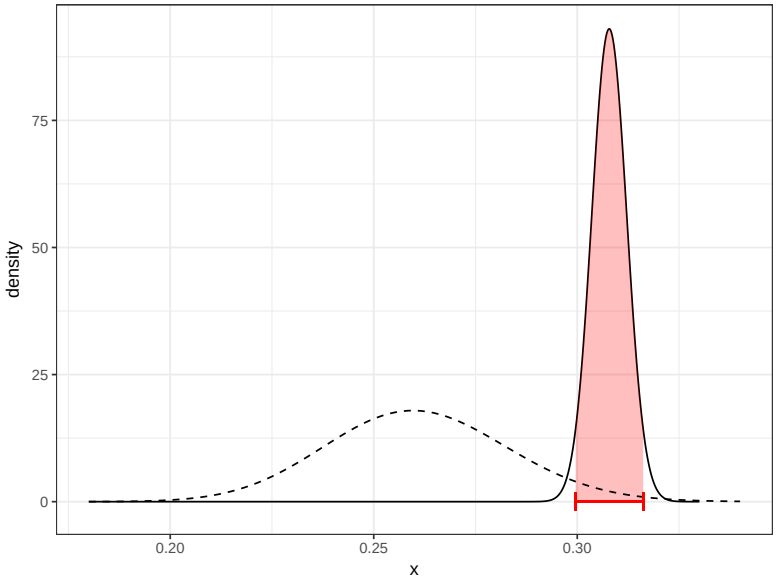

Figure 4.2: The posterior beta distribution for Derek Jeter (3465 H / 11195 AB), with the 95% credible interval highlighted in red. The prior is shown as a dashed curve.

```
  mutate(low  = qbeta(.025, alpha1, beta1),
         high = qbeta(.975, alpha1, beta1))
```

playerID	name	H	AB	average	low	high
brosisc01	Scott Brosius	1001	3889	0.257	0.245	0.271
jeterde01	Derek Jeter	3465	11195	0.310	0.300	0.316
knoblch01	Chuck Knoblauch	1839	6366	0.289	0.277	0.298
martiti02	Tino Martinez	1925	7111	0.271	0.260	0.280
posadjo01	Jorge Posada	1664	6092	0.273	0.262	0.283
strawda01	Darryl Strawberry	1401	5418	0.259	0.248	0.270
willibe02	Bernie Williams	2336	7869	0.297	0.285	0.305

These credible intervals can be visualized in a plot with points and errorbars like Figure 4.3. The vertical dashed red line is $\frac{\alpha_0}{\alpha_0+\beta_0}$: the mean batting average across history (based on our beta fit), that everything was being shrunk towards.

```
yankee_1998_career %>%
  mutate(name = reorder(name, eb_estimate)) %>%
  ggplot(aes(eb_estimate, name)) +
  geom_point() +
  geom_errorbarh(aes(xmin = low, xmax = high)) +
  geom_vline(xintercept = alpha0 / (alpha0 + beta0), color = "red", lty = 2) +
  xlab("Estimated batting average (w/ 95% interval)") +
  ylab("Player")
```

Figure 4.1, which showed each player's posterior beta distribution, technically communicated more information, but this is far more readable, and communicates most of what we're interested in: the center of

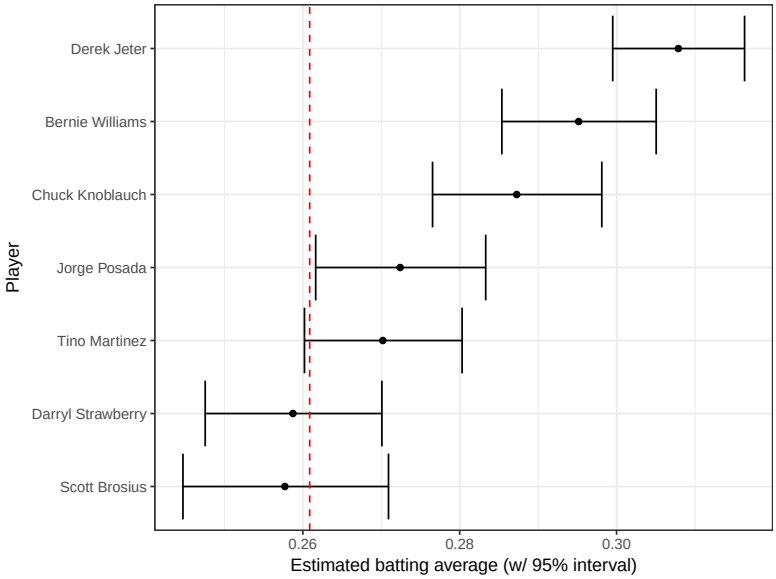

Figure 4.3: 95% credible intervals for each of seven Yankees. The points are the empirical Bayes estimates for each, and the vertical red line is the prior mean.

the distribution and the typical range of values.

4.4 Credible intervals and confidence intervals

Note that posterior credible intervals are similar to frequentist confidence intervals, but they are not the same thing. There's a philosophical difference, in that frequentists treat the true parameter as fixed while Bayesians treat it as a probability distribution. You can find one great explanation of the distinction in this Cross Validated post.

But there's also a very practical difference, in that credible intervals take prior information into account. Suppose we took 20 random players and constructed both frequentist confidence intervals and posterior credible intervals for each (Figure 4.4).

These are sorted in order of how many times a player went up to bat (thus, how much information we have about them). Notice that once there's enough information, the credible intervals and confidence intervals are nearly identical. But in cases where batters got 1 or 2 hits out of 10, the credible interval is much narrower than the cred interval. This is because empirical Bayes brings in our knowledge from the full data, just as it did for the point estimate, so that we know it's not plausible for a batter to have an average close to 0 or as high as .5.

4.4.1 Calculating confidence intervals

The relationship between credible and confidence intervals is particularly deep in the case of estimating a proportion. Two of the most

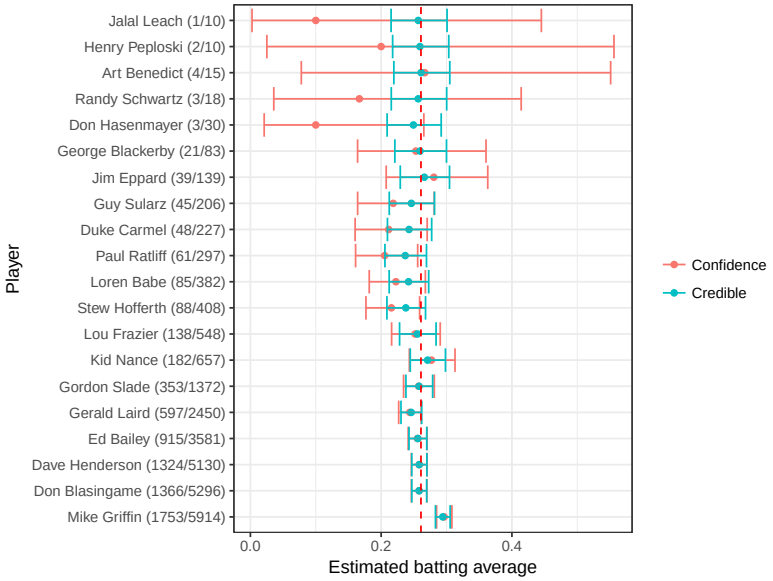

Figure 4.4: Frequentist confidence intervals and Bayesian credible intervals for 20 random players. Each player's batting record is shown next to their name, and they are ordered in terms of increasing AB.

common methods for constructing frequentist confidence intervals, Clopper-Pearson (the default in R's `binom.test` function, as shown in Figure 4.4) and Jeffreys, actually use the quantiles of the Beta distribution in very similar ways, which is the reason credible and confidence intervals start looking identical once there's enough information.

Method	α_{low}	β_{low}	α_{high}	β_{high}
Credible interval	$\alpha_0 + x$	$\beta_0 + n - x$	$\alpha_0 + x$	$\beta_0 + n - x$
Jeffreys	$1/2 + x$	$1/2 + n - x$	$1/2 + x$	$1/2 + n - x$
Clopper-Pearson	x	$n - x + 1$	$x + 1$	$n - x$

Table 4.1: Formulae for computing a Bayesian credible interval, or a Jeffreys or Clopper-Pearson confidence interval.

Table 4.1 shows the formulae for each of these intervals in terms the parameters one would use to the `qbeta` function in R. A particular 95% interval would be calculated as:

$$[\text{qbeta}(0.025, \alpha_{\text{low}}, \beta_{\text{low}}), \text{qbeta}(0.975, \alpha_{\text{high}}, \beta_{\text{high}})]$$

For example, the Clopper-Pearson confidence interval for a player with 10 hits out of 30 at-bats would be `qbeta(0.025, 10, 30 - 10 + 1)`=0.173 for the lower bound, and `qbeta(0.975, 10 + 1, 30 - 10)`=0.528 for the upper bound.

Notice that the Jeffreys prior is identical to the Bayesian credible interval when $\alpha_0 = \frac{1}{2}; \beta_0 = \frac{1}{2}$. This is called an **uninformative prior** or a Jeffreys prior, and is basically pretending that we know *nothing* about batting averages. The Clopper-Pearson interval is a bit odder, since its priors are different for the lower and upper bounds.[4] This makes it slightly more conservative (wider) than the Jeffrey's prior, with a lower lower bound and a higher upper bound.

[4] In fact Clopper-Pearson isn't using a proper prior for either, since it is effectively setting $\alpha_0 = 0$ for the low bound and $\beta_0 = 0$ for the high bound, which are both illegal parameters for the beta distribution.

One important mathematical observation is that the Bayesian credible interval, the Clopper-Pearson interval, and the Jeffreys interval all start looking more and more identical when:

- the evidence is more informative (large n), or
- the prior is less informative (small α_0, small β_0)

This fits what we saw in Figure 4.4. Bayesian methods are especially helpful relative to frequentist methods when the prior makes up a relatively large share of the information.

Like most applied statisticians, I don't consider myself too attached to the Bayesian or frequentist philosophies, but rather use whatever method is useful for a given situation. But while I've seen non-Bayesian approaches to point estimate shrinkage[5], I haven't yet seen a principled way of shrinking confidence intervals by sharing information across observations. This makes empirical Bayes posteriors quite useful!

It is not necessarily the case for all methods that there is a close equivalent between a confidence interval and a credible interval with an uninformative prior. But it happens more often than you might think! As Rasmuth Bååth puts it, "Inside every classical test there is a Bayesian model trying to get out."

[5] One example of a frequentist approach to shrinkage is James-Stein estimation, which was one of the first rigorous methods to take advantage of "sharing information across observations."

Part II

Hypothesis testing

5
Hypothesis testing and FDR

So far, we've been able to construct both point estimates and credible intervals from on each player's batting performance, while taking into account that we have more information about some players than others.

But sometimes, rather than estimating a value, we're looking to answer a yes or no question about each hypothesis, and thus classify them into two groups. For example, suppose we were constructing a Hall of Fame, where we wanted to include all players that have a batting probability (chance of getting a hit) greater than .300. We want to include as many players as we can, but we need to be sure that each belongs.

In the case of baseball, this is just for illustration- in real life, there are a lot of other, better metrics to judge a player by! But the problem of *hypothesis testing* appears whenever we're trying to identify candidates for future study. We need a principled approach to decide which players are worth including, and that can handle multiple testing problems.[1] To solve this, we're going to apply a Bayesian approach to a method usually associated with frequentist statistics, namely **false discovery rate control**.

This approach is very useful outside of baseball, and even outside of beta/binomial models. We could be asking which genes in an organism are related to a disease, which answers to a survey have changed over time, or which counties have an unusually high incidence of a disease. Knowing how to work with posterior predictions for many observations, and come up with a set of candidates for further study, is an essential skill in data science.

[1] Multiple testing is a common issue in statistics, based on the fact that if you test many hypotheses, a few will "get lucky" and appear positive just by chance. For example, if you tested a thousand fair coins, a few might get 8 heads in a row; that wouldn't mean they were rigged coins.

5.1 Setup code

As usual, we start with code that sets up the variables analyzed in this chapter.

```
library(dplyr)
library(tidyr)
library(Lahman)

career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(Pitching, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB)) %>%
  mutate(average = H / AB)

career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID")

# values estimated by maximum likelihood in Chapter 3
alpha0 <- 101.4
beta0 <- 287.3

career_eb <- career %>%
  mutate(eb_estimate = (H + alpha0) / (AB + alpha0 + beta0),
         alpha1 = H + alpha0,
         beta1 = AB - H + beta0)
```

5.2 Posterior Error Probabilities

Consider the legendary player Hank Aaron. His career batting average is 0.3050, but we'd like to base our Hall of Fame admission on his "true probability" of hitting. Should he be permitted in our >.300 Hall of Fame?

When Aaron's batting average is shrunken by empirical Bayes (Chapter 3), we get an estimate of 0.3037. We thus *suspect* that his true probability of hitting is higher than .300, but we're not necessarily certain of that. As we did in Chapter 12.3, let's take a look at his posterior beta distribution (Figure 5.1).

We can see that there is a nonzero probability (shaded) that his true probability of hitting is less than .3. We can calulate this probability with the cumulative distribution function (CDF) of the beta distribution, which in R is computed by the pbeta function.

```
career_eb %>%
  filter(name == "Hank Aaron")
```

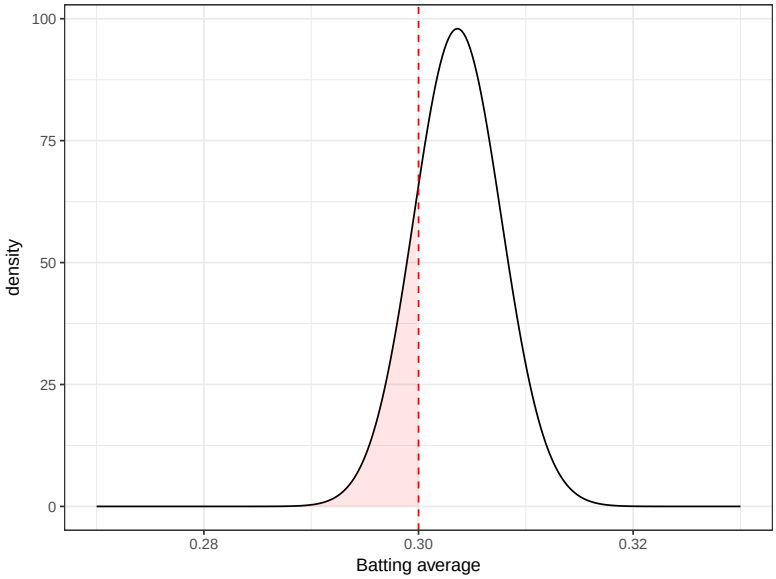

Figure 5.1: The posterior distribution for the true batting average of Hank Aaron (3771 H / 12364 AB). The batting average .3 is marked as a dashed red line, and the region where his batting average is less than .3 is shaded.

```
## # A tibble: 1 × 8
##    playerID       name     H    AB average
##       <chr>      <chr> <int> <int>   <dbl>
## 1 aaronha01 Hank Aaron  3771 12364   0.305
## # ... with 3 more variables:
## #   eb_estimate <dbl>, alpha1 <dbl>,
## #   beta1 <dbl>
```

```
pbeta(.3, 3850, 8818)
```

```
## [1] 0.169
```

This probability that he doesn't belong in the Hall of Fame is called the **Posterior Error Probability**, or **PEP**.[2] It's equally straightforward to calculate the PEP for every player, just like we calculated the credible intervals for each player in Chapter 12.3.

[2] We could just as easily have calculated the probability Aaron *does* belong, which is Posterior Inclusion Probability, or PIP. (Note that PIP = 1 − PEP) The reason we chose to measure the PEP rather than the PIP will become clear once we introduce the false discovery rate.

```
career_eb <- career_eb %>%
    mutate(PEP = pbeta(.3, alpha1, beta1))
```

What can examine the distribution of the PEP across players in Figure 5.2. Unsurprisingly, for most players, it's almost certain that they *don't* belong in the hall of fame: we know that their batting averages are below .300. If they were included, it is almost certain that they would be an error. In the middle are the borderline players: the ones where we're not sure. And down there close to 0 are the rare but proud players who we're (effectively) certain belong in the hall of fame.

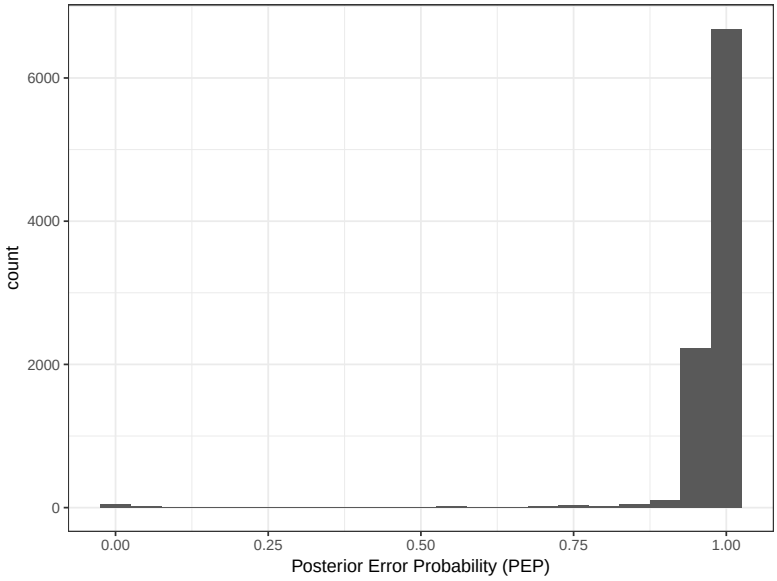

Figure 5.2: Histogram of posterior error probability (PEP) values across all players.

Note that the PEP is closely related to the estimated batting average, as shown in Figure 5.3. Notice that crossover point: to have a PEP less than 50%, you need to have a shrunken batting average greater than .300. That's because the shrunken estimate is the center of our posterior beta distribution (the "over/under" point). If a player's shrunken estimate is above .300, it's more likely than not that their true average is as well. And the players we're not sure about (PEP $\approx$ .5) have batting averages very close to .300.

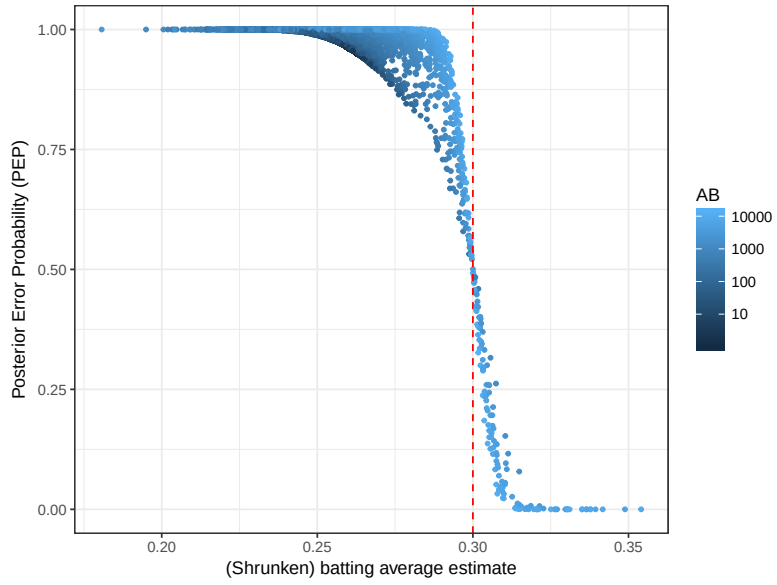

Figure 5.3: Relationship of the shrunken batting average and the posterior error probability of whether the player's batting average is $> .3$. The value .300 is marked as a dashed red line.

Notice also the relationship between the number of at-bats (the

amount of evidence) and the PEP. If a player's shrunken batting average is .28, but he hasn't batted many times, it is still possible his true batting average is above .3 (the credible interval is wide). However, if a player with a score of .28 has a high AB (light blue), the credible interval becomes thinner, we become confident that the true probability of hitting is under .3, and the PEP goes up to 1.

5.3 False Discovery Rate

Now we want to set some threshold for inclusion in our Hall of Fame. This criterion is up to us: what kind of goal do we want to set? There are many options, but I'll propose one common in statistics: *let's try to include as many players as possible, while ensuring that no more than 5% of the Hall of Fame was mistakenly included.* Put another way, we want to ensure that *if you're in the Hall of Fame, the probability you belong there is at least 95%.*

This criterion is called **false discovery rate control**. It's particularly relevant in scientific studies, where we might want to come up with a set of candidates (e.g. genes, countries, individuals) for future study. There's nothing special about 5%: if we wanted to be more strict, we could choose the same policy, but change our desired FDR to 1% or .1%. Similarly, if we wanted a broader set of candidates to study, we could set an FDR of 10% or 20%.

Let's start with the easy cases. Who are the players with the lowest posterior error probability?

rank	name	H	AB	eb_estimate	PEP
1	Rogers Hornsby	2930	8173	0.354	0
2	Ed Delahanty	2596	7505	0.342	0
3	Willie Keeler	2932	8591	0.338	0
4	Shoeless Joe Jackson	1772	4981	0.349	0
5	Nap Lajoie	3242	9589	0.335	0
6	Tony Gwynn	3141	9288	0.335	0
7	Harry Heilmann	2660	7787	0.338	0
8	Lou Gehrig	2721	8001	0.336	0
9	Billy Hamilton	2158	6268	0.339	0
10	Eddie Collins	3315	9949	0.330	0

These players are a no-brainer for our Hall of Fame: there's basically no risk in including them. But suppose we instead tried to include the top 100. What do the 90th-100th players look like?

rank	name	H	AB	eb_estimate	PEP
90	Mike Piazza	2127	6911	0.305	0.164
91	Rip Radcliff	1267	4074	0.307	0.168
92	Denny Lyons	1333	4294	0.306	0.174
93	Don Mattingly	2153	7003	0.305	0.176
94	Hank Aaron	3771	12364	0.304	0.185
95	John Stone	1391	4494	0.306	0.196
96	Taffy Wright	1115	3583	0.306	0.196
97	Matt Holliday	1837	5972	0.305	0.206
98	George Burns	2018	6573	0.304	0.211
99	Ed Morgan	879	2810	0.306	0.213
100	Home Run Baker	1838	5984	0.304	0.227

These players are borderline (like Hank Aaron at 94). We would guess that their career batting average is greater than .300, but we aren't as certain.

Let's say we chose to take the top 100 players for our Hall of Fame (thus, cut it off at Home Run Baker). What would we predict the false discovery rate to be? That is, what fraction of these 100 players would be falsely included?

```
top_players <- career_eb %>%
  arrange(PEP) %>%
  head(100)
```

Well, we know the PEP of each of these 100 players, which is the probability that that individual player is a false positive. This means we can just add up these probabilities to get the expected value (the average) of the total number of false positives.[3]

[3] If it's not clear why you can add up the probabilities like that, check out this explanation of linearity of expected value).

```
sum(top_players$PEP)
```

```
## [1] 5.46
```

This means that of these 100 players, we expect that about four and a half of them are false discoveries. Now, we don't know *which* four or five players we are mistaken about! (If we did, we could just kick them out of the hall). But we can make predictions about the players in aggregate. Here, we can see that taking the top 100 players would get pretty close to our goal of FDR = 5%.

Note that we're calculating the FDR as 4.43/100 = 4.43%. Thus, we're really computing the *mean* PEP: the average Posterior Error Probability.

```
mean(top_players$PEP)
```

```
## [1] 0.0546
```

We could have asked the same thing about the first 50 players, or the first 200. For each Hall of Fame cutoff we set, we could calculate a false discovery rate.

```
sorted_PEP <- career_eb %>%
  arrange(PEP)

mean(head(sorted_PEP$PEP, 50))
```

```
## [1] 0.00151
```

```
mean(head(sorted_PEP$PEP, 200))
```

```
## [1] 0.265
```

5.4 Q-values

We could experiment with many thresholds to get our desired FDR for each. But it's even easier just to compute them all thresholds at once, by computing the cumulative mean of all the (sorted) posterior error probabilities. This cumulative mean is called a **q-value**.[4]

[4] This approach uses the `cummean` function from dplyr, short for "cumulative mean". For example, `cummean(c(1, 2, 6, 10))` returns `(1, 1.5, 3, 4.75)`.

```
career_eb <- career_eb %>%
  arrange(PEP) %>%
  mutate(qvalue = cummean(PEP))
```

The term q-value was first defined by John Storey (Storey, 2002) as an analogue to the p-value for controlling FDRs in multiple testing. The q-value is convenient because we can say "to control the FDR at X%, collect only hypotheses where $q < X$".

```
hall_of_fame <- career_eb %>%
  filter(qvalue < .05)
```

Controlling at 5% ends up with 97 players in the Hall of Fame. If we wanted to be more careful about letting players in, we'd simply set a stricter q-value threshold, such as 1%.

```
strict_hall_of_fame <- career_eb %>%
  filter(qvalue < .01)
```

At that point we'd include only 64 players.

It's useful to look at how many players would be included at various q-value thresholds (Figure 5.4). This shows that you could include 200 players in the Hall of Fame, but at that point you'd expect that more than 25% of them would be incorrectly included. On the other side, you could create a hall of 50 players and be very confident that all of them have a batting probability of .300.

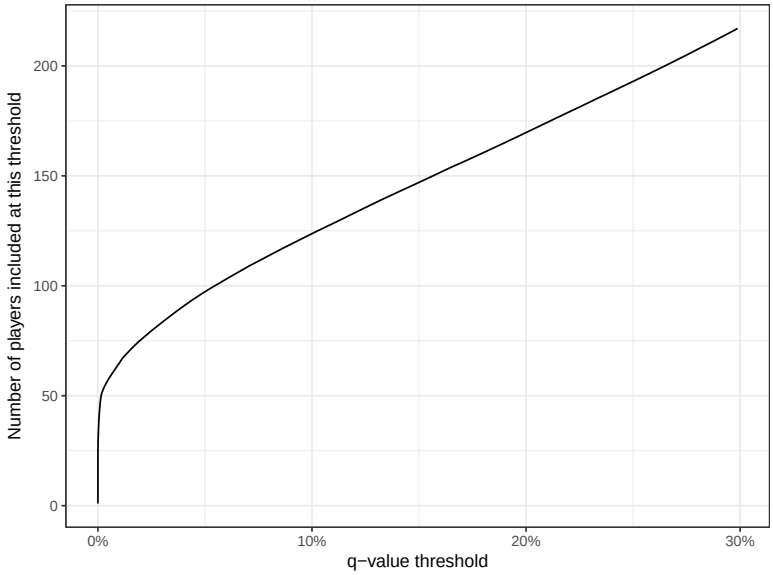

Figure 5.4: Comparison of the q-value threshold (for finding players with an average above .300) and the number of players that would be included at that threshold.

It's worth emphasizing the difference between measuring an individual's posterior error probability and the q-value, which is the false discovery rate of a group including that player. Hank Aaron has a PEP of 17%, but he can be included in the Hall of Fame while keeping the FDR below 5%. If this is surprising, imagine that you were instead trying to keep the average *height* of the group above 6'0". You would start by including all players taller than 6'0", but could also include some players who were 5'10" or 5'11" while preserving your average. Similarly, we simply need to keep the average PEP of the players below 5%.[5]

[5] For this reason, the PEP is sometimes called the *local* false discovery rate, which emphasizes both the connection and the distinction between PEP and FDR.

5.5 *Frequentists and Bayesians; meeting in the middle*

Before we move on, there's an interesting statistical implication to this analysis. So far in this book, we've been taking a Bayesian approach to our estimation and interpretation of batting averages. We haven't really used any frequentist statistics: in particular, we haven't seen a single p-value or null hypothesis. Now we've used our posterior distributions to compute q-values, and used that to control false discovery rate.

But it's relevant to note that the q-value was originally defined in terms of null hypothesis significance testing, particularly as a transformation of p-values under multiple testing (Storey and Tibshirani, 2003). By calculating, and then averaging, the posterior error probability, we've found another way to control FDR. This connection is explored in two great papers from my former advisor, (Storey, 2003) and (Käll et al., 2008).

There are some notable differences between our approach here and typical FDR control.[6], but this is a great example of the sometimes underappreciated technique of examining the frequentist properties of Bayesian approaches- and, conversely, understanding the Bayesian interpretations of frequentist goals.

[6] One major difference is that we aren't defining a null hypothesis (we aren't assuming any players have a batting average exactly *equal* to .300), but are instead trying to avoid what Andrew Gelman calls "Type S errors": getting the "sign" of an effect wrong.

6
Bayesian A/B testing

Who is a better batter: Mike Piazza or Hank Aaron?

Well, Mike Piazza has a slightly higher career batting average (2127 hits / 6911 at-bats = 0.308) than Hank Aaron (3771 hits / 12364 at-bats = 0.305). But can we say with confidence that his skill is *actually* higher, or is it possible he just got lucky a bit more often?

In this series of posts about an empirical Bayesian approach to batting statistics, we've been estimating batting averages by modeling them as a binomial distribution with a beta prior. But we've been looking at a single batter at a time. What if we want to compare *two* batters, find a probability that one is better than the other, and estimate *by how much*?

This is a topic rather relevant to my own work and to the data science field, because understanding the difference between two proportions is important in **A/B testing**. One of the most common examples of A/B testing is comparing clickthrough rates ("out of X impressions, there have been Y clicks")- which on the surface is similar to our batting average estimation problem ("out of X at-bats, there have been Y hits"").[1]

Here, we're going to look at an empirical Bayesian approach to comparing two batters. We'll define the problem in terms of the difference between each batter's posterior distribution, and look at four mathematical and computational strategies we can use to resolve this question. While we're focusing on baseball here, remember that similar strategies apply to A/B testing, and indeed to many Bayesian models.

[1] The differences between frequentist and Bayesian A/B testing is a topic I've blogged about in greater depth, particularly about the problem of early stopping

6.1 Setup

As usual, we start with code that sets up the variables analyzed in this chapter.

```
library(dplyr)
library(tidyr)
library(Lahman)

# Grab career batting average of non-pitchers
# (allow players that have pitched <= 3 games, like Ty Cobb)
pitchers <- Pitching %>%
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
  filter(gamesPitched > 3)

career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB)) %>%
  mutate(average = H / AB)

# Add player names
career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID")

# values estimated by maximum likelihood in Chapter 3
alpha0 <- 101.4
beta0 <- 287.3

# For each player, update the beta prior based on the evidence
# to get posterior parameters alpha1 and beta1
career_eb <- career %>%
  mutate(eb_estimate = (H + alpha0) / (AB + alpha0 + beta0)) %>%
  mutate(alpha1 = H + alpha0,
         beta1 = AB - H + beta0) %>%
  arrange(desc(eb_estimate))
```

6.2 Comparing posterior distributions

So let's take a look at the two batters in question, Hank Aaron and Mike Piazza.

```
# while we're at it, save them as separate objects too for later:
aaron <- career_eb %>% filter(name == "Hank Aaron")
piazza <- career_eb %>% filter(name == "Mike Piazza")
```

```
two_players <- bind_rows(aaron, piazza)

two_players
```

```
## # A tibble: 2 × 8
##    playerID        name     H    AB average
##       <chr>       <chr> <int> <int>   <dbl>
## 1 aaronha01  Hank Aaron  3771 12364   0.305
## 2 piazzmi01 Mike Piazza  2127  6911   0.308
## # ... with 3 more variables:
## #   eb_estimate <dbl>, alpha1 <dbl>,
## #   beta1 <dbl>
```

We see that Piazza has a slightly higher average (H/AB), *and* a higher shrunken empirical bayes estimate ($(H + \alpha_0)/(AB + \alpha_0 + \beta_0)$, where α_0 and β_0 are our priors).

But is Piazza's *true* probability of getting a hit higher? Or is the difference due to chance? To answer, let's consider, just as we did in Chapter 12.3 to find credible intervals, the actual posterior distributions. The posteriors give the range of plausible values for their "true" batting averages after we've taken the evidence (their batting record) into account. Recall that these posterior distributions are modeled as beta distributions with the parameters $\text{Beta}(\alpha_0 + H, \alpha_0 + \beta_0 + H + AB)$.

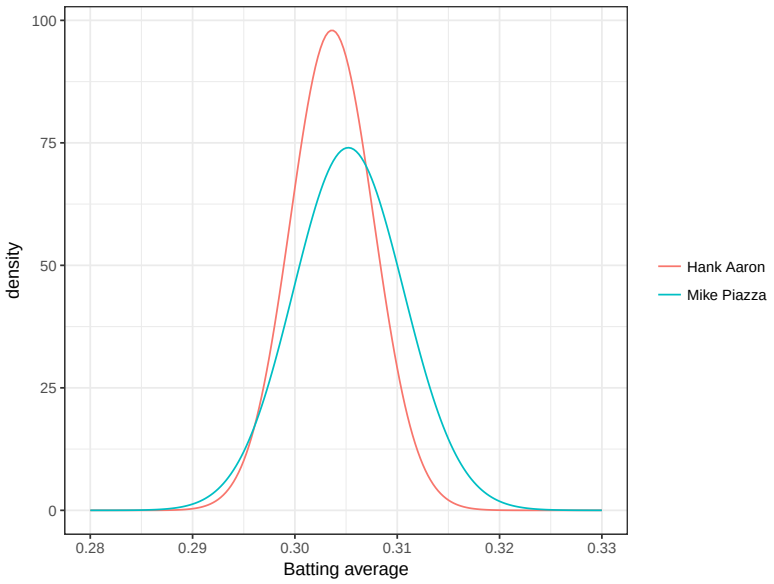

Figure 6.1: Posterior distributions for the batting average of Hank Aaron and Mike Piazza.

These posterior distributions (Figure 6.1) are therefore a probabilistic representation of our *uncertainty* in each estimate. When asking the probability Piazza is better, we're asking "if I picked a random draw from Piazza's distribution and a random draw from Aaron's, what's the probability Piazza is higher"?

Well, notice that those two distributions overlap *a lot*! There's enough uncertainty in each of those estimates that Aaron could easily be better than Piazza.

This changes if we throw another player in, retired Yankee Hideki Matsui (Figure 6.2). Hideki Matsui is a fine batter (above average for major league baseball), but not up to the level of Aaron and Piazza: notice that his posterior distribution of batting averages barely overlaps theirs. If we took a random draw from Matsui's distribution and from Piazza's, it's very unlikely that Matsui's would be higher.

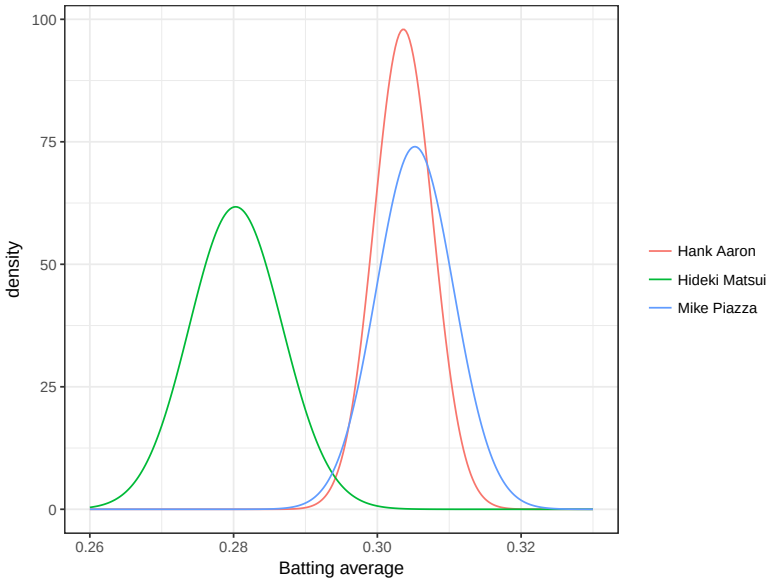

Figure 6.2: Posterior distributions for the batting average of Hank Aaron, Mike Piazza, and Hideki Matsui.

We may be interested in the probability that Piazza is better than Aaron within our model. We can already tell from the graph that it's greater than 50%, but probably not much greater. How could we quantify it?

We'd need to know the *probability one beta distribution is greater than another*. This question is not trivial to answer, and I'm going to illustrate four routes that are common lines of attack in a Bayesian problem:

- Simulation of posterior draws
- Numerical integration
- Closed-form solution
- Closed-form approximation

Which of these approaches you choose depends on your particular problem, as well as your computational constraints. In many cases an exact closed-form solution may not be known or even exist. In some cases (such as running machine learning in production) you

may be heavily constrained for time, while in others (such as drawing conclusions for a scientific paper) you care more about precision.

6.2.1 Simulation of posterior draws

If we don't want to do any math today (I'm sympathetic!), we could simply try simulation. We could use each player's α_1 and β_1 parameters, draw a million items from each of them using `rbeta`, and compare the results.

```
piazza_simulation <- rbeta(1e6, piazza$alpha1, piazza$beta1)
aaron_simulation <- rbeta(1e6, aaron$alpha1, aaron$beta1)

sim <- mean(piazza_simulation > aaron_simulation)
sim
```

```
## [1] 0.595
```

This gives about a 59.5% probability Piazza is better than Aaron! An answer like this is often good enough, depending on your need for precision and the computational efficiency. You could turn up or down the number of draws depending on how much you value speed vs precision.

Notice we didn't have to do any mathematical derivation or proofs. Even if we had a much more complicated model, the process for simulating from it would still have been pretty straightforward. This is one of the reasons Bayesian simulation approaches like MCMC have become popular: computational power has gotten very cheap, while doing math is as expensive as ever.

6.2.2 Integration

These two posteriors each have their own (independent) distribution, and together they form a *joint distribution*- that is, a density over particular pairs of x and y. That joint distribution could be imagined as a density cloud (Figure 6.3).

Here, we're asking what fraction of the joint probability density lies below that black line, where Piazza's average is greater than Aaron's. Notice that a bit more of the cloud's mass lies below than above: that's confirming the posterior probability that Piazza is better is about 60%.

The way to calculate this quantitatively is numerical integration, which is how Chris Stucchio approaches the problem in this post and this Python script. Here's a simple approach in R.

```
d <- .00002
limits <- seq(.29, .33, d)
```

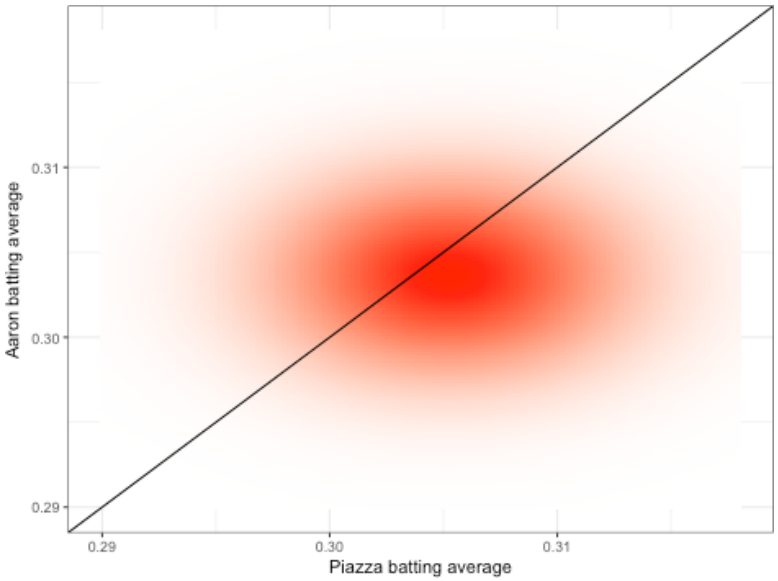

Figure 6.3: The joint probability density of Piazza's and Aaron's possible batting averages. Darker red means higher probability, and the black line represents the $x = y$ line.

```
sum(outer(limits, limits, function(x, y) {
  (x > y) *
    dbeta(x, piazza$alpha1, piazza$beta1) *
    dbeta(y, aaron$alpha1, aaron$beta1) *
    d ^ 2
}))
```

```
## [1] 0.593
```

Like simulation, this is a bit on the "brute force" side. (And unlike simulation, the approach becomes intractable in problems that have many dimensions, as opposed to the two dimensions here).

6.2.3 Closed-form solution

You don't need to be great at calculus to be a data scientist. But it's useful to know how to find people that *are* great at calculus. When it comes to A/B testing, the person to find is often Evan Miller.

This post lays out a closed-form solution Miller derived for the probability a draw from one beta distribution is greater than a draw from another:

$$p_A \sim \text{Beta}(\alpha_A, \beta_A)$$

$$p_B \sim \text{Beta}(\alpha_B, \beta_B)$$

$$\Pr(p_B > p_A) = \sum_{i=0}^{\alpha_B - 1} \frac{B(\alpha_A + i, \beta_A + \beta_B)}{(\beta_B + i) B(1 + i, \beta_B) B(\alpha_A, \beta_A)}$$

(Where B is the beta function). If you'd like an intuition behind this formula... well, you're on your own. But it's pretty straightforward to implement in R.[2]

[2] I'm borrowing notation from the Chris Stucchio post and calling this function h.

```
h <- function(alpha_a, beta_a,
              alpha_b, beta_b) {
  j <- seq.int(0, round(alpha_b) - 1)
  log_vals <- (lbeta(alpha_a + j, beta_a + beta_b) - log(beta_b + j) -
               lbeta(1 + j, beta_b) - lbeta(alpha_a, beta_a))
  1 - sum(exp(log_vals))
}

h(piazza$alpha1, piazza$beta1,
  aaron$alpha1, aaron$beta1)
```

```
## [1] 0.596
```

Having an exact solution is pretty handy![3] So why did we even look at simulation/integration approaches? Well, the downsides are:

[3] Note that this solution is exact only for integer values of α_b: we're rounding it here, which is a trivial difference in most of our examples but may matter in others.

- *Not every problem has a solution like this.* And even if it does, we may not know it. That's why it's worth knowing how to run a simulation. (If nothing else, they let us check our math!)
- *This solution is slow for large α_B, and not straightforward to vectorize*: notice that term that iterates from 0 to $\alpha_B - 1$. If we run A/B tests with thousands of clicks, this step is going to constrain us (though it's still usually faster than simulation or integration).

6.2.4 Closed-form approximation

As this report points out, there's a much faster approximation we can use. Notice that when α and β are both fairly large, the beta starts looking a lot like a normal distribution, so much so that it can be closely approximated. In fact, if you draw the normal approximation to the two players we've been considering, they are *visually indistinguishable* (Figure 6.4).

The probability one normal variable is greater than another is *very easy to calculate*- much easier than the beta!

```
h_approx <- function(alpha_a, beta_a, alpha_b, beta_b) {
  u1 <- alpha_a / (alpha_a + beta_a)
  u2 <- alpha_b / (alpha_b + beta_b)
  var1 <- (alpha_a * beta_a) /
    ((alpha_a + beta_a) ^ 2 * (alpha_a + beta_a + 1))
  var2 <- (alpha_b * beta_b) /
    ((alpha_b + beta_b) ^ 2 * (alpha_b + beta_b + 1))
```

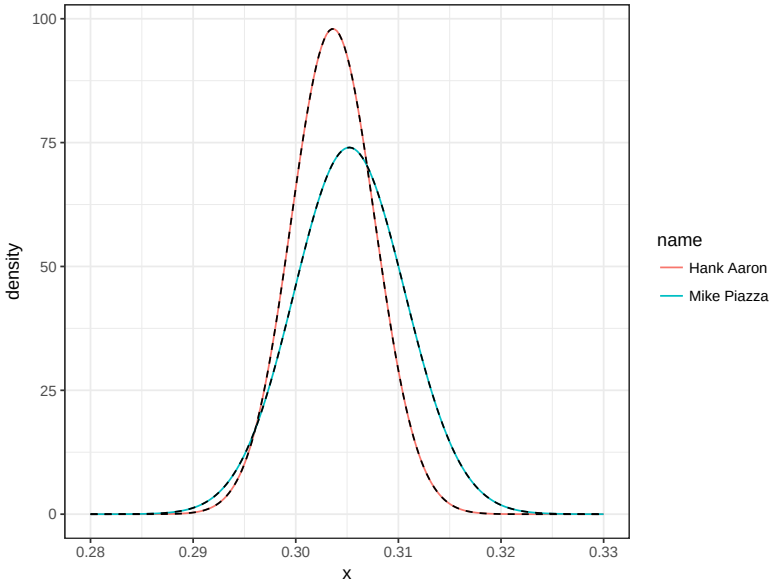

Figure 6.4: The posterior beta distribution of batting averages for two players, shown alongside the normal approximation to each as a dashed line.

```
  pnorm(0, u2 - u1, sqrt(var1 + var2))
}

h_approx(piazza$alpha1, piazza$beta1, aaron$alpha1, aaron$beta1)

## [1] 0.595
```

This calculation is very fast, and (in R terms) it's *vectorizable*.

The disadvantage is that for low α or low β, the normal approximation to the beta is going to fit rather poorly. While the simulation and integration approaches were inexact, this one will be *systematically biased*: in some cases it will always give too high an answer, and in some cases too low. But when we have priors $\alpha_0 = 101.4$ and $\beta_0 = 287.3$, as we do here, our parameters are never going to be low, so we're safe using it.

6.3 Confidence and credible intervals

In classical (frequentist) statistics, you may have seen this kind of "compare two proportions" problem before, perhaps laid out as a "contingency table".

Player	Hits	Misses
Hank Aaron	3771	8593
Mike Piazza	2127	4784

One of the most common ways to approach these contingency table problems is with Pearson's chi-squared test, implemented in R as `prop.test`.

```
prop.test(two_players$H, two_players$AB)
```

```
##
##  2-sample test for equality of
##  proportions with continuity correction
##
## data:  two_players$H out of two_players$AB
## X-squared = 0.1, df = 1, p-value = 0.7
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.0165  0.0109
## sample estimates:
## prop 1 prop 2
##  0.305  0.308
```

We see a non-significant p-value of .70, indicating the test couldn't find a difference.[4] Something else useful that `prop.test` gives you is a confidence interval for the difference between the two players. We learned in Chapter 12.3 about using **credible intervals** to represent the uncertainty in each player's average. Now we'll use empirical Bayes to compute the credible interval about the *difference* in these two players.

[4] We won't talk about p-values here (we talked a little about ways to translate between p-values and posterior probabilities in Chapter 11.4), but we can agree it would have been strange if the p-value were significant, given that the posterior distributions overlapped so much.

We could do this with simulation or integration, but we'll use our normal approximation approach since it's the most efficient (we'll also compute our posterior probability while we're at it).

```
credible_interval_approx <- function(a, b, c, d) {
  u1 <- a / (a + b)
  u2 <- c / (c + d)
  var1 <- a * b / ((a + b) ^ 2 * (a + b + 1))
  var2 <- c * d / ((c + d) ^ 2 * (c + d + 1))

  mu_diff <- u2 - u1
  sd_diff <- sqrt(var1 + var2)

  data_frame(posterior = pnorm(0, mu_diff, sd_diff),
             estimate = mu_diff,
             conf.low = qnorm(.025, mu_diff, sd_diff),
             conf.high = qnorm(.975, mu_diff, sd_diff))
}

credible_interval_approx(piazza$alpha1, piazza$beta1,
                         aaron$alpha1, aaron$beta1)
```

```
## # A tibble: 1 × 4
##   posterior estimate conf.low conf.high
```

```
##        <dbl>     <dbl>     <dbl>      <dbl>
## 1      0.595 -0.00162   -0.0149     0.0116
```

It's not particularly exciting for this Piazza/Aaron comparison (notice it's very close to the confidence interval we calculated with `prop.test`). So let's select 20 random players, and compare each of them to Mike Piazza: how many players can we say are better than Piazza? We'll also calculate the confidence interval using `prop.test`, and compare them (Figure 6.5).

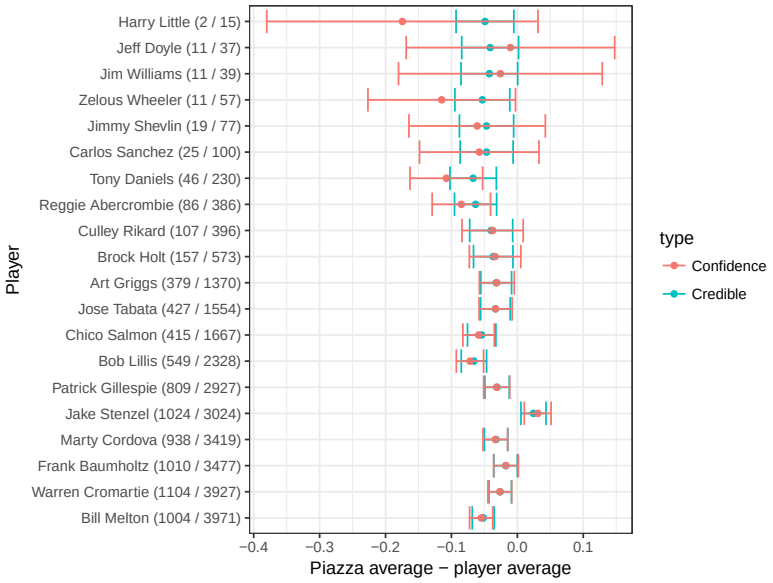

Figure 6.5: Confidence and credible intervals for comparing the batting of 20 randomly selected players to Mike Piazza. Players are sorted in increasing order of the number of at-bats.

Notice the same pattern we saw in Chapter 4.4. When we don't have a lot of information about a player, their credible interval ends up smaller than their confidence interval, because we're able to use the prior to adjust our expectations (Harry Little's batting average may be lower than Mike Piazza's, but we're confident it's not .3 lower). When we do have a lot of information, the credible intervals and confidence intervals converge almost perfectly.[5]

Thus, we can think of empirical Bayes A/B credible intervals as being a way to "shrink" frequentist confidence intervals, by sharing power across players.

[5] This can be derived mathematically, based on the fact that `prop.test`'s confidence interval is in fact very similar to our normal approximation along with an uninformative prior and a small continuity correction, but it's left as an exercise for the reader.

Part III

Extending the Model

7

Beta binomial regression

In this book we've been using the empirical Bayes method to estimate batting averages of baseball players. Empirical Bayes is useful in these examples because when we don't have a lot of information about a batter, they're "shrunken" towards the average across all players, as a natural consequence of the beta prior.

But there's a complication that we haven't yet approached. **When players are better, they are given more chances to bat!**[1]. That means there's a relationship between the number of at-bats (AB) and the true batting average. For reasons explained, this makes our estimates systematically inaccurate.

In this chapter, we'll adjust our model to a new one where each batter has his own prior, using a method called **beta-binomial regression**. We show that this new model lets us adjust for the confounding factor while still relying on the empirical Bayes philosophy. We also note that this gives us a general framework for allowing a prior to depend on known information, which will become important in Chapter 11.3.

[1] Hat tip to Hadley Wickham for pointing this complication out to me.

7.1 Setup

As usual, we start with code that sets up the variables analyzed in this chapter.

```
library(dplyr)
library(tidyr)
library(Lahman)
library(ggplot2)
theme_set(theme_bw())

# grab career batting average of non-pitchers
# (allow players that have pitched <= 3 games, like Ty Cobb)
pitchers <- Pitching %>%
```

```
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
  filter(gamesPitched > 3)

career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB)) %>%
  mutate(average = H / AB)

# add player names
career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID")

# values estimated by maximum likelihood in Chapter 3
alpha0 <- 101.4
beta0 <- 287.3
prior_mu <- alpha0 / (alpha0 + beta0)

# for each player, update the beta prior based on the evidence
# to get posterior parameters alpha1 and beta1
career_eb <- career %>%
  mutate(eb_estimate = (H + alpha0) / (AB + alpha0 + beta0)) %>%
  mutate(alpha1 = H + alpha0,
         beta1 = AB - H + beta0) %>%
  arrange(desc(eb_estimate))
```

Recall that the `eb_estimate` column gives us estimates about each player's batting average, estimated from a combination of each player's record with the beta prior parameters estimated from everyone (α_0, β_0). For example, a player with only a single at-bat and a single hit ($H = 1; AB = 1; H/AB = 1$) will have an empirical Bayes estimate of

$$(H + \alpha_0)/(AB + \alpha_0 + \beta_0) = (1 + 101.4)/(1 + 101.4 + 287.3) = 0.263$$

Now, here's the complication. Let's compare at-bats (on a log scale) to the raw batting average (Figure 7.1). We notice that batters with low ABs have more variance in our estimates- that's a familiar pattern, because we have less information about them and the raw estimate is noisier.

But notice a second trend: as the number of at-bats increases, the batting average *also* increases. Unlike the variance, this is *not* an

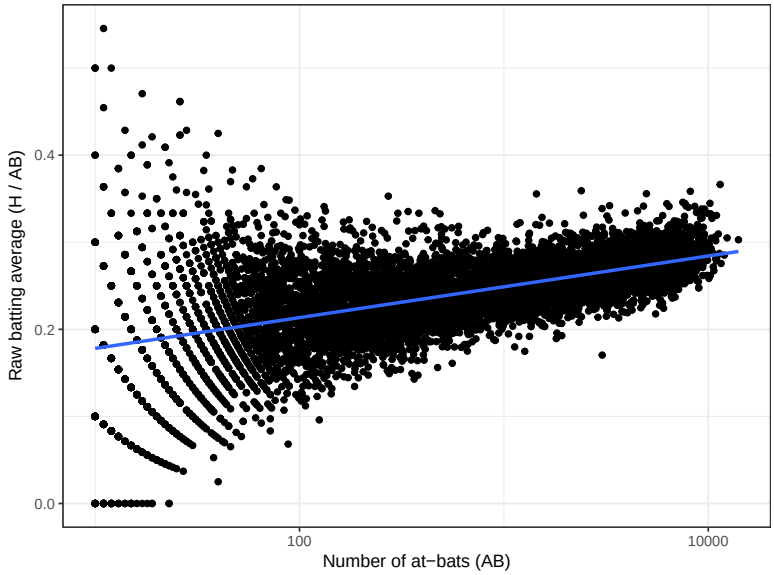

Figure 7.1: Relationship between the number of at-bats (AB) and the raw batting average (H / AB) across all players with at least 10 at-bats.

artifact of our measurement: it's a result of the choices of baseball managers! Better batters get played more: they're more likely to be in the starting lineup and to spend more years playing professionally.

Now, there are many other factors that are correlated with a player's batting average (year, position, team, etc). But this one is particularly important, because it confounds our ability to perform empirical Bayes estimation.

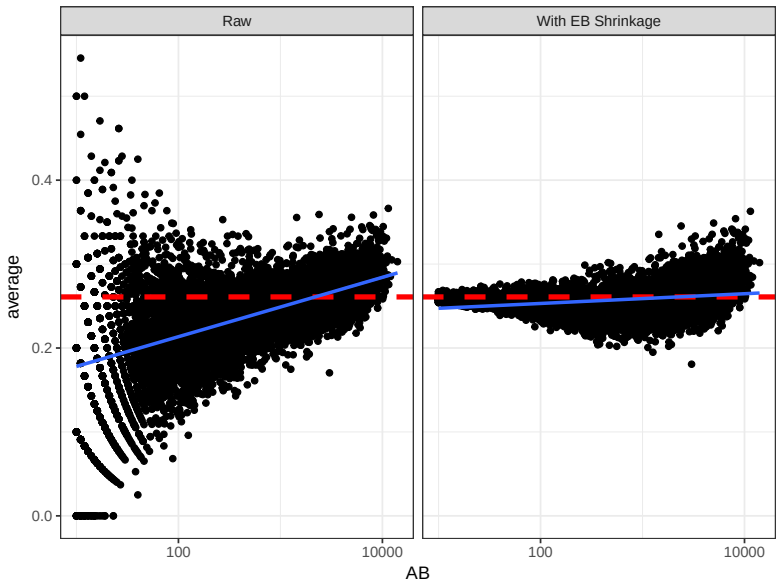

Figure 7.2: Scatter plot of the relationship AB has with raw batting average (left) and with empirical Bayes shrunken estimates (right). The prior mean .261 is shown as a horizontal dashed red line, -fit lines are shown in blue.

Figure 7.2 shows how empirical Bayes shrinkage affects the estimates of player batting averages. That horizontal red line shows the

prior mean that we're "shrinking" towards ($\frac{\alpha_0}{\alpha_0+\beta_0} = 0.261$). Notice that it is too high for the low-AB players. For example, the median batting average for players with 5-20 at-bats is 0.167, and they get shrunk *way* towards the overall average! The high-AB crowd basically stays where they are, because each batter has a lot of evidence.

So since low-AB batters are getting overestimated, and high-AB batters are staying where they are, we're working with a biased estimate that is systematically *overestimating* batter ability. If we were working for a baseball manager (like in Moneyball), that's the kind of mistake we could get fired for!

7.2 Accounting for AB in the model

How can we fix our model? We'll need to have AB somehow influence our priors, particularly affecting the mean batting average. In particular, we want the typical batting average to be linearly affected by log(AB).

First we should write out what our current model is, in the form of a **generative process**, in terms of how each of our variables is generated from particular distributions. Defining p_i to be the true probability of hitting for batter i (that is, the "true average" we're trying to estimate), we're assuming

$$p_i \sim \text{Beta}(\alpha_0, \beta_0)$$

$$H_i \sim \text{Binom}(\text{AB}_i, p_i)$$

(We're letting the totals AB_i be fixed and known per player). We made up this model in one of the first posts in this series and have been using it since.

I'll point out that there's another way to write the p_i calculation, by **re-parameterizing** the beta distribution. Instead of parameters α_0 and β_0, let's write it in terms of μ_0 and σ_0:

$$p_i \sim \text{Beta}(\mu_0/\sigma_0, (1-\mu_0)/\sigma_0)$$

Here, μ_0 represents the mean batting average, while σ represents how spread out the distribution is (note that $\sigma = \frac{1}{\alpha+\beta}$). When σ is high, the beta distribution is very wide (a less informative prior), and when σ is low, it's narrow (a more informative prior). Way back in my first post about the beta distribution, this is basically how I chose parameters: I wanted $\mu = .27$, and then I chose a σ that would give the desired distribution that mostly lay between .210 and .350, our expected range of batting averages.

Now that we've written our model in terms of μ and σ, it becomes easier to see how a model could take AB into consideration. We simply define μ so that it includes log(AB) as a linear term[2]:

$$\mu_i = \mu_0 + \mu_{\text{AB}} \cdot \log(\text{AB})$$

$$\alpha_{0,i} = \mu_i / \sigma_0$$

$$\beta_{0,i} = (1 - \mu_i) / \sigma_0$$

Then we define the batting average p_i and the observed H_i just like before:

$$p_i \sim \text{Beta}(\alpha_{0,i}, \beta_{0,i})$$

$$H_i \sim \text{Binom}(\text{AB}_i, p_i)$$

This particular model is called **beta-binomial regression**. We already had each player represented with a binomial whose parameter was drawn from a beta, but now we're allowing the expected value of the beta to be influenced.

[2] If you have some experience with regressions, you might notice a problem: μ can theoretically go below 0 or above 1, which is impossible for a β distribution. Thus in a real model we would use a "link function", such as the logistic function, to keep μ between 0 and 1. I used a linear model (and `mu.link = "identity"` in the `gamlss` call) to make the math in this introduction simpler, and because for this particular data it leads to almost exactly the same answer (try it).

7.3 Step 1: Fit the model across all players

Going back to the basics of empirical Bayes from Chapter 3, our first step is to fit these prior parameters: μ_0, μ_{AB}, σ_0. When doing so, it's ok to momentarily "forget" we're Bayesians- we picked our α_0 and β_0 using maximum likelihood, so it's OK to fit these using a maximum likelihood approach as well. You can use the gamlss (Stasinopoulos and Rigby, 2016) package for fitting beta-binomial regression using maximum likelihood.

```
library(gamlss)

fit <- gamlss(cbind(H, AB - H) ~ log(AB),
              data = career_eb,
              family = BB(mu.link = "identity"))
```

We can pull out the coefficients using `tidy()` from my broom package.[3]

```
library(broom)

td <- tidy(fit)
td
```

[3] The broom package provides methods for "tidying" model objects into data frames. See `?gamlss_tidiers` for documentation on this particular tidier.

```
##   parameter        term estimate std.error
## 1        mu (Intercept)   0.1426  0.001577
## 2        mu     log(AB)   0.0153  0.000215
## 3     sigma (Intercept)  -6.2935  0.023052
##   statistic p.value
## 1      90.4       0
## 2      71.2       0
## 3    -273.0       0
```

This gives us our three parameters: $\mu_0 = 0.143$, $\mu_{\text{AB}} = 0.015$, and (since `sigma` has a log-link) $\sigma_0 = \exp(-6.294) = 0.002$.

This means that our new prior beta distribution for a player *depends on* the value of AB. For example, here are our prior distributions for several values of AB.

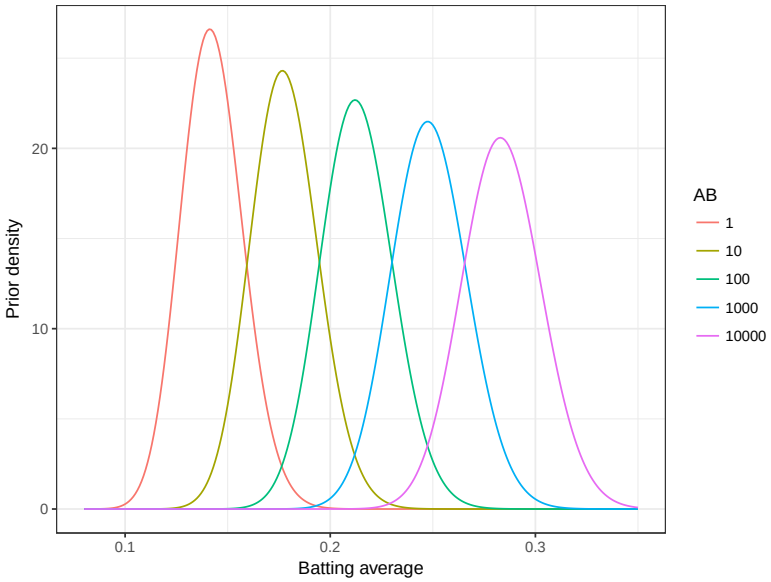

Figure 7.3: The density of the prior distribution for a player with particular numbers of at-bats.

Notice that there is still uncertainty in our prior- a player with 10,000 at-bats could have a batting average ranging from about .22 to .35. But the range of that uncertainty changes greatly depending on the number of at-bats- any player with AB = 10,000 is almost certainly better than one with AB = 10.

7.4 Step 2: Estimate each player's average using this prior

Now that we've fit our overall model, we repeat our second step of the empirical Bayes method. Instead of using a single α_0 and β_0 values as the prior, we choose the prior for each player based on their AB. We then update using their H and AB just like before.

Here, all we need to calculate are the `mu` (that is, $\mu = \mu_0 + \mu_{\log(\text{AB})}$)

and `sigma` (σ) parameters for each person. (Here, `sigma` will be the same for everyone, but that may not be true in more complex models). This can be done using the `fitted` method on the gamlss object.

```
mu <- fitted(fit, parameter = "mu")
sigma <- fitted(fit, parameter = "sigma")

head(mu)
```

```
##     1     2     3     4     5     6
## 0.286 0.281 0.273 0.262 0.279 0.284
```

```
head(sigma)
```

```
##       1       2       3       4       5
## 0.00185 0.00185 0.00185 0.00185 0.00185
##       6
## 0.00185
```

Now we can calculate α_0 and β_0 parameters for each player, according to $\alpha_{0,i} = \mu_i/\sigma_0$ and $\beta_{0,i} = (1 - \mu_i)/\sigma_0$. From that, we can update based on H and AB to calculate new $\alpha_{1,i}$ and $\beta_{1,i}$ for each player.

```
career_eb_wAB <- career_eb %>%
  dplyr::select(name, H, AB, original_eb = eb_estimate) %>%
  mutate(mu = mu,
         alpha0 = mu / sigma,
         beta0 = (1 - mu) / sigma,
         alpha1 = alpha0 + H,
         beta1 = beta0 + AB - H,
         new_eb = alpha1 / (alpha1 + beta1))
```

We visualize how this changes our estimates in Figure 7.4. Notice that relative to the previous empirical Bayes estimate (shown in Figure 3.3), this one gives lower estimates for batters with low AB and about the same for high-AB batters. This fits with our earlier description- we've been systematically over-estimating batting averages.

We can also revisit the AB/estimate relationship examined in Figure 7.1, and see whether new method solves the problem of shrinking low-AB batters to be too high (Figure 7.5).

Notice that our original estimate ("EB Estimate") used to shrink batters towards the overall average, but the new estimate ("EB w/ Regression") now shrinks them towards the overall *trend* fit by beta binomial regression, represented by that red slope.[4]

Don't forget that this change in the posteriors won't just affect shrunken estimates. It will affect all the ways we've used posterior

[4] If you work in in my old field of gene expression, you may be interested to know that empirical Bayes shrinkage towards a trend is exactly what some differential expression packages such as edgeR do with per-gene dispersion estimates. It's a powerful concept that allows a balance between individual observations and overall expectations.

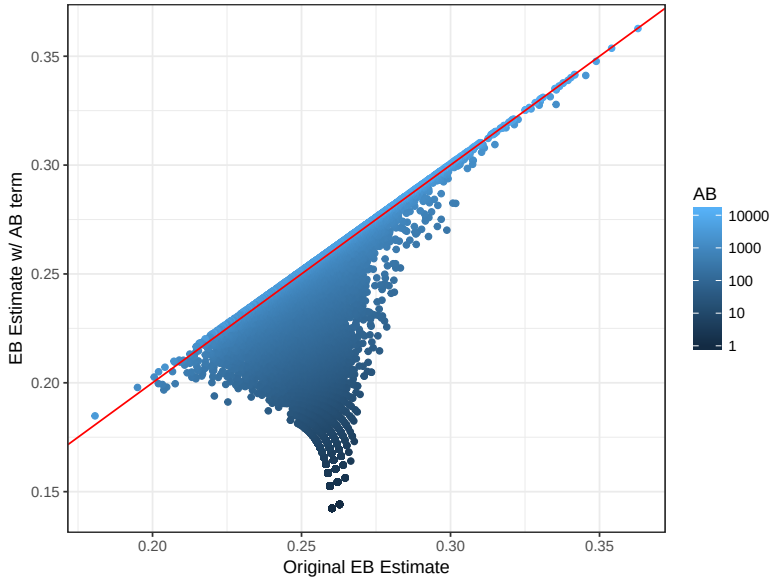

Figure 7.4: The relationship between the original empirical Bayes shrunken estimates and the values under the beta-binomial regression model.

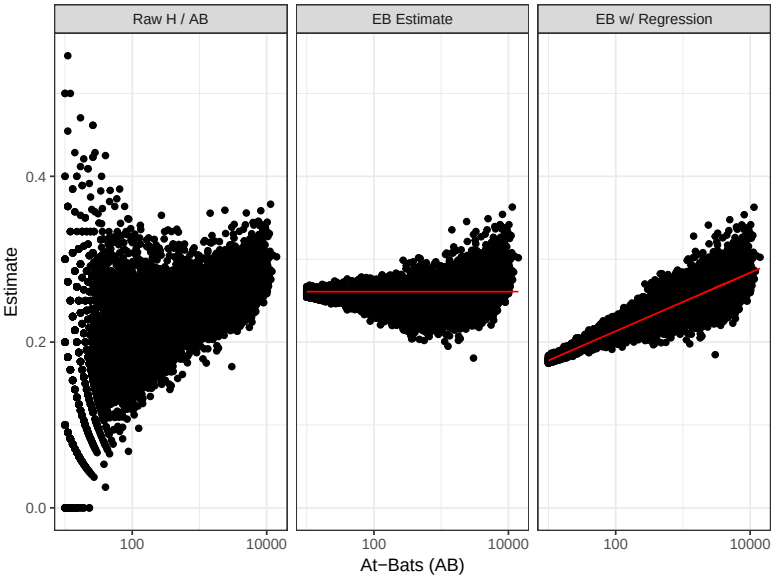

Figure 7.5: The relationship between AB and the estimate for three methods: raw batting average, shrunken batting average, and averages shrunk towards a relationship found through regression.

distributions in previous chapters: credible intervals, posterior error probabilities, and A/B comparisons. Improving the model by taking AB into account can help all these results more accurately reflect reality.

8

Empirical Bayesian hierarchical modeling

Suppose you were a scout hiring a new baseball player, and were choosing between two that have had 100 at-bats each:

- A left-handed batter who has hit **30 hits / 100 at-bats**
- A right-handed batter who has hit **30 hits / 100 at-bats**

Who would you guess was the better batter?

This seems like a silly question: they both have the same exact batting record. But what if I told you that historically, left-handed batters are slightly better hitters than right-handed? How could you incorporate that evidence?

In Chapter 7, we used the method of beta-binomial regression to incorporate information (specifically the number of at-bats a player had) into a per-player prior distribution. We did this to correct a bias of the algorithm, but we could do a lot more with this method: in particular, we can include other factors that might change our prior expectations of a player.

These are each particular applications of Bayesian hierarchical modeling, where the priors for each player are not fixed, but rather depend on other latent variables. In our empirical Bayesian approach to hierarchical modeling, we'll estimate this prior using beta binomial regression, and then apply it to each batter.[1] This change in the model can influence our credible intervals, A/B testing, and the other methods we've explored.

[1] This strategy is useful in many applications beyond baseball- for example, if we were analyzing ad clickthrough rates on a website, we might notice that different countries have different clickthrough rates, and therefore fit different priors for each.

8.1 Setup

As usual, we start with code that sets up the variables analyzed in this chapter.

```
library(gamlss)
library(dplyr)
library(tidyr)
```

```
library(Lahman)
library(ggplot2)
theme_set(theme_bw())

# Grab career batting average of non-pitchers
# (allow players that have pitched <= 3 games, like Ty Cobb)
pitchers <- Pitching %>%
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
  filter(gamesPitched > 3)

# we're keeping some extra information for later in the post:
# a "bats" column and a "year" column
career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB), year = mean(yearID)) %>%
  mutate(average = H / AB)

# Add player names
career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast, bats) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID")
```

Based on our model in Chapter 7, we perform beta binomial regression using the gamlss package. This fits a model that allows the mean batting average μ to depend on the number of at-bats a player has had.

```
library(gamlss)

fit <- gamlss(cbind(H, AB - H) ~ log(AB),
              data = dplyr::select(career, -bats),
              family = BB(mu.link = "identity"))
```

The prior α_0 and β_0 can then be computed for each player based on μ and a dispersion parameter σ.

```
career_eb <- career %>%
  mutate(mu = fitted(fit, "mu"),
         sigma = fitted(fit, "sigma"),
         alpha0 = mu / sigma,
         beta0 = (1 - mu) / sigma,
```

```
    alpha1 = alpha0 + H,
    beta1 = beta0 + AB - H,
    estimate = alpha1 / (alpha1 + beta1))
```

Now we've corrected for one confounding factor, AB. One important aspect of this prediction is that it won't be useful when we've just hired a "rookie" player, and we're wondering what his batting average will be. This observed variable AB is based on a player's *entire career*, such that a low number is evidence that a player didn't have much of a chance to bat.[2]

But there's some information we *can* use even at the start of a player's career. Part of the philosophy of the Bayesian approach is to bring our *prior expectations* in mathematically. Let's try doing that with some factors that influence batting success.

[2] If we wanted to make a prediction for a rookie, we'd have to consider the distribution of possible AB's the player could end up with and integrate over that, which is beyond the scope of this book.

8.2 Right- and left- handed batters

It's well known in sabermetrics that left-handed batters tend to bat slightly better.[3] The Lahman dataset provides that information in the `bats` column.

[3] In fact, the general belief is that left-handed batters have an advantage *against right-handed pitchers*, but since most pitchers historically have been right-handed this evens out to an advantage.

```
career %>%
  count(bats)
```

```
## # A tibble: 4 × 2
##     bats     n
##   <fctr> <int>
## 1      B   815
## 2      L  2864
## 3      R  5869
## 4     NA   840
```

These letters represent "Both" (switch hitters), "Left", and "Right", respectively. One interesting feature is that while the ratio of righties to lefties is about 9-to-1 in the general population, in professional baseball it is only 2-to-1. Managers like to hire left-handed batters- in itself, this is some evidence of a left-handed advantage! We also see that there are a number of batters (mostly from earlier in the game's history) that we don't have handedness information for. We'll filter them out of this analysis.

Incorporating this as a predictor is as simple as adding `bats` to the formula in the `gamlss` call (our beta-binomial regression).

```
# relevel to set right-handed batters as the baseline
career2 <- career %>%
  filter(!is.na(bats)) %>%
```

```
  mutate(bats = relevel(bats, "R"))

fit2 <- gamlss(cbind(H, AB - H) ~ log(AB) + bats,
               data = career2,
               family = BB(mu.link = "identity"))
```

We can then look at the coefficients using the `tidy()` function from broom.

```
library(broom)
tidy(fit2)
```

```
##   parameter        term estimate std.error
## 1        mu (Intercept)  0.14077  0.001626
## 2        mu     log(AB)  0.01516  0.000219
## 3        mu       batsB -0.00108  0.000990
## 4        mu       batsL  0.01011  0.000634
## 5     sigma (Intercept) -6.37789  0.023299
##   statistic  p.value
## 1      86.6 0.00e+00
## 2      69.1 0.00e+00
## 3      -1.1 2.73e-01
## 4      15.9 1.75e-56
## 5    -273.7 0.00e+00
```

According to our beta-binomial regression, there is indeed a statistically significant advantage to being left-handed, with lefties getting a hit about 1% more often. This may seem like a small effect, but over the course of multiple games it could certainly make a difference. In contrast, there's apparently no detectable advantage to being able to bat with both hands.

For our empirical Bayes estimation, this means every combination of handedness and AB now has its own prior, as shown in Figure 8.1. Left-handed batters, and those who had more hits in their career, would have generally higher expectations.

Note that this prior can still easily be overcome by enough evidence. For example, consider our hypothetical pair of battters from the introduction, where each has a 30% success rate, but where one is left-handed and one right-handed. If the batters had few at-bats (such as 10 or 100), we'd guess that the left-handed batter was better, but the posterior for the two will converge as AB increases to 1000 or 10,000 (Figure 8.2).

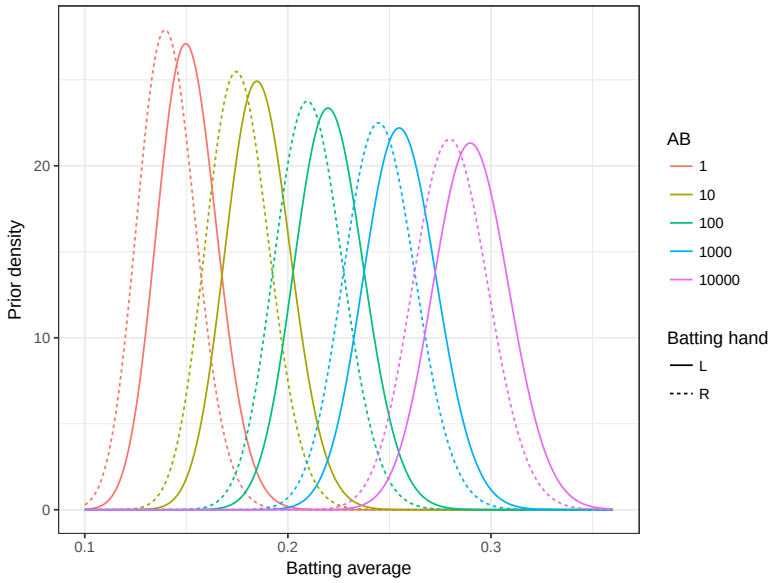

Figure 8.1: The prior distribution according to the hierarchical model for players with particular combinations of AB and handedness.

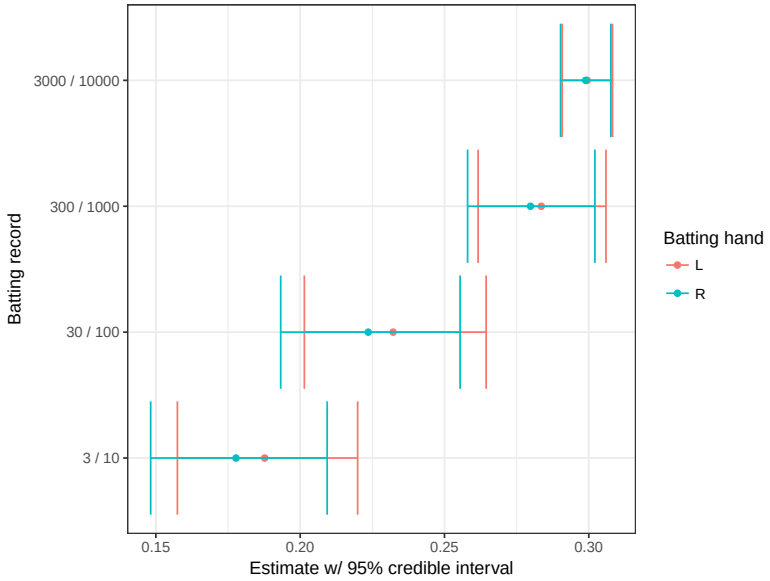

Figure 8.2: Empirical Bayes estimates and 95% credible intervals for two hypothetical batters with a 30% success rate, one left-handed and one right-handed.

8.3 Over time

One of the most dramatic pieces of information we've "swept under the rug" in our analysis is the time period when each player was active. It's absurd to expect that players in the 1880s would have the same ranges of batting averages as players today, and we should take that into account in our estimates. Each player in the dataset therefore includes a `year` column representing the average of the years that player was an active batter.

name	bats	H	AB	year	average
Hank Aaron	R	3771	12364	1965	0.305
Tommie Aaron	R	216	944	1967	0.229
Andy Abad	L	2	21	2003	0.095
John Abadie	R	11	49	1875	0.224
Ed Abbaticchio	R	772	3044	1905	0.254
Charlie Abbey	L	492	1751	1895	0.281

Could we simply fit a linear model with respect to year, such as `~ log10(AB) + bats + year`? Well, before we fit a model we should always look at a graph. A boxplot comparing batting averages across decades is a good start (Figure 8.3).

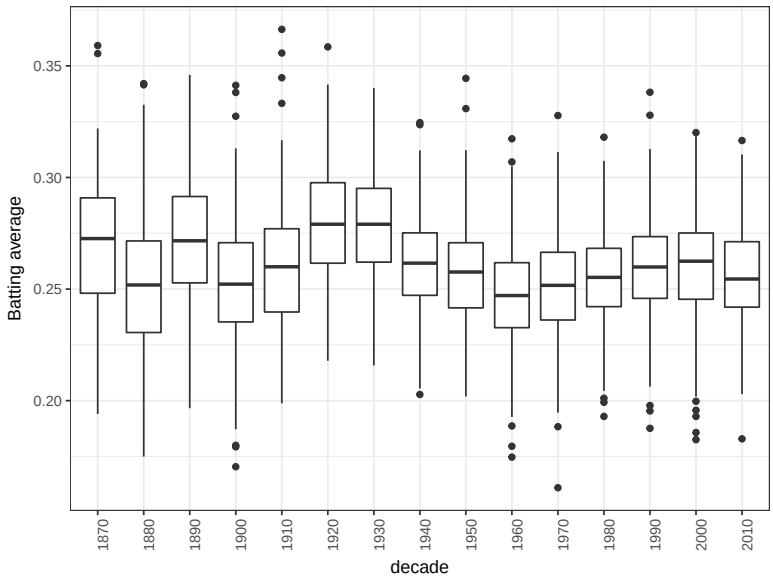

Figure 8.3: Boxplot of batting averages within each decade. To reduce the effect of noise, only players with more than 500 at-bats are included.

Well, there's certainly a trend over time, but there's nothing linear about it: batting averages have both risen and fallen across time. If you're interested in baseball history and not just Bayesian statistics, you may notice that this graph marks the "power struggle" between offense and defense.

- The rise in the 1920s and 1930s marks the end of the dead-ball era,

where hitting, especially home runs, became a more important part of the game
- The batting average "cools off" as pitchers adjust their technique, especially when the range of the strike zone was increased in 1961
- Batting average rose again in the 1970s thanks to the designated hitter rule, where pitchers (in one of the two main leagues) were no longer required to bat
- It looks like batting averages may again be drifting downward

In any case, we certainly can't fit a simple linear trend here. To capture the nonlinear trend, we could instead fit a natural cubic spline using the ns function:[4]

```
library(splines)

fit3 <- gamlss(cbind(H, AB - H) ~ 0 + ns(year, df = 5)
                                    + bats + log(AB),
               data = career2,
               family = BB(mu.link = "identity"))
```

We now have a prior for each year, handedness, and number of at-bats. For example, Figure 8.4 the prior distribution for each year and handedness for a hypothetical player with AB = 1000.

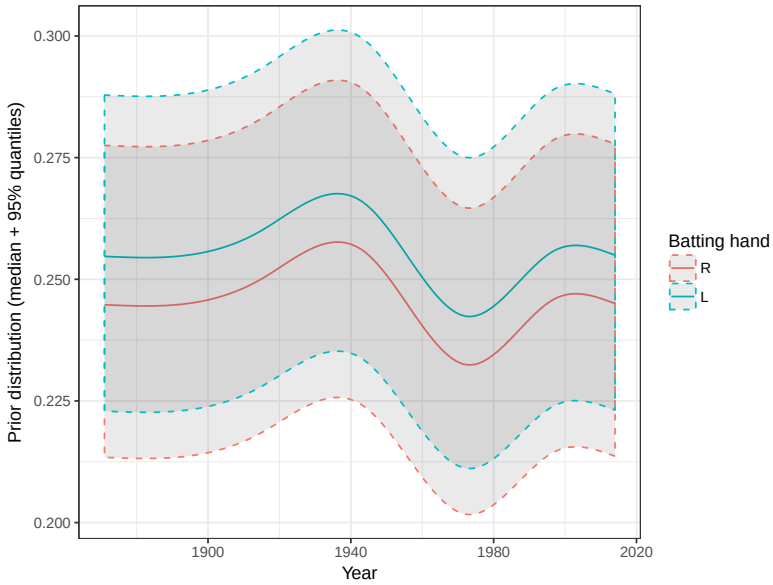

Figure 8.4: The prior distribution that would be used for a left- or right-handed player at a particular point in time. Shown are the mean and the 95% intervals for each prior.

Note that those intervals don't represent uncertainty about our trend, but rather 95% range in prior batting averages. Each combination of year and left/right handedness is a beta distribution, of which we're seeing just one cross-section.

[4] Why 5 degrees of freedom? Not very scientific- I just tried a few and picked one that roughly captured the shapes we saw in the boxplots. If you have too few degrees of freedom you can't capture the complex trend we're seeing here, but if you have too many you'll overfit to noise in your data. If you're not familiar with splines, what's important is that even in a linear model, we can include nonlinear trends.

8.3.1 Interaction terms

One of the implicit assumptions of the above model is that the effect of left-handedness hasn't changed over time. But this may not be true! We can change the formula to allow an interaction term `ns(year, 5) * bats`, which lets the effect of handedness change over time.

```
fit4 <- gamlss(cbind(H, AB - H) ~ 0 + ns(year, 5) * bats
                                    + log(AB),
               data = career2,
               family = BB(mu.link = "identity"))
```

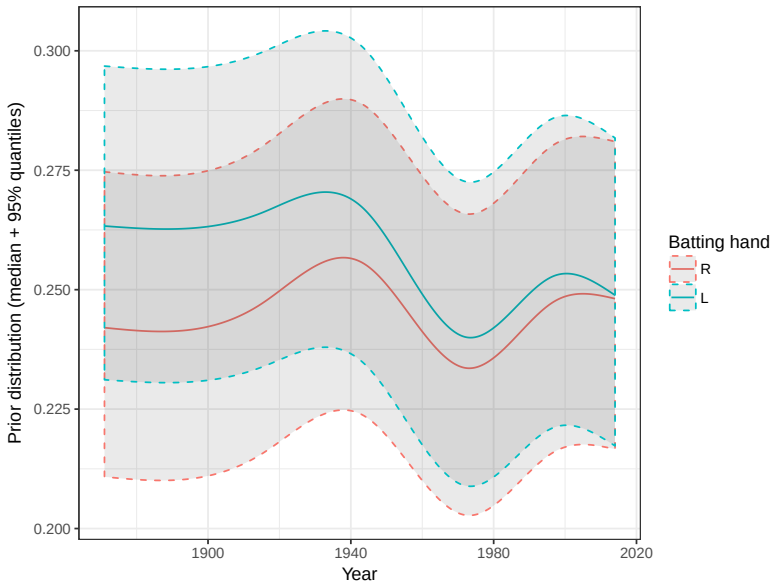

Figure 8.5: The prior distribution that would be used for a left- or right-handed player at a particular point in time, allowing for an interaction term between time and handedness.

These new priors are shown in Figure 8.5. Interesting- we can now see that *the gap between left-handed and right-handed batters has been closing since the start of the game,* such that today the gap has basically completely disappeared. This suggests that managers and coaches may have learned how to deal with left-handed batters.[5]

[5] Inspired by this, we might wonder if the percentage of games started by left-handed pitchers have been going up over time, and indeed they have. This is one thing I like about fitting hierarchical models like these: they don't just improve your estimation, they can also give you insights into your data.

8.3.2 Posterior distributions

Let's go back to those two batters with a record of 30 hits out of 100 at-bats. We've now seen that this would be a different question in different years. Let's consider what it would look like in three different years, each 50 years apart.

```
players <- crossing(year = c(1915, 1965, 2015),
                    bats = c("L", "R"),
                    H = 30,
                    AB = 100)
```

```
players_posterior <- players %>%
  mutate(mu = predict(fit4, what = "mu", newdata = players),
         sigma = predict(fit4, what = "sigma",
                         newdata = players, type = "response"),
         alpha0 = mu / sigma,
         beta0 = (1 - mu) / sigma,
         alpha1 = alpha0 + H,
         beta1 = beta0 + AB - H)

players_posterior
```

```
## # A tibble: 6 × 10
##    year  bats     H    AB    mu   sigma
##   <dbl> <chr> <dbl> <dbl> <dbl>   <dbl>
## 1  1915     L    30   100 0.230 0.00141
## 2  1915     R    30   100 0.211 0.00141
## 3  1965     L    30   100 0.208 0.00141
## 4  1965     R    30   100 0.201 0.00141
## 5  2015     L    30   100 0.212 0.00141
## 6  2015     R    30   100 0.212 0.00141
## # ... with 4 more variables: alpha0 <dbl>,
## #   beta0 <dbl>, alpha1 <dbl>, beta1 <dbl>
```

We can see in Figure 8.6 how these posterior distributions (the `alpha1` and `beta1` we chose) differ for each of the three. If this comparison had happened in 1915, you may have wanted to pick the left-handed batter. We wouldn't have been sure he was better (we'd need a method from Chapter 6 to determine that), but it was more likely than not. But today, there'd be basically no reason to: left- versus right- handedness has almost no extra information.

8.4 Uncertainty in hyperparameters

We've followed the philosophy of empirical Bayes so far: we fit hyperparameters (α_0, β_0, or our coefficients for time and handedness) for our model using maximum likelihood (e.g. beta-binomial regression), and then use that as the prior for each of our observations.

There's a problem I've been ignoring so far with the empirical Bayesian approach, which is that there's uncertainty in these hyperparameters as well. When we come up with an alpha and beta, or come up with particular coefficients over time, we are treating those as fixed knowledge, as if these are the priors we "entered" the experiment with. But in fact each of these parameters were chosen from this same data, and in fact each comes with a confidence interval that we're en-

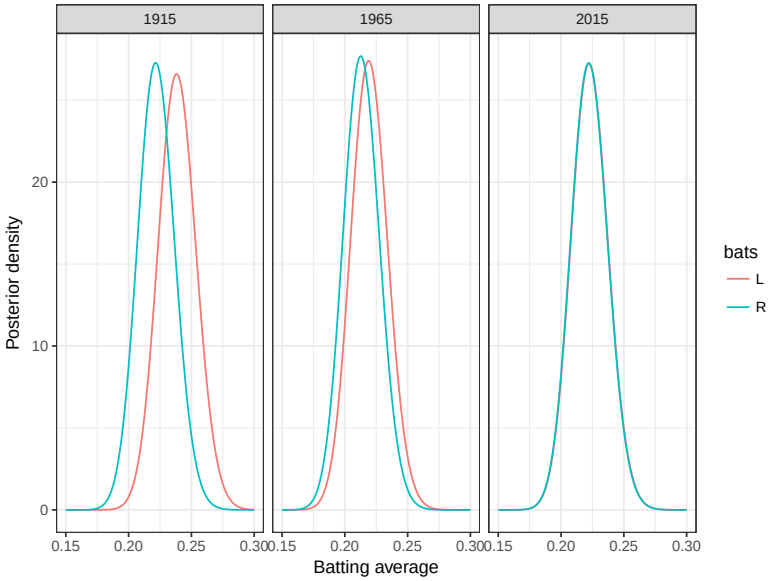

Figure 8.6: Posterior distributions for batters with a record of 30 / 100, whether left- or right- handed and at different points in time.

tirely ignoring. This is sometimes called the "double-dipping" problem among critics of empirical Bayes.

This wasn't a big deal when we were estimating just α_0 and β_0 for the overall dataset: we had so much data, and were estimating so few parameters, that we could feel good about the approach. But now that we're fitting this many parameters, we're *pushing it*. Actually quantifying this, and choosing methods robust to the charge of "double-dipping", involves Bayesian methods outside the scope of this book. But I wanted to note that this chapter reaches approximately the edge of what empirical Bayes can be used for reliably.

9

Mixture models and expectation-maximization

So far, we've been treating our overall distribution of batting averages as a beta distribution, which is a simple distribution between 0 and 1 that has a single peak. But what if that weren't a good fit? For example, what if we had a multimodal distribution, with multiple peaks?

In this chapter, we're going to consider what to do when your binomial proportions are made up of multiple peaks, and when you don't know which observation belongs to which clusters. For example, so far in our analysis we've been filtering out pitchers, who tend to have a much lower batting average than non-pitchers. If you include them, the data instead have two separate modes.

Imagine that you *didn't know* which players were pitchers, and you wanted to separate the data into two groups according to your best prediction. This is very common in practical machine learning applications, such as clustering and segmentation.

We'll now examine mixture models, where we treat the distribution of batting averages as a **mixture of two beta-binomial distributions**, and need to guess which player belongs to which group. This will also introduce the concept of an expectation-maximization algorithm, which is important in both Bayesian and frequentist statistics. We'll show how to calculate a posterior probability for the cluster each player belongs to, and see that mixture models are still a good fit for the empirical Bayes framework.

9.1 Setup

As usual, we start with code that sets up the variables analyzed in this chapter. [1]

```
library(dplyr)
library(tidyr)
library(Lahman)
```

[1] We're changing this analysis slightly to look only at National League batters since the year 1980. Why? Because National League pitchers are required to bat (while American League pitchers don't in typical games), and because focusing on modern batters helps reduce the noise within each group.

```
library(ggplot2)
theme_set(theme_bw())

# Identify those who have pitched at least three games
pitchers <- Pitching %>%
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
  filter(gamesPitched > 3)

career <- Batting %>%
  filter(AB > 0, lgID == "NL", yearID >= 1980) %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB), year = mean(yearID)) %>%
  mutate(average = H / AB,
         isPitcher = playerID %in% pitchers$playerID)

# Add player names
career <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast, bats) %>%
  unite(name, nameFirst, nameLast, sep = " ") %>%
  inner_join(career, by = "playerID")
```

We've been filtering out pitchers in the previous chapters, which make batting averages look roughly like a beta distribution. But when we leave them in, the data looks a lot less like a beta, as shown in Figure 9.1.

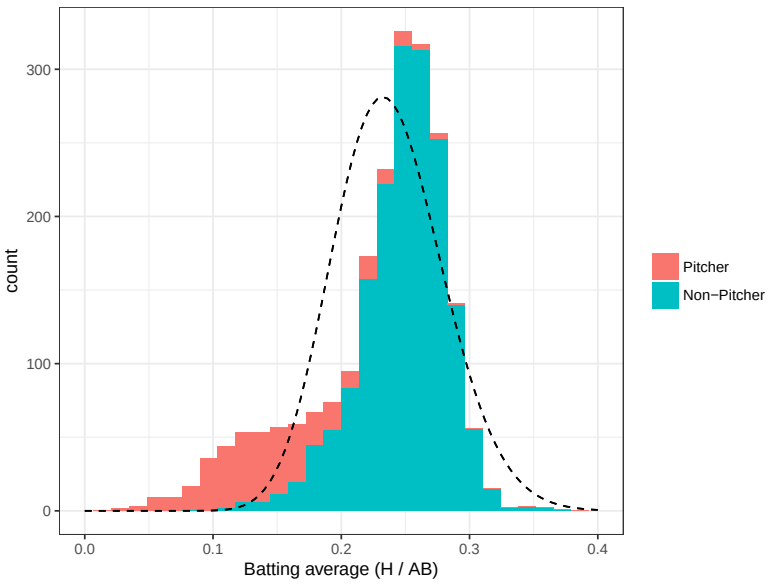

Figure 9.1: The distribution of batting averages when pitchers are included. The beta distribution that would be fit by maximum likelihood is shown as a dashed line.

The dashed density curve represents the beta distribution we would naively fit to this data. We can see that unlike our earlier analysis, where we'd filtered out pitchers, the beta is not a good fit- but that it's plausible that we could fit the data using *two* beta distributions, one for pitchers and one for non-pitchers.

In this example, we know which players are pitchers and which aren't. But if we didn't, we would need to assign each player to a distribution, or "cluster", before performing shrinkage on it. In a real analysis it's not realistic that we wouldn't know which players are pitchers, but it's an excellent illustrative example of a mixture model and of expectation-maximization algorithms.

9.2 Expectation-maximization

The challenge of mixture models is that at the start, we don't know which observations belong to which cluster, nor what the parameters of each distribution is. It's difficult to solve these problems at the same time- so an expectation-maximization (EM) algorithm takes the jump of estimating them one at a time, and **alternating** between them.

The first thing to do in an EM clustering algorithm is to assign our clusters **randomly**.

```
set.seed(2016)

# We'll fit the clusters only with players that have had at least 20 at-bats
starting_data <- career %>%
  filter(AB >= 20) %>%
  select(-year, -bats, -isPitcher) %>%
  mutate(cluster = factor(sample(c("A", "B"), n(), replace = TRUE)))
```

9.2.1 Maximization

Now that we've got cluster assignments, we can examine the densities of each cluster (Figure 9.2). It doesn't look like much of a division- they have basically the same density! That's OK: one of the nice features of expectation-maximization is that we don't actually have to start with good clusters to end up with a good result.

We'll now write a function for fitting a beta-binomial distribution using maximum likelihood estimation (and the `dbetabinom.ab` function from the VGAM package). This is a process we performed before in Chapter 3.2: here we're just encapsulating it into a function.[2]

```
library(VGAM)
```

[2] For the sake of simplicity, we didn't use our beta-binomial regression approach from Chapter 7 that takes into account the relationship between batting average and AB. In a more comprehensive model, we could change the maximization step to incorporate that estimation.

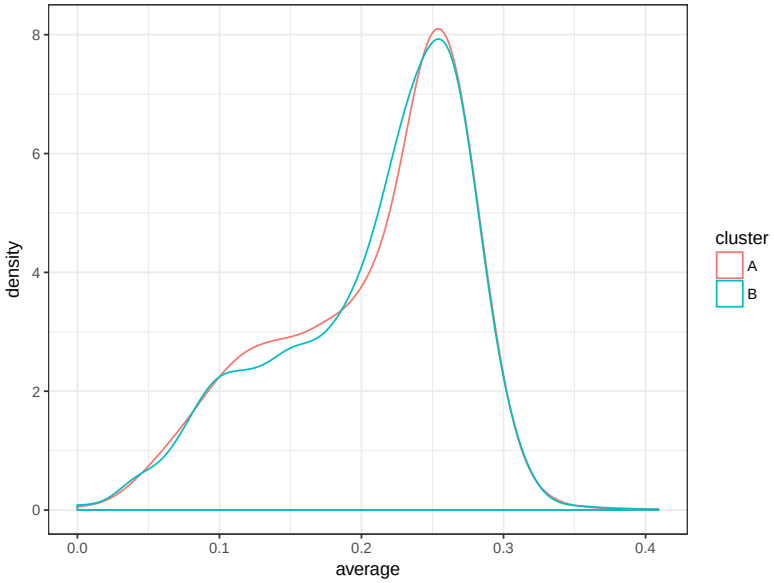

Figure 9.2: The density of batting averages among players assigned to cluster A or to cluster B.

```
fit_bb_mle <- function(x, n) {
  # dbetabinom.ab is the likelihood function for a beta-binomial
  # using n, alpha and beta as parameters
  ll <- function(alpha, beta) {
    -sum(dbetabinom.ab(x, n, alpha, beta, log = TRUE))
  }
  m <- stats4::mle(ll, start = list(alpha = 3, beta = 10),
                   method = "L-BFGS-B", lower = c(0.001, .001))
  ab <- stats4::coef(m)
  data_frame(alpha = ab[1], beta = ab[2])
}
```

For example, here are the alpha and beta chosen for the entire data as a whole:

```
fit_bb_mle(starting_data$H, starting_data$AB)
```

```
## # A tibble: 1 × 2
##   alpha  beta
##   <dbl> <dbl>
## 1  12.8  45.5
```

Now we're working with a mixture model, so finding α_0 and β_0 for the overall data won't work. This time, we're going to fit the model within each of our (randomly assigned) clusters.

```
fits <- starting_data %>%
  group_by(cluster) %>%
  do(fit_bb_mle(.$H, .$AB)) %>%
```

```
  ungroup()

fits
```

```
## # A tibble: 2 × 3
##   cluster alpha  beta
##    <fctr> <dbl> <dbl>
## 1       A  12.1  43.3
## 2       B  13.6  47.9
```

This was the maximization step: find the maximum likelihood parameters (in this case, two alpha/beta values, and a per-cluster probability), pretending we knew the assignments.

9.2.2 Expectation

We now have an estimated density for each cluster (Figure 9.3). It's worth noting that these are pretty similar distributions, and that neither is a good fit to the data.

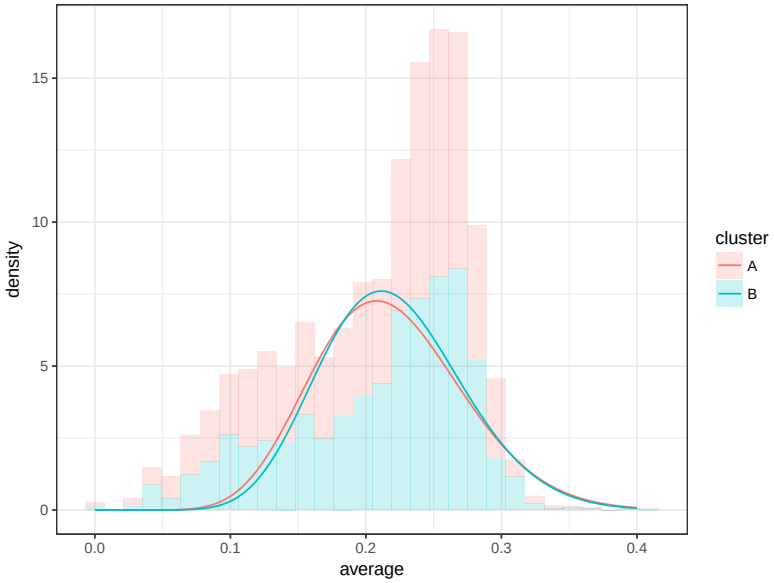

Figure 9.3: Density within each of the randomly assigned clusters, along with a histogram of the cluster assignments.

However, notice that due to a small random difference, cluster B is **slightly** more likely than cluster A for batting averages above about .2, and vice versa below .2.

Consider therefore that each player has a likelihood it would have been generated from cluster A, and a likelihood it would have been generated from cluster B. We can use `VGAM::dbetabinom.ab` to calculate these likelihoods.

```
crosses <- starting_data %>%
  select(-cluster) %>%
```

```
  crossing(fits) %>%
  mutate(likelihood = VGAM::dbetabinom.ab(H, AB, alpha, beta))

crosses
```

```
## # A tibble: 6,418 × 9
##     playerID               name     H    AB
##        <chr>              <chr> <int> <int>
## 1  abbotje01        Jeff Abbott    11    42
## 2  abbotje01        Jeff Abbott    11    42
## 3  abbotji01         Jim Abbott     2    21
## 4  abbotji01         Jim Abbott     2    21
## 5  abbotku01        Kurt Abbott   475  1860
## 6  abbotku01        Kurt Abbott   475  1860
## 7  abbotky01        Kyle Abbott     3    31
## 8  abbotky01        Kyle Abbott     3    31
## 9  abercre01 Reggie Abercrombie    86   386
## 10 abercre01 Reggie Abercrombie    86   386
## # ... with 6,408 more rows, and 5 more
## #   variables: average <dbl>,
## #   cluster <fctr>, alpha <dbl>, beta <dbl>,
## #   likelihood <dbl>
```

For example, consider Jeff Abbott, who got 11 hits out of 42 at-bats. He had a 8.95% chance of getting that if he were in cluster A, but a 9.26% chance if he were in cluster B. For that reason (even though it's a small difference), we'll put him in B. Similarly we'll put Kyle Abbott in cluster A: 3/31 was more likely to come from that distribution.

We can do that for every player using `group_by` and `top_n`:

```
assignments <- starting_data %>%
  select(-cluster) %>%
  crossing(fits) %>%
  mutate(likelihood = VGAM::dbetabinom.ab(H, AB, alpha, beta)) %>%
  group_by(playerID) %>%
  top_n(1, likelihood) %>%
  ungroup()

assignments
```

```
## # A tibble: 3,209 × 9
##     playerID               name     H    AB
##        <chr>              <chr> <int> <int>
## 1  abbotje01        Jeff Abbott    11    42
## 2  abbotji01         Jim Abbott     2    21
```

```
## 3  abbotku01        Kurt Abbott   475  1860
## 4  abbotky01        Kyle Abbott     3    31
## 5  abercre01 Reggie Abercrombie    86   386
## 6  abnersh01        Shawn Abner   110   531
## 7  abreubo01        Bobby Abreu  1607  5395
## 8  abreuto01         Tony Abreu   129   509
## 9  acevejo01       Jose Acevedo     8   101
## 10 aceveju01       Juan Acevedo     6    65
## # ... with 3,199 more rows, and 5 more
## #   variables: average <dbl>,
## #   cluster <fctr>, alpha <dbl>, beta <dbl>,
## #   likelihood <dbl>
```

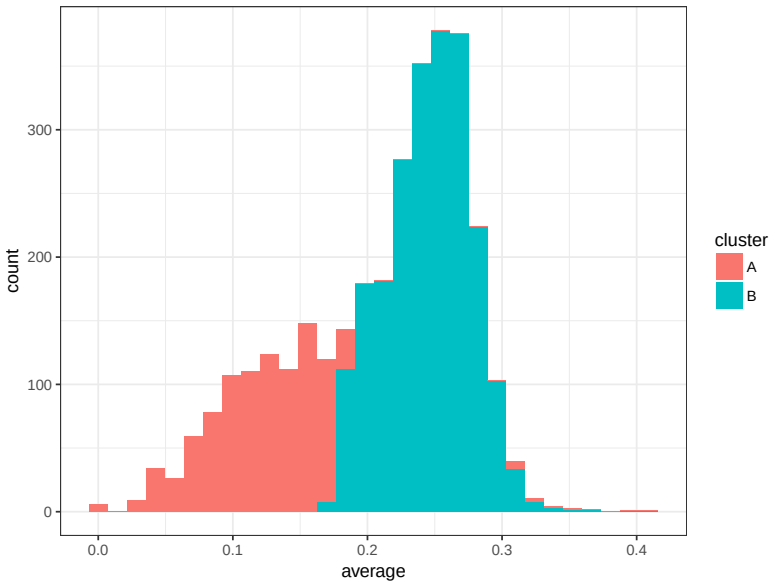

Figure 9.4: Assignments of players to clusters based on the beta-binomial model.

That's the expectation step: **assigning each person to the most likely cluster**.

Figure 9.4 shows the histogram of player assignments. Something really important happened here: even though the two beta models we'd fit were very similar, we still split up the data rather neatly. Generally batters with a higher average ended up in cluster B, while batters with a lower average were in cluster A. (Note that due to B having a slightly higher prior probability, it was possible for players with a low average- but also a low AB- to be assigned to cluster B).

9.2.3 Iteration over expectation and maximization

The above two steps got to a better set of assignments than our original, random ones. But there's no reason to believe these are as good

as we can get. So we **repeat** the two steps, choosing new parameters for each distribution in the mixture and then making new assignments each time.

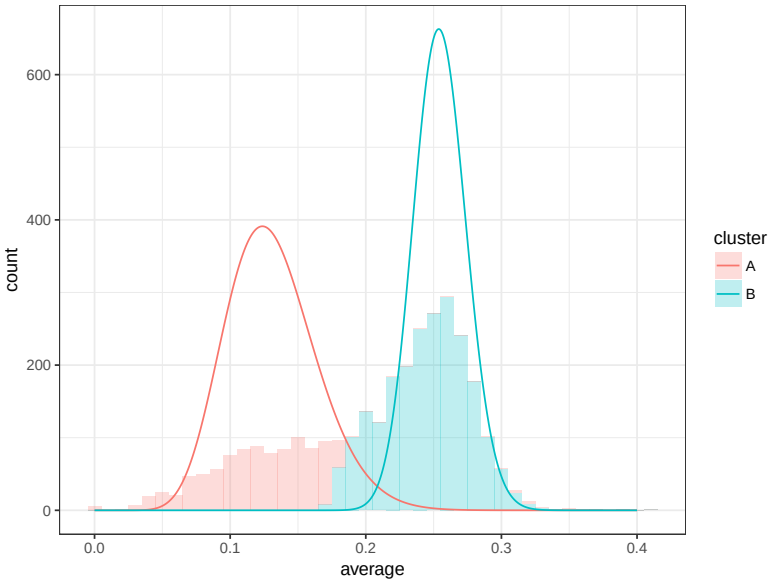

Figure 9.5: Distribution of the beta-binomial density, fit to each of the clusters assigned in the last iteration.

For example, now that we've reassigned each player's cluster, we could re-fit the beta-binomial with the new assignments (Figure 9.5). Unlike our first model fit, we can see that cluster A and cluster B have diverged a lot. Now we can take those parameters and perform a new estimation step. Generally we will do this multiple times, as an iterative process. This is the heart of an expectation-maximization algorithm, where we switch between assigning clusters (expectation) and fitting the model from those clusters (maximization).

```
set.seed(1337)

iterate_em <- function(state, ...) {
  # maximization
  fits <- state$assignments %>%
    group_by(cluster) %>%
    do(fit_bb_mle(.$H, .$AB)) %>%
    ungroup()

  # expectation
  assignments <- state$assignments %>%
    select(playerID:average) %>%
    crossing(fits) %>%
    mutate(likelihood = VGAM::dbetabinom.ab(H, AB, alpha, beta)) %>%
    group_by(playerID) %>%
```

```
    top_n(1, likelihood) %>%
    ungroup()

  list(assignments = assignments, fits = fits)
}

library(purrr)
init <- list(assignments = starting_data)
iterations <- accumulate(1:5, iterate_em, .init = init)
```

Here I used the `accumulate` function from the `purrr` package, which is useful for running data through the same function repeatedly and keeping intermediate states. I haven't seen others use this tidy approach to EM algorithms, and there are existing R approaches to mixture models. I like this approach both because it's transparent about what we're doing in each iteration, and because our iterations are now combined in a tidy format that's easy to summarize and visualize.

We could visualize how our assignments changed over the course of the iteration (Figure 9.6).

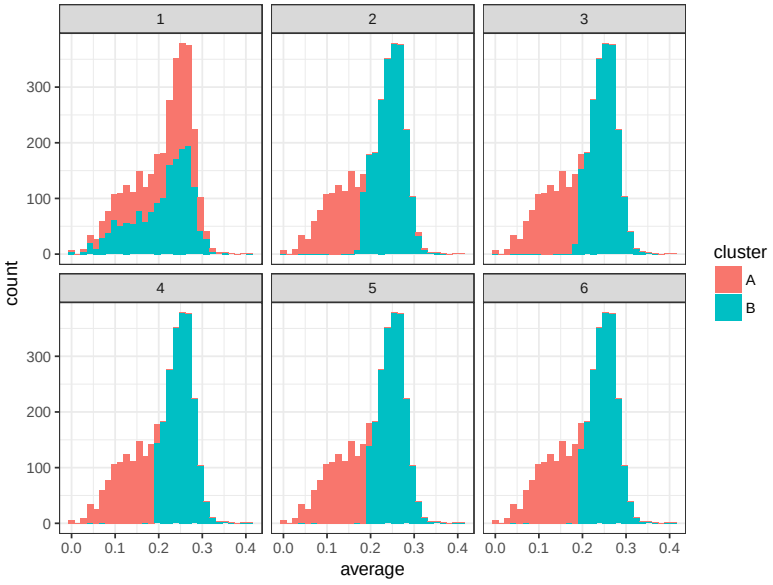

Figure 9.6: Histogram of the assignments of players to clusters A and B at each iteration of the expectation-maximization algorithm.

We notice that only the first few iterations led to a shift in the assignments, after which it appears to converge to stable assignments to clusters A and B. When the assignments converge, the estimated parameters will reach a steady point as well.

```
fit_iterations %>%
  crossing(x = seq(.001, .4, .001)) %>%
```

```
  mutate(density = dbeta(x, alpha, beta)) %>%
  ggplot(aes(x, density, color = iteration, group = iteration)) +
  geom_line() +
  facet_wrap(~ cluster)
```

9.3 Assigning players to clusters

We now have estimated α and β parameters for each of the two clusters.

```
final_parameters <- last(iterations)$fits

final_parameters
```

```
## # A tibble: 2 × 3
##   cluster alpha  beta
##    <fctr> <dbl> <dbl>
## 1       A  27.1   170
## 2       B 145.4   419
```

How would we assign players to clusters to get a posterior probability that the player belongs to that cluster? Well, let's arbitrarily pick the six players that each batted exactly 100 times.

name	H	AB	year	average	isPitcher
Jose de Jesus	11	100	1990	0.11	TRUE
Juan Nicasio	12	100	2012	0.12	TRUE
Mike Mahoney	18	100	2002	0.18	FALSE
Robinson Cancel	20	100	2007	0.20	FALSE
Mike Busch	22	100	1996	0.22	FALSE
Ryan Shealy	32	100	2006	0.32	FALSE

Notice that two of them actually were pitchers, and four were not. Where would our mixture model classify each of them? Well, we'd consider the likelihood each would get the number of hits they did if they were a pitcher in cluster A or a non-pitcher in cluster B (Figure 9.7).

By Bayes' Theorem, we can simply use the ratio of one likelihood (say, A in red) to the sum of the two likelihoods to get the posterior probability (Figure 9.8).

Based on this, we feel confident that Juan Nicasio and Jose de Jesus are pitchers, and that Ryan Shealy isn't, but we'd be a bit less sure about Mahoney, Cancel, and Busch.[3]

This allows us to assign all players in the dataset to one of the two clusters.

```
career_likelihoods <- career %>%
  filter(AB > 20) %>%
```

[3] By checking the `isPitcher` column in the table above, we can see that we were right about Nicasio, de Jesus, and Shealy. As it turns out none of Mahoney, Cancel, and Busch were pitchers, but were rather relatively weak batters.

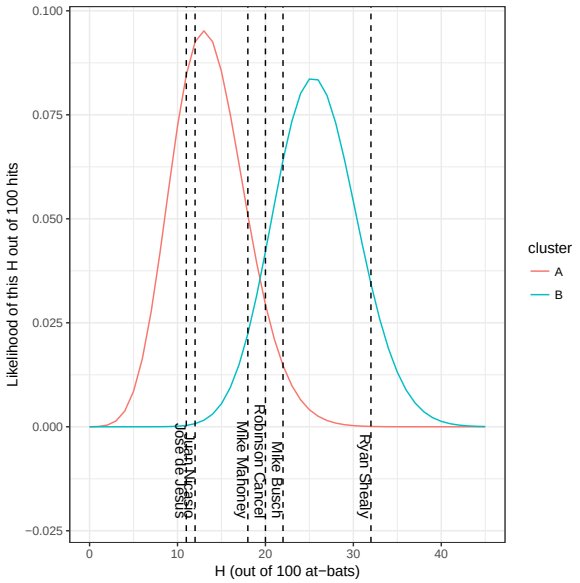

Figure 9.7: The likelihood that cluster A or cluster B would generate each of these six players' records.

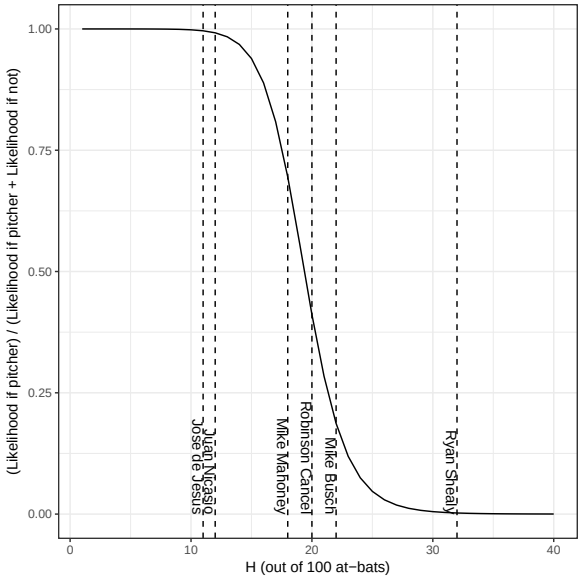

Figure 9.8: The posterior probability that each of the 6 players with 100 at-bats is in the pitcher cluster.

```
  crossing(final_parameters) %>%
  mutate(likelihood = VGAM::dbetabinom.ab(H, AB, alpha, beta)) %>%
  group_by(playerID) %>%
  mutate(posterior = likelihood / sum(likelihood))

career_assignments <- career_likelihoods %>%
  top_n(1, posterior) %>%
  ungroup()
```

Since we know whether each player actually is a pitcher or not, we can also compute a confusion matrix. How many pitchers were accidentally assigned to cluster B, and how many non-pitchers were assigned to cluster A? In this case we'll look only at the ones for which we had at least 80% confidence in our classification.

True category	A	B
Non-pitcher	106	1697
Pitcher	697	55

This isn't bad, considering the only information we used was the batting average. Note that we didn't even use data on who were pitchers to train the model, but just let the clusters of batting averages define themselves.

9.4 Empirical bayes shrinkage with a mixture model

We've gone to all this work to find posterior probabilities of each player's assignments to one of two clusters. How can we use this in empirical Bayes shrinkage, or with the other methods we've described in this book?

Well, consider that all of our other methods have worked because the posterior was another beta distribution (thanks to the beta being the conjugate prior of the binomial). However, now that each point might belong to one of two beta distributions, our posterior will be a *mixture* of betas. This mixture is made up of the posterior from each cluster, weighted by the probability the point belongs to that cluster.

For example, consider the aforementioned six players who had exactly 100 at-bats. Their posterior distributions are shown in Figure 9.9.

For example, we are pretty sure that Jose de Jesus and Juan Nicasio are part of the "pitcher" cluster, so that makes up most of their posterior mass, and all of Ryan Shealy's density is in the "non-pitcher" cluster. However, we're pretty split on Mike Mahoney and Robinson Cancel: each could be a pitcher who is unusually good at batting, or a non-pitcher who is unusually bad.

Can we perform shrinkage like we did in Chapter 3? If our goal is

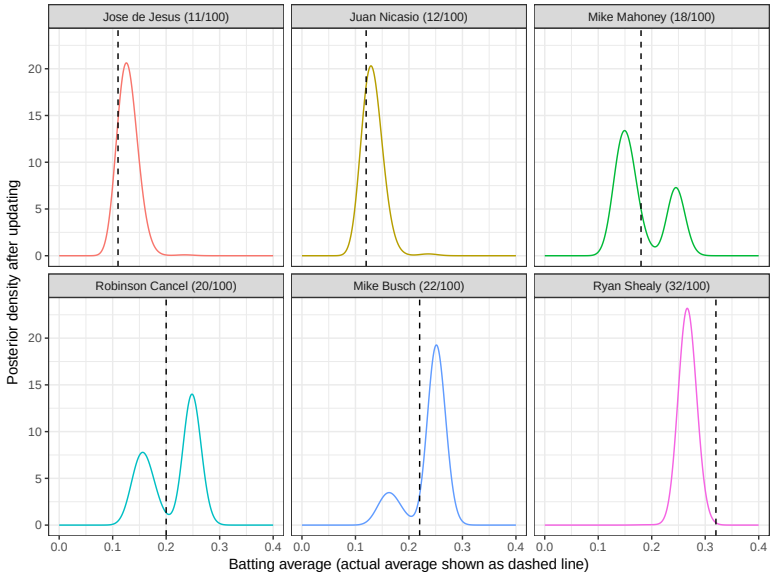

Figure 9.9: Posterior distributions for the batting average of each of the six players with 100 at-bats. Each player's raw batting average is shown as a dashed vertical line.

still to find the mean of each posterior, then yes! Thanks to linearity of expected value, we can simply average the two distribution means, weighing each by the probability the player belongs to that cluster.

$$\hat{p} = \Pr(A)\frac{\alpha_A + H}{\alpha_A + \beta_A + AB} + \Pr(B)\frac{\alpha_B + H}{\alpha_B + \beta_B + AB}$$

This calculation can be performed in code as a `group_by()` and `summarize()` for each player.

```
eb_shrinkage <- career_likelihoods %>%
  mutate(shrunken_average = (H + alpha) / (AB + alpha + beta)) %>%
  group_by(playerID) %>%
  summarize(shrunken_average = sum(posterior * shrunken_average))
```

For example, we are pretty sure that Jose de Jesus and Juan Nicasio are part of the "pitcher" cluster (high $\Pr(A)$, low $\Pr(B)$), which means they mostly get shrunken towards that center. We are quite certain Ryan Shealy is not a pitcher, so he'll be updated based entirely on the non-pitcher distribution.

We can see how this affects the shrunken estimates, compared to shrinking towards a single prior, in terms of the relationship between AB and the estimate (Figure 9.10).

Notice that instead of shrinking towards a single value, as we would if we applied the estimation method from Chapter 3, the batting averages are now shrunken towards two centers: one higher value for the non-pitcher cluster, one smaller value for the pitcher cluster. Players that are exactly in between don't really get shrunken in either direction- they're "pulled equally" by both.

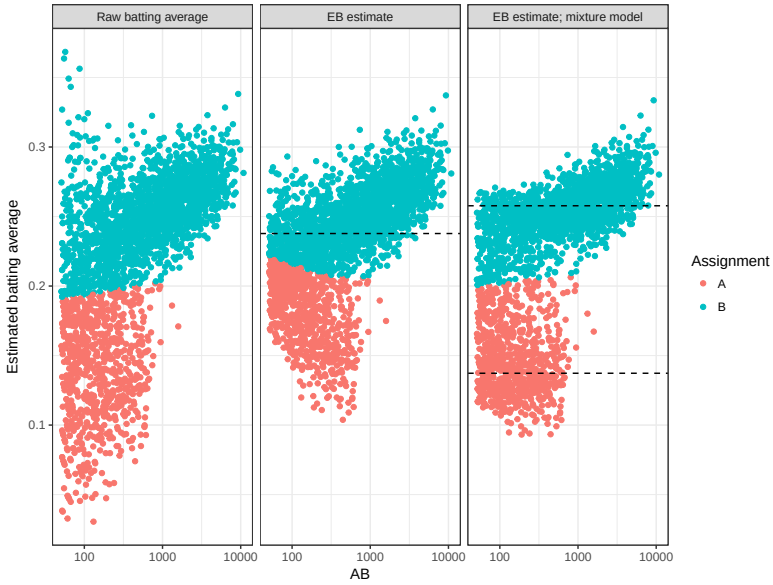

Figure 9.10: Effect of empirical Bayes shrinkage towards either a single beta prior or a mixture model. The prior means of the overall model or the clusters are shown as dashed lines. Colors are assigned based on the mixture model.

Not all of the methods in this book are so easy to adapt to a multimodal distribution. For example, a credible interval (Chapter 12.3) is ambiguous in a multimodal distribution[4], and we'd need to rethink our approach to Bayesian A/B testing (Chapter 6). But since we do have a posterior distribution for each player- even though it's not a beta- we'd be able to face these challenges.

[4] For example, we'd have to choose whether to pick the 2.5% and 97.5% quantiles, or to pick the 95% containing the most probability density. The latter might be *two separate intervals* for one observation, e.g. "the batting average is either in $(.15, .19)$ or in $(.25, .28)$".

10
The multinomial and the Dirichlet

So far in this book, we've been focusing on the batting average (the number of hits divided by the number of at-bats) as a summary of each player's batting ability. Besides being one of the most common baseball statistics, it's a perfect example of the binomial distribution.

But this over-simplifies what makes batters contribute to a team's offense, because not all hits are equal! Each hit could be a single, double, triple, or home run.[1] A batting average counts each of these results identically, but a player would much rather advance to a triple or home run than get a single.

[1] If you don't follow baseball, just know that these relate to how far a player gets to advance around the baseball diamond after their hit, based on the strength of their hit and how well the other team fielded the ball. Advancing farther (with a double, triple, or especially a home run) is better than a single.

In this chapter, we're going to consider a different measure of batting skill: slugging percentage, which sabermetricians generally prefer to batting average as a measure of offensive power. The catch is that we are no longer dealing with a "success / total" model, but rather describing multiple categories (singles, doubles, and so on) out of the total at-bats. This means we have to move away from the binomial and the beta, and introduce two new distributions: the **multinomial** and the **Dirichlet**.

10.1 Setup

Until now we've only needed H (hits) and AB (at-bats) from the Lahman baseball database (as well as handedness and time in Chapter 11.3), but now we need to extract the number of singles, doubles, triples, and home runs from each player over their entire career.

```
library(Lahman)
library(dplyr)
library(tidyr)

pitchers <- Pitching %>%
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
```

```
  filter(gamesPitched > 3)

player_names <- Master %>%
  transmute(playerID, name = paste(nameFirst, nameLast))

# include the "bats" (handedness) and "year" column for later
hit_types <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  rename(Double = X2B, Triple = X3B) %>%
  group_by(playerID) %>%
  summarize_each(funs(sum(., na.rm = TRUE)), AB, H, Double, Triple, HR) %>%
  inner_join(player_names, by = "playerID") %>%
  transmute(playerID, name, AB, H,
            Single = H - Double - Triple - HR,
            Double, Triple, HR,
            NonHit = AB - H)
```

Here's the data we've collected for a few players.

name	AB	H	Single	Double	Triple	HR	NonHit
Hank Aaron	12364	3771	2294	624	98	755	8593
Tommie Aaron	944	216	155	42	6	13	728
Andy Abad	21	2	2	0	0	0	19
John Abadie	49	11	11	0	0	0	38
Ed Abbaticchio	3044	772	619	99	43	11	2272
Charlie Abbey	1751	492	360	67	46	19	1259

For example, Hank Aaron got 3771 hits out of 12,364 at-bats, and these hits included 755 home-runs (one of the highest records in history).

10.1.1 Slugging percentage

These outcomes make up a percentage of each player's total hits. Figure 10.1 shows the distribution of each of these percentages.

We can see that the most common kind of hit is a single, followed by doubles, with triples being the rarest type of hit.[2] Notice that each of these distributions looks roughly like a beta distribution; that will become important later.

[2] Triples are rarest because any time the ball gets hit out of the park counts as a home run, but good outfielders can usually stop a player from getting to third base. You have to be pretty fast to get a triple in professional baseball!

In order to take these particular categories of hits into account, we can use a slugging percentage (abbreviated SLG). This is computed as:

$$\text{SLG} = \frac{\text{Singles} + 2 \cdot \text{Doubles} + 3 \cdot \text{Triples} + 4 \cdot \text{HR}}{\text{AB}}$$

Notice that unlike the batting average, this gives more weight to more useful types of hits, particularly home runs. Also notice that this

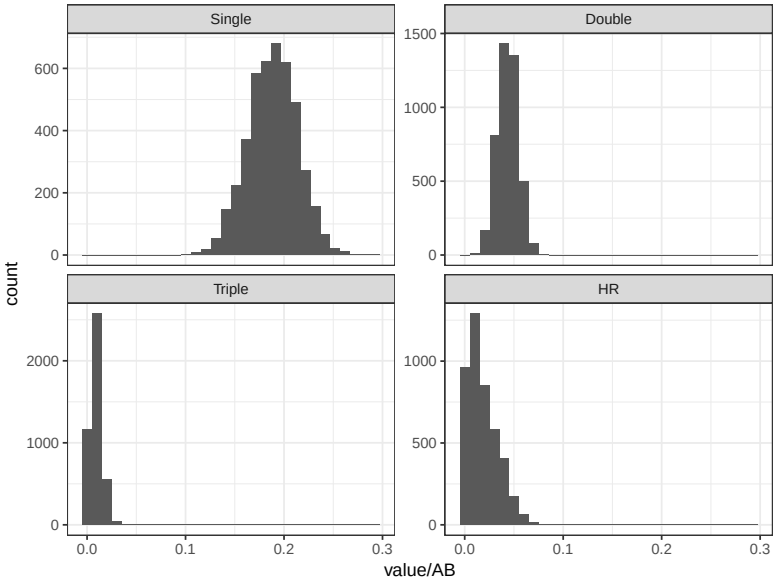

Figure 10.1: Percentages of each players' hits that are made up of singles, doubles, triples, or home runs.

will lie between 0 (if a player has no hits) and 4.000 (if a player's hits were entirely home runs).

It's straightforward to use R to compute this for each player.

```
hit_types <- hit_types %>%
  mutate(slugging = (Single + 2 * Double + 3 * Triple + 4 * HR) / AB)
```

Much like in Chapter 3, we might be interested in the highest slugging averages across all players.

name	H	Single	Double	Triple	HR	NonHit	slugging
Charlie Lindstrom	1	0	0	1	0	0	3.0
Frank O'Connor	2	1	0	0	1	0	2.5
Hal Deviney	2	1	0	1	0	0	2.0
Uel Eubanks	1	0	1	0	0	0	2.0
Roy Gleason	1	0	1	0	0	0	2.0
Larry Gowell	1	0	1	0	0	0	2.0

Just as we saw when we examined players with the highest batting average, this definitely isn't what we're looking for. The top slugging averages are dominated by players with a single triple, or a single and a home run. We need a way to estimate trustworthy slugging averages in the presence of noise.

Empirical Bayes comes to the rescue once again!

10.2 The Dirichlet-multinomial distribution

In the rest of this book, we've been assuming that there were two possible outcomes from each at-bat: a hit, or non-hit. Since there were

two possible outcomes (a hit and a non-hit), we were able to represent it as a binomial, like a coin flip.

Now that we're examining five types of hits, we need a new distribution.

10.2.1 Multinomial distribution

There are five possible outcomes from each at-bat:

- Single
- Double
- Triple
- Home run
- Non-hit

Rather than a coin flip like the binomial, this is like a die roll: in this case, a five sided die where each side has a probability of coming up. This is described by the **multinomial** distribution.

The multinomial distribution is characterized by two parameters: n, here the number of at-bats, and $p_{1...k}$, a vector of probabilities for each category (here $k = 5$). For example, if all five types were equally likely, p would be the vector $(.2, .2, .2, .2, .2)$.

We can use the `rmultinom()` function to simulate draws from the multinomial. Here we're simulating 3 draws, where each of the 5 categories is equally likely.

```
rmultinom(3, 100, c(.2, .2, .2, .2, .2))
```

```
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   17   16   20
## [3,]   26   22   18
## [4,]   25   14   20
## [5,]   13   27   19
```

This results in a matrix, where each column is a draw from the multinomial, and each row is a category of the outcome. Notice that each of these columns adds up to 100, the total number of "at-bats".

We could change the vector of probabilities so that, for example, the first two category are relatively unlikely, the next two more likely, and the last outcome the most likely.

```
rmultinom(3, 100, c(.05, .05, .2, .2, .5))
```

```
##      [,1] [,2] [,3]
## [1,]    9    3    2
## [2,]    7    5    4
```

```
## [3,]   17   14   23
## [4,]   18   22   23
## [5,]   49   56   48
```

In the batting model, each of these rows could represent one of the categories of hit, such as a single, a double, or a non-hit.[3]

[3] Notice that the binomial is therefore a special case of the multinomial, in the case that $k = 2$.

10.2.2 Dirichlet distribution

The binomial had two categories (success and failure), and it had a convenient conjuate prior of the beta distribution. This meant that if our prior was a beta distribution and you observed some evidence, out posterior would also be a beta distribution.

There's an equivalent conjugate prior for the multinomial distribution, which is called the **Dirichlet distribution**. Instead of parameters α and β like the beta distribution, it has parameters $\alpha_{1...k}$: one value for each category of outcome.

The Dirichlet distribution isn't built into R, but we can use functions such as `rdiric()` from the VGAM package (Yee, 2016) to simulate draws from it.[4]

[4] Note that while `rmultinom()` creates one column per draw, `rdiric()` creates one row for each draw. Also note that in these examples, I'm using `VGAM::` rather than loading it because the VGAM package loads several functions that conflict with dplyr.

```
VGAM::rdiric(3, c(1, 1, 1, 1, 1))
```

```
##          [,1]   [,2]    [,3]   [,4]    [,5]
## [1,] 0.10789 0.2117 0.11176 0.4643 0.10439
## [2,] 0.02157 0.5427 0.01276 0.2325 0.19040
## [3,] 0.04389 0.4731 0.01410 0.4320 0.03696
```

Notice that each draw from the Dirichlet (each row) is made up of values that sum to 1. Just like the beta distribution could be used to generate a probability, this can be used to generate an allocation of probabilities across k (5) outcomes.

Intuitively, we can think of the Dirichlet distribution as being governed by two properties:

- The higher the relative value of one parameter $\alpha_i / \sum \alpha_{1...k}$, the more probability mass that category i tends to take up
- The higher the total value $\sum \alpha_{1...k}$, the less variance there is within each category.

For example, consider Figure 10.2, which shows histograms simulated from the Dirichlet distribution for three possible sets of parameters (shown on the right-hand side). Notice that since a draw from the Dirichlet is a vector of k values, a simulation of the Dirichlet needs to show one histogram for each category.

When the parameters are $(1, 1, 1, 1, 1)$, the five distributions are equal, each having a mean of .2. When they are $(5, 2, 2, 1, 1)$, the probability mass shifts towards the categories with the higher α_i. The

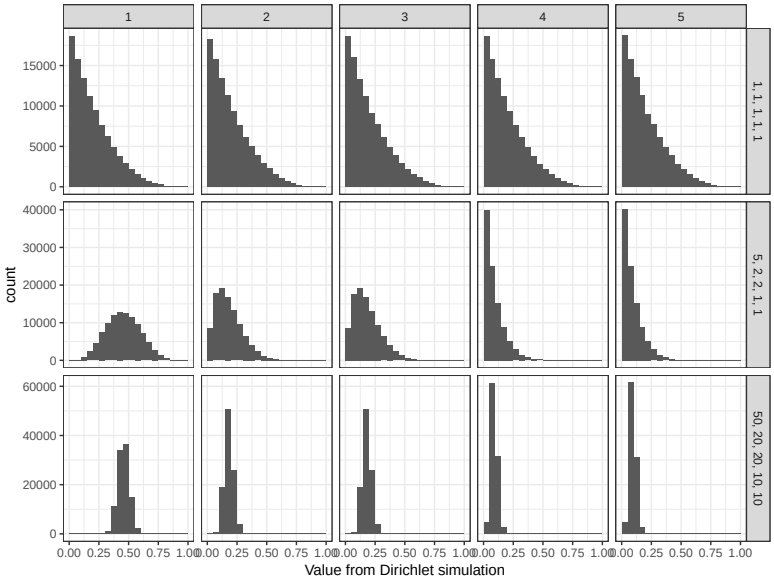

Figure 10.2: Histograms of simulated values from the Dirichlet distribution. Each row of graphs is one set of parameters, such as $(1, 1, 1, 1, 1)$, and each column is one of the five categories.

means are then, in turn, .5, .2, .2. .1, and .1. Thus, the relative size of the α_i values control which probabilities end up higher.

Besides the relative sizes of the parameters, the total $\sum \alpha_{1...k}$ also matters. When the parameters are $(500, 200, 200, 100, 100)$ (bottom row of Figure 10.2), the means are the same as when the parameters were $(5, 2, 2, 1, 1)$. However, there is much less variation within each group.

Notice that these properties, when considered within each individual category, are similar to how the beta distribution behaves (in which $\frac{\alpha}{\alpha+\beta}$ controlled the mean, and $\alpha + \beta$ affected the variance). Since this is the conjugate prior to the multinomial while the beta is conjugate prior to the binomial, this makes sense!

10.2.3 Fitting a Dirichlet-multinomial distribution

Just as we fit the beta-binomial distribution to batting average in Chapter 3, we'll have to fit a Dirichlet-multinomial distribution to this data to use it as a prior. The fastest and easiest way I've found to fit a Dirichlet-multinomial distribution is the (helpfully named) DirichletMultinomial package from Bioconductor (Morgan, 2016).[5]

[5] If you haven't used Bioconductor packages before, see here for instructions on installing it.

```
hit_500 <- hit_types %>%
  filter(AB >= 500)

hit_matrix <- hit_500 %>%
  select(Single, Double, Triple, HR, NonHit) %>%
  as.matrix()
```

```
dm_fit <- DirichletMultinomial::dmn(hit_matrix, 1)
```

The broom package doesn't have a tidying method for this type of object (DMN), but we can write one right now so that it's easy to extract parameters from it as a data frame.

```
library(broom)

tidy.DMN <- function(x, ...) {
  ret <- as.data.frame(x@fit)
  tbl_df(fix_data_frame(ret, c("conf.low", "estimate", "conf.high")))
}

dm_params <- tidy(dm_fit)
dm_params
```

```
## # A tibble: 5 × 4
##     term conf.low estimate conf.high
##    <chr>    <dbl>    <dbl>     <dbl>
## 1 Single   34.433   35.311    36.212
## 2 Double    8.203    8.419     8.641
## 3 Triple    1.801    1.853     1.906
## 4     HR    2.756    2.837     2.919
## 5 NonHit  134.601  137.999   141.482
```

How did these parameters compare to the data? Let's compare the expected density to the actual distribution for each of the categories of hits (10.3).

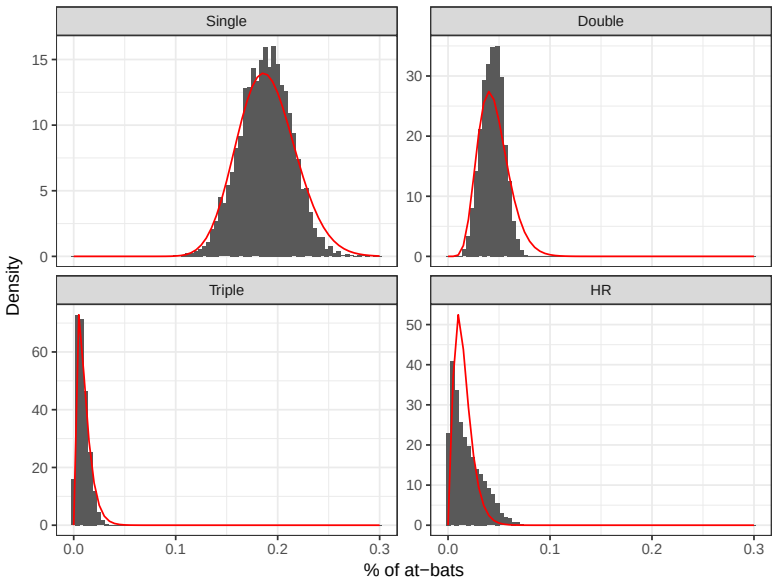

Figure 10.3: The density of the Dirichlet distribution as fit by maximum likelihood, compared to the histogram of percentages for each type of hit.

Notice that the fit was quite good, but not perfect. While the fit follows the distribution of singles and triples closely, it overestimated the variance in doubles, and underestimated the variance in home runs. There are a few possible reasons, but it's still good enough that we can use it for empirical Bayes shrinkage of the slugging average.

10.3 Updating the posterior distribution

Recall that when updating a beta prior with parameters α_0 and β_0, the posterior beta distribution was:

$$\text{Beta}(\alpha_0 + \text{Hits}, \beta_0 + \text{Misses})$$

The Dirichlet works in a similar way. Suppose our Dirichlet has five categories (as we have in this batting example), with prior parameters $\alpha_0^{(1...5)}$.[6] Then suppose we obesrve counts in the five categories: x_1 being the number of singles, x_2 the number of doubles, and so on.

Our new prior would be:

$$\text{Dirichlet}(\alpha_0^{(1)} + x_1, \alpha_0^{(2)} + x_2, \alpha_0^{(3)} + x_3, \alpha_0^{(4)} + x_4, \alpha_0^{(5)} + x_5)$$

Thus, much as the beta prior starts a player off with a certain number of hits and misses, the Dirichlet prior starts a player off with a number of singles, doubles, triples, home-runs, and misses.

[6] Almost any mathematically rigourous discussion of the Dirichlet would treat α_0 as a vector of length k. I'm referring to 5 categories here to make it especially clear what we're doing with the number of singles, doubles, triples, etc, rather than explaining it in mathematically general terms.

10.3.1 Empirical Bayes shrinkage of slugging percentage

You can use the Dirichlet prior, and the "pseudo-counts" that it starts each category off with, to shrink slugging percentages for each player.

```
# Extracting the pseudo-counts into a one-row data frame
par_total <- sum(dm_params$estimate)
par <- dm_params %>%
  select(term, estimate) %>%
  spread(term, estimate)

par
```

```
## # A tibble: 1 × 5
##   Double    HR NonHit Single Triple
## *  <dbl> <dbl>  <dbl>  <dbl>  <dbl>
## 1  8.419 2.837    138  35.31  1.853
```

For example, to perform empirical Bayes shrinkage on the slugging percentage, we'd add 8.4189 to the number of doubles, 2.8365 to the number of home runs, and so on, and *then* compute the slugging percentage. This is straightforward to implement.

```
w <- c(1:4, 0)
slugging_mean <- sum(w * dm_params$estimate) / sum(dm_params$estimate)
slugging_mean
```

```
## [1] 0.3704
```

```
hit_types_eb <- hit_types %>%
  mutate(slugging_eb = ((Single + par$Single) +
                          (Double + par$Double) * 2 +
                          (Triple + par$Triple) * 3 +
                          (HR + par$HR) * 4) /
           (AB + par_total))
```

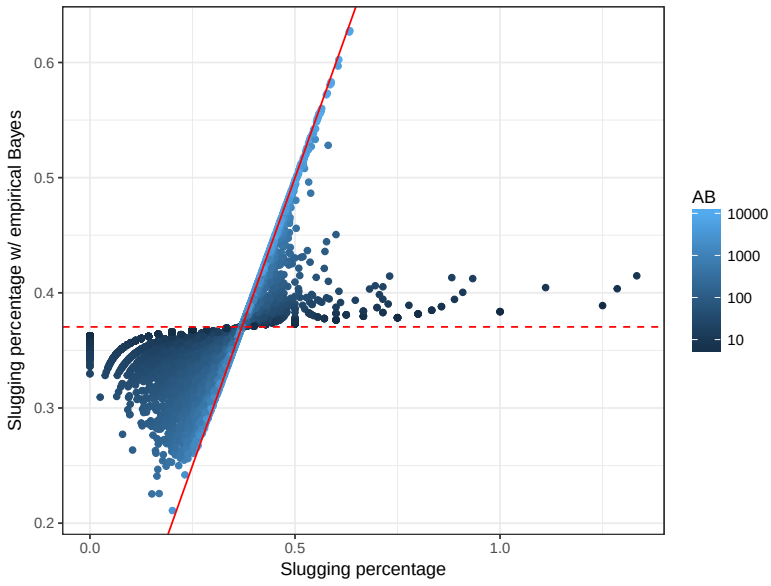

Figure 10.4: Comparison of the raw slugging percentage and the shrunken slugging percentage (for players at least 3 at-bats). The mean of the prior distribution is shown as a flat dashed line.

Figure 10.4 shows how this shrinkage changed our estimates of each player's slugging percentage. Notice that this looks a *lot* like the effect of empirical Bayes on our batting average estimates, back in Figure 3.3. We're seeing everything get shrunk towards our central estimate of 0.3704 (the estimated percentage for a player with no at-bats), except players with a lot of evidence.

We can now look at the batters with the best shrunken slugging averages in history.

playerID	name	AB	H	Single	Double	Triple	HR	NonHit	slugging	slugging_eb
willite01	Ted Williams	7706	2654	1537	525	71	521	5052	0.6338	0.6276
gehrilo01	Lou Gehrig	8001	2721	1531	534	163	493	5280	0.6324	0.6265
bondsba01	Barry Bonds	9847	2935	1495	601	77	762	6912	0.6069	0.6025
greenha01	Hank Greenberg	5193	1628	847	379	71	331	3565	0.6050	0.5969
pujolal01	Albert Pujols	7943	2519	1422	561	16	520	5424	0.5882	0.5832
mcgwima01	Mark McGwire	6187	1626	785	252	6	583	4561	0.5882	0.5818

This list is no longer topped by players who had only a few at-bats and got a home run or triple. Instead, we see several players from early in the game's history who were legendary for their home run ability, such as Ted Williams and Lou Gehrig. We also see Barry Bonds and Mark McGwire, two recent record holders for most home runs in a season.

Applying the Dirichlet-multinomial distribution allowed us to adjust slugging percentages the same way we have batting averages. It's also shown that the beta-binomial model we've used throughout the book is just one example of how useful the Bayesian approach is: many useful distributions (such as the normal or the Poisson) also have other distributions that serve as conjugate priors. Check out this list of conjugate priors for more.[7]

[7] You may have noticed a complication with our model: we didn't consider that five categories of hits aren't independent for each player, but rather have a natural order. It would be implausible for a player who either misses or gets home runs, with nothing in between. Other Bayesian models can handle this, as explored in (Agresti and Hitchcock, 2005), but it is beyond the scope of this book.

Part IV

Computation

11
The ebbr package

We've introduced a number of statistical techniques in this book: estimating a beta prior, beta-binomial regression, hypothesis testing, mixture models, and many other components of the empirical Bayes approach. These methods are useful whenever you have many observations of success/total data. In the final chapters, we're going to shift from introducing statistical methods to discuss using them in practice in R.

Each chapter has come with some code that implements the statistical methods described, which you're welcome to adapt for your own data.[1] However, you'd probably find yourself copying and pasting quite large amount of code, which can take you out of the flow of your own data analysis and introduces opportunities for mistakes.

In this chapter, we'll introduce the ebbr package for performing empirical Bayes on binomial data in R. The package offers convenient tools for performing almost all the analyses we've done during this series, along with documentation and examples. This also serves as a review of the entire book: we'll touch on each chapter briefly and recreate some of the key results.

[1] Previous chapters showed the code only for selected steps, such as when estimates were actually being computed, and almost never the code to generate figures. Since the remaining chapters are meant as a guide to using empirical Bayes, we'll include much more code, including code to visualize model results using ggplot2.

11.1 Setup

As usual, we start with code that sets up the variables analyzed in this chapter.

```
library(Lahman)
library(dplyr)
library(tidyr)

# Grab career batting average of non-pitchers
# (allow players that have pitched <= 3 games, like Ty Cobb)
pitchers <- Pitching %>%
  group_by(playerID) %>%
```

```
  summarize(gamesPitched = sum(G)) %>%
  filter(gamesPitched > 3)

# Add player names
player_names <- Master %>%
  tbl_df() %>%
  dplyr::select(playerID, nameFirst, nameLast, bats) %>%
  unite(name, nameFirst, nameLast, sep = " ")

# include the "bats" (handedness) and "year" column for later
career_full <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB), year = mean(yearID)) %>%
  inner_join(player_names, by = "playerID") %>%
  filter(!is.na(bats))

# We don't need all this data for every step
career <- career_full %>%
  select(-bats, -year)
```

11.2 Empirical Bayes estimation

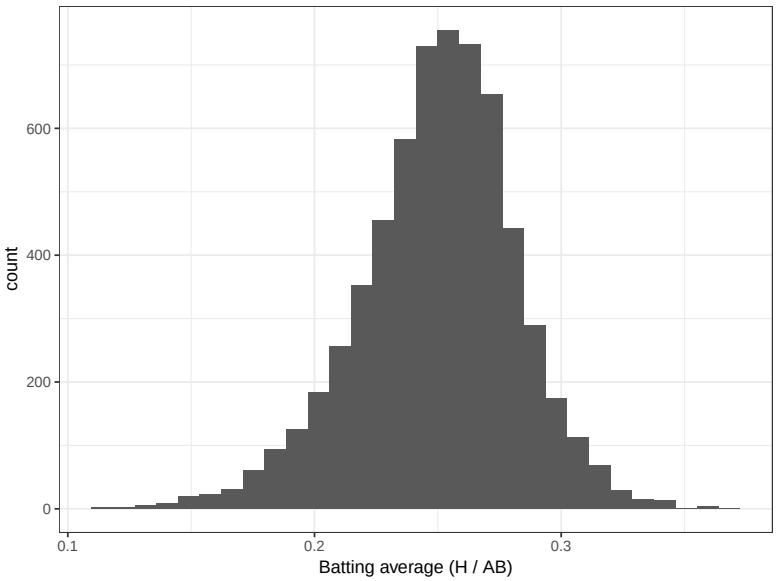

Figure 11.1: Distribution of player batting averages.

In Chapter 3, we noticed that the distribution of player batting averages looked roughly like a beta distribution (Figure 11.1). We thus wanted to estimate the beta prior for the overall dataset, which is the

first step of empirical Bayes analysis.

The `ebb_fit_prior()` function encapsulates this, taking the data along with the success/total columns and fitting the beta through maximum likelihood.

```
library(ebbr)

# filter for only players with more than 500 at-bats
prior <- career %>%
  filter(AB >= 500) %>%
  ebb_fit_prior(H, AB)

prior
```

```
## Empirical Bayes binomial fit with method mle
## Parameters:
## # A tibble: 1 × 2
##   alpha  beta
##   <dbl> <dbl>
## 1  96.9   275
```

`prior` is an `ebb_prior` object, which is a statistical model object containing the details of the beta distribution. Here we can see the overall alpha and beta parameters. (We'll see later that it can store more complicated models).

The second step of empirical Bayes analysis is updating each observation based on the overall statistical model. Based on the philosophy of the broom package, this is achieved with the `augment()` function.

```
augment(prior, data = career)
```

```
## # A tibble: 9,548 × 10
##     playerID     H    AB           name
##        <chr> <int> <int>          <chr>
## 1  aaronha01  3771 12364     Hank Aaron
## 2  aaronto01   216   944   Tommie Aaron
## 3   abadan01     2    21      Andy Abad
## 4  abadijo01    11    49    John Abadie
## 5  abbated01   772  3044 Ed Abbaticchio
## 6  abbeych01   492  1751  Charlie Abbey
## 7  abbotda01     1     7     Dan Abbott
## 8  abbotfr01   107   513    Fred Abbott
## 9  abbotje01   157   596    Jeff Abbott
## 10 abbotku01   523  2044    Kurt Abbott
## # ... with 9,538 more rows, and 6 more
## #   variables: .alpha1 <dbl>, .beta1 <dbl>,
```

```
## #   .fitted <dbl>, .raw <dbl>, .low <dbl>,
## #   .high <dbl>
```

Notice we've now added several columns to the original data, each beginning with `.` (which is a convention of the `augment` verb to avoid rewriting existing columns). We have the `.alpha1` and `.beta1` columns as the parameters for each player's posterior distribution, as well as `.fitted` representing the new posterior mean (the "shrunken average").

We often want to run these two steps in sequence: estimating a model, then using it as a prior for each observation. The `ebbr` package provides a shortcut, combining them into one step with `add_ebb_estimate()`.[2]

[2] The `add_` prefix is inspired by `add_residuals` and `add_predictions` in the modelr package, which also fit a statistical model and append columns based on it.

```
eb_career <- career %>%
  add_ebb_estimate(H, AB, prior_subset = AB >= 500)

eb_career
```

```
## # A tibble: 9,548 × 10
##     playerID     H    AB           name
##        <chr> <int> <int>          <chr>
## 1  aaronha01  3771 12364     Hank Aaron
## 2  aaronto01   216   944   Tommie Aaron
## 3   abadan01     2    21      Andy Abad
## 4  abadijo01    11    49    John Abadie
## 5  abbated01   772  3044 Ed Abbaticchio
## 6  abbeych01   492  1751  Charlie Abbey
## 7  abbotda01     1     7     Dan Abbott
## 8  abbotfr01   107   513    Fred Abbott
## 9  abbotje01   157   596    Jeff Abbott
## 10 abbotku01   523  2044    Kurt Abbott
## # ... with 9,538 more rows, and 6 more
## #   variables: .alpha1 <dbl>, .beta1 <dbl>,
## #   .fitted <dbl>, .raw <dbl>, .low <dbl>,
## #   .high <dbl>
```

Note the `prior_subset` argument, which noted that while we wanted to keep all the shrunken values in our output, we wanted to fit the prior only on individuals with at least 500 at-bats.

11.2.1 Estimates and credible intervals

Having the posterior estimates for each player lets us explore the model results using our normal tidy tools like dplyr and ggplot2. For example, we could visualize how batting averages were shrunken towards the mean of the prior (Figure 11.2).

```
eb_career %>%
  ggplot(aes(.raw, .fitted, color = AB)) +
  geom_point() +
  geom_abline(color = "red") +
  scale_color_continuous(trans = "log", breaks = c(1, 10, 100, 1000)) +
  geom_hline(yintercept = tidy(prior)$mean, color = "red", lty = 2) +
  labs(x = "Raw batting average",
       y = "Shrunken batting average")
```

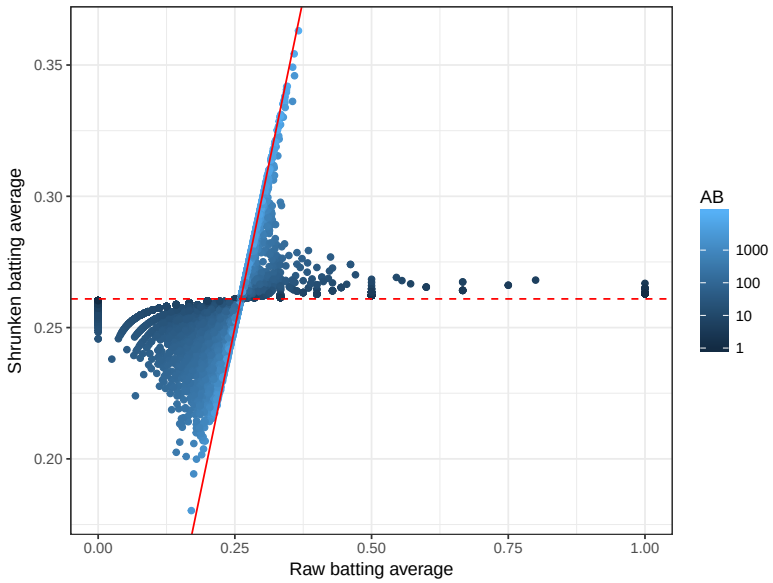

Figure 11.2: Comparison of the raw batting average and the empirical Bayes shrunken batting average.

This was one of the most important visualizations in Chapter 3. I like how it captures what empirical Bayes estimation is doing: moving all batting averages towards the prior mean (the dashed red line), but moving them less if there is a lot of information about that player (high AB).

In Chapter 12.3 we used credible intervals to visualize our uncertainty about each player's true batting average. The output of `add_ebb_estimate()` comes with those credible intervals in the form of `.low` and `.high` columns. This makes it easy to visualize these intervals for particular players, such as the 1998 Yankees (Figure 11.3).

```
yankee_1998 <- c("brosisc01", "jeterde01", "knoblch01",
                 "martiti02", "posadjo01", "strawda01", "willibe02")

eb_career %>%
  filter(playerID %in% yankee_1998) %>%
  mutate(name = reorder(name, .fitted)) %>%
  ggplot(aes(.fitted, name)) +
```

```
  geom_point() +
  geom_errorbarh(aes(xmin = .low, xmax = .high)) +
  labs(x = "Estimated batting average (w/ 95% confidence interval)",
       y = "Player")
```

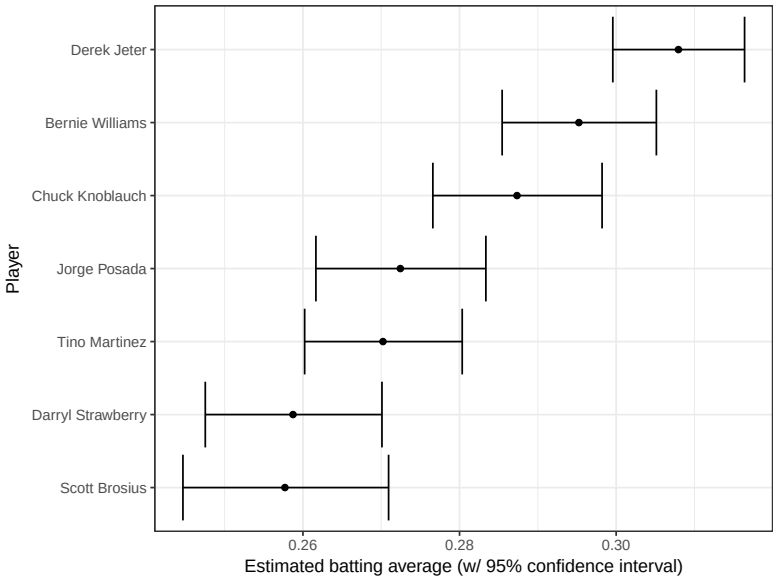

Figure 11.3: Credible intervals for seven selected players from the 1998 Yankee team.

Notice that once we have the output from `add_ebb_estimate()`, we're no longer relying on the `ebbr` package, only on dplyr and ggplot2.

11.3 Hierarchical modeling

In Chapters 7 and 11.3, we examined how this beta-binomial model may not be appropriate, because of the relationship between a player's at-bats and their batting average. Good batters tend to have long careers, while poor batters may retire quickly.

```
career %>%
  filter(AB >= 10) %>%
  ggplot(aes(AB, H / AB)) +
  geom_point() +
  geom_smooth(method = "lm") +
  scale_x_log10()
```

We solved this by fitting a prior that depended on AB, through the process of **beta-binomial regression**. The `add_ebb_estimate()` function from ebbr offers this option, by setting `method = "gamlss"` and providing a formula to `mu_predictors`.[3]

[3] You can also provide `sigma_predictors` to have the variance of the beta depend on regressors, though it's less common that you'd want to do so.

```
eb_career_ab <- career %>%
  add_ebb_estimate(H, AB, method = "gamlss",
                   mu_predictors = ~ log10(AB))

eb_career_ab
```

```
## # A tibble: 9,548 × 14
##     playerID     H    AB           name
##        <chr> <int> <int>          <chr>
## 1  aaronha01  3771 12364     Hank Aaron
## 2  aaronto01   216   944   Tommie Aaron
## 3   abadan01     2    21      Andy Abad
## 4  abadijo01    11    49    John Abadie
## 5  abbated01   772  3044 Ed Abbaticchio
## 6  abbeych01   492  1751  Charlie Abbey
## 7  abbotda01     1     7     Dan Abbott
## 8  abbotfr01   107   513    Fred Abbott
## 9  abbotje01   157   596    Jeff Abbott
## 10 abbotku01   523  2044    Kurt Abbott
## # ... with 9,538 more rows, and 10 more
## #   variables: .mu <dbl>, .sigma <dbl>,
## #   .alpha0 <dbl>, .beta0 <dbl>,
## #   .alpha1 <dbl>, .beta1 <dbl>,
## #   .fitted <dbl>, .raw <dbl>, .low <dbl>,
## #   .high <dbl>
```

The augmented output is now a bit more complicated (and hard to fit into a printed output). Besides the posterior parameters such as `.alpha1`, `.beta1`, and `.fitted`, each observation also has its own prior parameters `.alpha0` and `.beta0`. These are predicted based on the regression on `AB`.

The other parameters, such as `.fitted` and the credible interval, are now shrinking towards a *trend* rather than towards a constant. We can see this by plotting `AB` against the raw batting averages and the shrunken estimates (Figure 11.4).

```
eb_career_ab %>%
  filter(AB > 10) %>%
  rename(Raw = .raw, Shrunken = .fitted) %>%
  gather(type, estimate, Raw, Shrunken) %>%
  ggplot(aes(AB, estimate)) +
  geom_point() +
  facet_wrap(~ type) +
  scale_x_log10()
```

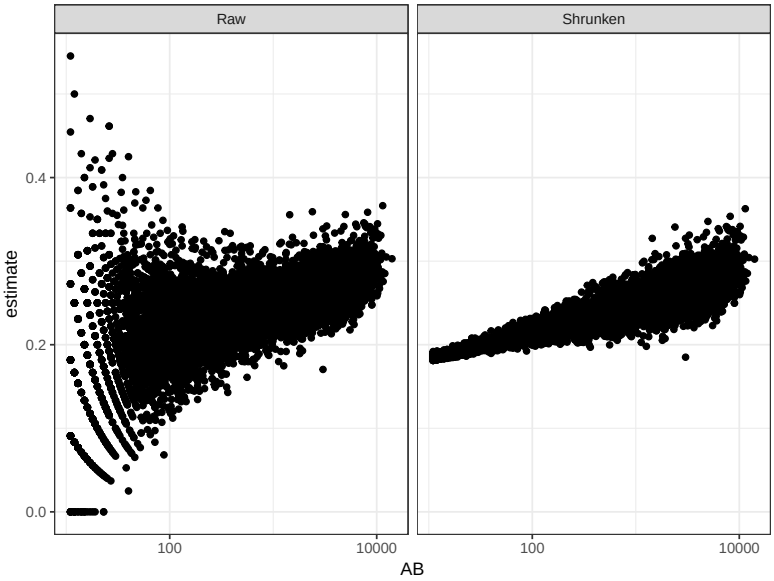

Figure 11.4: The relationship between AB and the raw estimate, as well as with and the shrunken empirical Bayes estimate using a beta-binomial regression model.

As we saw in the Chapter 11.3, the model's `mu_predictors` argument can incorporate still more useful information. For example, it could include what year each player batted in, and whether they are left- or right- handed, both of which tend to affect batting average.

```
library(splines)

eb_career_prior <- career_full %>%
  ebb_fit_prior(H, AB, method = "gamlss",
                mu_predictors = ~ 0 + ns(year, df = 5) * bats + log(AB))
```

In this case we're fitting the prior with `ebb_fit_prior()` rather than adding the estimates with `add_ebb_estimate()`. This lets us feed it new data that we generate ourselves, and examine how the posterior distribution would change. This takes a bit more work, but lets us re-generate one of the more interesting plots from that post about how time and handedness relate (Figure 11.5).

```
# fake data ranging from 1885 to 2013
fake_data <- crossing(H = 300,
                      AB = 1000,
                      year = seq(1885, 2013),
                      bats = c("L", "R"))

# find the mean of the prior, as well as the 95% quantiles,
# for each of these combinations. This does require a bit of
# manual manipulation of alpha0 and beta0:
augment(eb_career_prior, newdata = fake_data) %>%
```

```
  mutate(prior = .alpha0 / (.alpha0 + .beta0),
         prior.low = qbeta(.025, .alpha0, .beta0),
         prior.high = qbeta(.975, .alpha0, .beta0)) %>%
  ggplot(aes(year, prior, color = bats)) +
  geom_line() +
  geom_ribbon(aes(ymin = prior.low, ymax = prior.high), alpha = .1, lty = 2) +
  ylab("Prior distribution (mean + 95% quantiles)")
```

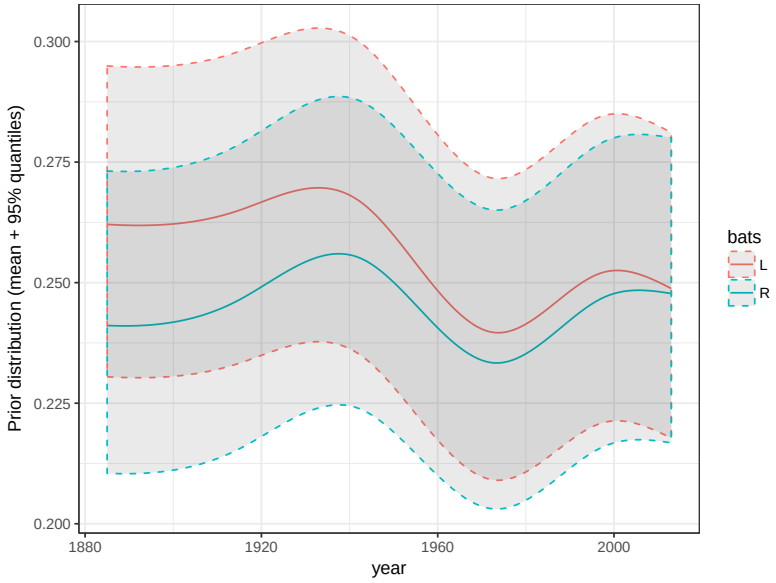

Figure 11.5: The prior distribution that would be used for a left- or right-handed player at a particular point in time. Shown are the mean and the 95% intervals for each prior.

Since the `ebbr` package makes these models convenient to fit, you can try a few variations on the model and compare them.

11.4 Hypothesis testing

Chapter 11.4 examined the problem of hypothesis testing. For example, we wanted to get a posterior probability for the statement "this player's true batting average is greater than .300", so that we could construct a "Hall of Fame" of such players.

This method is implemented in the `add_ebb_prop_test` (notice that like `add_ebb_estimate`, it adds columns to existing data). `add_ebb_prop_test` takes the output of an earlier `add_ebb_estimate` operation, which contains posterior parameters for each observation, and appends columns to it:

```
test_300 <- career %>%
  add_ebb_estimate(H, AB, method = "gamlss", mu_predictors = ~ log10(AB)) %>%
  add_ebb_prop_test(.300, sort = TRUE)

test_300
```

```
## # A tibble: 9,548 × 16
##     playerID     H    AB
##        <chr> <int> <int>
## 1   cobbty01  4189 11434
## 2  hornsro01  2930  8173
## 3  speaktr01  3514 10195
## 4  delahed01  2596  7505
## 5  willite01  2654  7706
## 6  keelewi01  2932  8591
## 7  lajoina01  3242  9589
## 8  jacksjo01  1772  4981
## 9  gwynnto01  3141  9288
## 10 heilmha01  2660  7787
## # ... with 9,538 more rows, and 13 more
## #   variables: name <chr>, .mu <dbl>,
## #   .sigma <dbl>, .alpha0 <dbl>,
## #   .beta0 <dbl>, .alpha1 <dbl>,
## #   .beta1 <dbl>, .fitted <dbl>, .raw <dbl>,
## #   .low <dbl>, .high <dbl>, .pep <dbl>,
## #   .qvalue <dbl>
```

(Note the `sort = TRUE` argument, which sorts in order of our confidence in each player). There are now too many columns to read easily, so we'll select only a few of the most interesting ones:

```
test_300 %>%
  select(name, H, AB, .fitted, .low, .high, .pep, .qvalue)
```

```
## # A tibble: 9,548 × 8
##                    name     H    AB .fitted
##                   <chr> <int> <int>   <dbl>
## 1              Ty Cobb  4189 11434   0.363
## 2        Rogers Hornsby  2930  8173   0.354
## 3          Tris Speaker  3514 10195   0.342
## 4          Ed Delahanty  2596  7505   0.341
## 5          Ted Williams  2654  7706   0.340
## 6         Willie Keeler  2932  8591   0.338
## 7            Nap Lajoie  3242  9589   0.335
## 8  Shoeless Joe Jackson  1772  4981   0.348
## 9            Tony Gwynn  3141  9288   0.335
## 10       Harry Heilmann  2660  7787   0.338
## # ... with 9,538 more rows, and 4 more
## #   variables: .low <dbl>, .high <dbl>,
## #   .pep <dbl>, .qvalue <dbl>
```

Notice that two columns have been added, with per-player metrics

described in this post.

- `.pep`: the posterior error probability- the probability that this player's true batting average is less than .3.
- `.qvalue`: the q-value, which corrects for multiple testing by controlling for false discovery rate (FDR). Allowing players with a q-value below .05 would mean only 5% of the ones included would be false discoveries.

For example, we could find how many players would be added to our "Hall of Fame" with an FDR of 5%, or 1%:

```
sum(test_300$.qvalue < .05)
```

```
## [1] 115
```

```
sum(test_300$.qvalue < .01)
```

```
## [1] 79
```

11.4.1 Player-player A/B test

Chapter 6 discussed the case where instead of comparing each observation to a single threshold (like .300) we want to compare to another player's posterior distribution. We noted that this is similar to the problem of "A/B testing", where we might be comparing two click-through rates, each represented by successes / total.

The post compared each player in history to Mike Piazza, and found players we were confident were better batters. We'd first find Piazza's posterior parameters:

```
piazza <- eb_career_ab %>%
  filter(name == "Mike Piazza")

piazza_params <- c(piazza$.alpha1, piazza$.beta1)
piazza_params
```

```
## [1] 2281 5183
```

This vector of two parameters, an alpha and a beta, can be passed into `add_ebb_prop_test` just like we passed in a threshold.[4]

```
compare_piazza <- eb_career_ab %>%
  add_ebb_prop_test(piazza_params, approx = TRUE, sort = TRUE)
```

Again we select only a few interesting columns to display.

```
compare_piazza %>%
  select(name, H, AB, .fitted, .low, .high, .pep, .qvalue)
```

[4] This is rather similar to how the built-in prop.test function handles the difference between one-sample and two-sample tests. I've often found this kind of behavior annoying (determining what test to use based on the length of the input), but I must admit it is convenient.

```
## # A tibble: 9,548 × 8
##                    name     H    AB .fitted
##                   <chr> <int> <int>   <dbl>
## 1              Ty Cobb  4189 11434   0.363
## 2        Rogers Hornsby  2930  8173   0.354
## 3          Tris Speaker  3514 10195   0.342
## 4  Shoeless Joe Jackson  1772  4981   0.348
## 5          Ed Delahanty  2596  7505   0.341
## 6          Ted Williams  2654  7706   0.340
## 7         Willie Keeler  2932  8591   0.338
## 8        Harry Heilmann  2660  7787   0.338
## 9        Billy Hamilton  2158  6268   0.339
## 10           Nap Lajoie  3242  9589   0.335
## # ... with 9,538 more rows, and 4 more
## #   variables: .low <dbl>, .high <dbl>,
## #   .pep <dbl>, .qvalue <dbl>
```

Just like the one-sample test, the function has added `.pep` and `.qvalue` columns. From this we can see a few players who we're extremely confident are better than Piazza.

11.5 Mixture models

This brings us to mixture models. In Chapter 11.5 we noticed that when pitchers are included, the data looks a lot less like a beta distribution and more like a combination of two betas. Fitting a mixture model, to separate out the two beta distributions so they could be shrunken separately, took a solid amount of code in that chapter.

```
career_w_pitchers <- Batting %>%
  filter(AB >= 25, lgID == "NL", yearID >= 1980) %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB), year = mean(yearID)) %>%
  mutate(isPitcher = playerID %in% pitchers$playerID) %>%
  inner_join(player_names, by = "playerID")
```

The ebbr package thus provides the `ebb_fit_mixture()` function for fitting a mixture model using an iterative expectation-maximization algorithm. Like the other estimation functions, it takes a table as the first argument, followed by two arguments for the "successes" column and the "total" column:

```
set.seed(1337)
mm <- ebb_fit_mixture(career_w_pitchers, H, AB, clusters = 2)
```

It returns the parameters of two (or more) beta distributions, which can be extracted with the `tidy()` function.

```
tidy(mm)
```

```
## # A tibble: 2 × 6
##   cluster alpha  beta  mean number
##     <chr> <dbl> <dbl> <dbl>  <int>
## 1       2 161.2   460 0.259   1975
## 2       1  25.8   152 0.145    916
## # ... with 1 more variables:
## #   probability <dbl>
```

It also assigns each observation to the most likely cluster. Here, we can see that cluster 1 is made up of pitchers, while cluster 2 is the non-pitchers (Figure 11.6).

```
ggplot(mm$assignments, aes(H / AB, fill = .cluster)) +
  geom_histogram(alpha = 0.8, position = "identity")
```

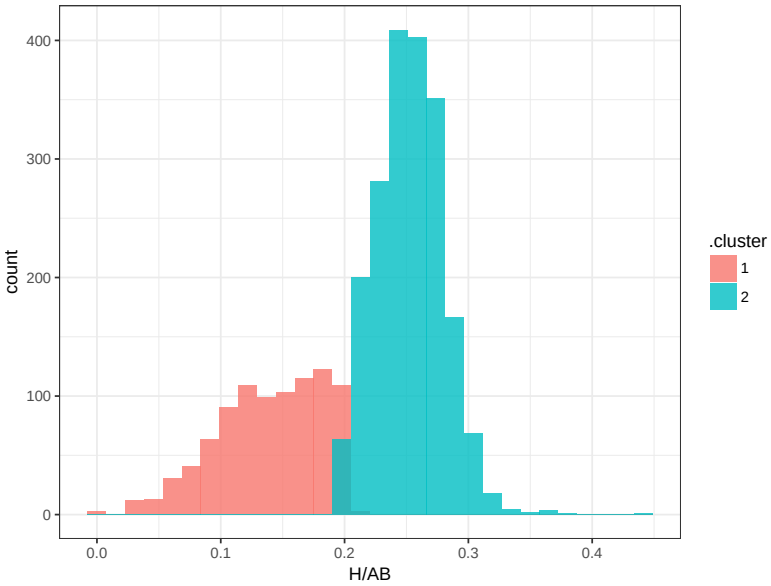

Figure 11.6: Histogram of mixture model assignments to the two clusters.

11.5.1 Mixture model across iterations

You may be interested in how the mixture model converged to its parameters. The `iterations` component of the `ebb_mixture` object contains details on the fits and assignments at each iteration. For example, we could examine the distribution of players assigned to each cluster at each iteration (Figure 11.7).

```
assignments <- mm$iterations$assignments
```

```
assignments %>%
```

```
  ggplot(aes(H / AB, fill = .cluster)) +
  geom_histogram(alpha = 0.8, position = "identity") +
  facet_wrap(~ iteration)
```

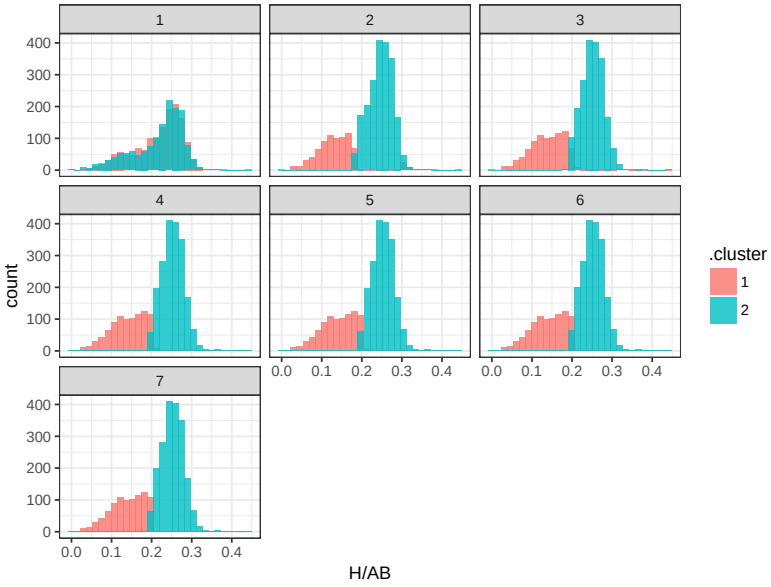

Figure 11.7: Histogram of the batting averages of players assigned to clusters A and B,

The `iterations` component also contains details on the parameters fit at each of the maximization steps, which can then be visualized to see how those parameters converged (Figure 11.8).

```
fits <- mm$iterations$fits

fits %>%
  gather(parameter, value, alpha, beta, mean) %>%
  ggplot(aes(iteration, value, color = parameter, lty = cluster)) +
  geom_line() +
  facet_wrap(~ parameter, scales = "free_y")
```

The ebbr package's functions for mixture modeling are currently a bit more primitive than the rest, and some of the details are likely to change (this is one of the reasons ebbr hasn't yet been submitted to CRAN). Still, the package makes it easier to experiment with expectation-maximization algorithms.

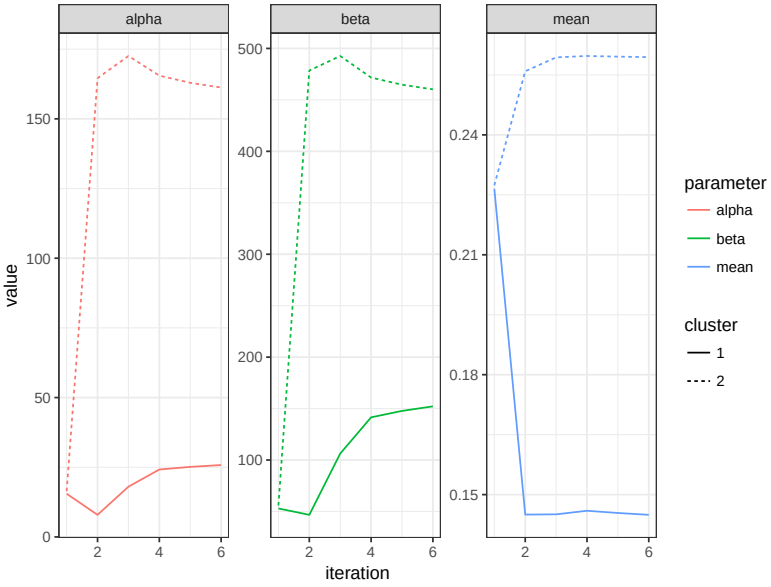

Figure 11.8: The estimated α and β parameters for each cluster at each iteration of the expectation-maximization algorithm.

12
Simulation

Over the course of this book we've touched on many statistical approaches for analyzing binomial data, all with the goal of estimating the batting average of each player based on their batting record. There's one question we haven't answered, though: **do these methods actually work?**

Even if we assume each player has a "true" batting average (as our model suggests), we don't *know* it, so we can't see if our methods estimated it accurately. For example, we think that empirical Bayes shrinkage gets closer to the true probabilities than raw batting averages do, but we can't actually measure the mean-squared error. This means we can't test our methods, or examine when they work well and when they don't.

In this last chapter we'll **simulate** some fake batting average data, which will let us know the true probabilities for each player, then examine how close our statistical methods get to the true solution. Simulation is a universally useful way to test a statistical method, to build intuition about its mathematical properies, and to gain confidence that we can trust its results. In particular, we'll demonstrate the tidyverse approach to simulation, which takes advantage of packages such as dplyr, tidyr, purrr and broom to examine many combinations of input parameters.

12.1 Setup

Most of the chapters started by assembling some per-player batting data. We're going to be simulating (i.e. making up) our data for this analysis, so you might think we don't need to look at real data at all. However, data is still necessary to estimating the parameters we'll use in the simulation, which keeps the experiment realistic and ensures that our conclusions will be useful.

```
library(Lahman)
library(dplyr)
library(tidyr)
library(purrr)

# Grab career batting average of non-pitchers
# (allow players that have pitched <= 3 games, like Ty Cobb)
pitchers <- Pitching %>%
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
  filter(gamesPitched > 3)

# include the "bats" (handedness) and "year" column for later
career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB))
```

12.1.1 Distribution of p and AB

The first step to developing a simulation study is specifying a generative model: describing what distributions each value follows. In the beta-binomial model we've been using for most of these posts, there are two values for each player i:

$$p_i \sim \text{Beta}(\alpha_0, \beta_0)$$

$$H_i \sim \text{Binom}(\text{AB}_i, p_i)$$

$\alpha_0; \beta_0$ are "hyperparameters": two unobserved values that describe the entire distribution. p_i is the true batting average for each player- we don't observe this, but it's the "right answer" for each batter that we're trying to estimate. AB_i is the number of at-bats the player had, which *is* observed.[1]

Our simulation approach is going to be to pick some "true" $\alpha_0; \beta_0$, then simulate p_i for each player. Since we're just picking any $\alpha_0; \beta_0$ to start with, we may as well estimate them from our data, since we know those are plausible values (though if we wanted to be more thorough, we could try a few other values and see how our accuracy changes).

To do this estimation, we can use `ebb_fit_prior` from the ebbr package introduced in Chapter 11 to fit the empirical Bayes prior.

[1] You might recall we introduced a more complicated model in Chapter 7 that had p_i depend on AB_i, which we'll revisit later in this chapter.

```
library(ebbr)
prior <- career %>%
```

```
  ebb_fit_prior(H, AB)

prior
```

```
## Empirical Bayes binomial fit with method mle
## Parameters:
## # A tibble: 1 × 2
##   alpha  beta
##   <dbl> <dbl>
## 1 72.13 215.1
```

These two hyperparameters are all we need to simulate 10,000 values of p_i, using the `rbeta` function.

```
alpha0 <- tidy(prior)$alpha
beta0 <- tidy(prior)$beta

# for example, generating 10 values
rbeta(10, alpha0, beta0)
```

```
##  [1] 0.25249 0.23077 0.24909 0.22659 0.23812
##  [6] 0.26877 0.20888 0.23660 0.26017 0.25646
```

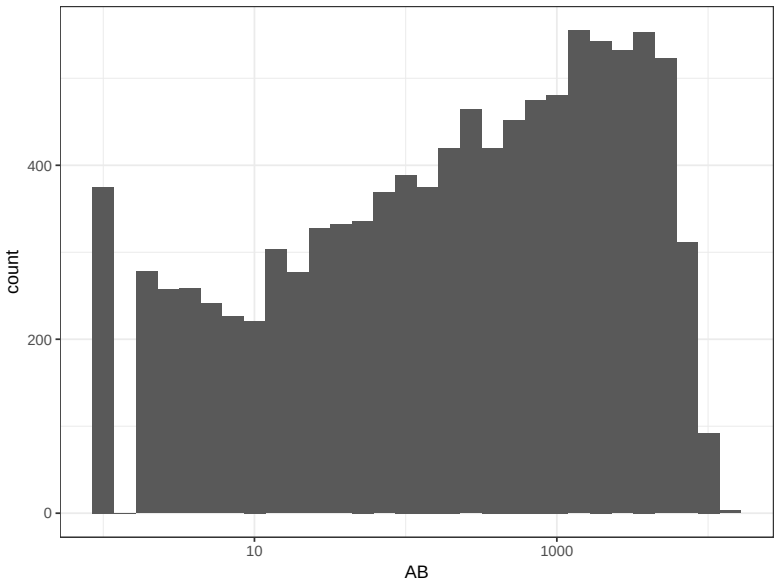

Figure 12.1: Distribution of AB, the number of at-bats, across all players (note the log x axis).

There's another component to this model: AB_i, the distribution of the number of at-bats. This is a much more unusual distribution (Figure 12.1; code not shown). The good news is, we don't *need* to simulate these AB_i values, since we're not trying to estimate them with empirical Bayes. We can just use the observed values we have!

(In a different study, we may be interested in how the success of empirical Bayes depends on the distribution of the ns).

Thus, to recap, we will:

- **Estimate** $\alpha_0; \beta_0$, which works because the parameters are not observed, but there are only a few and we can predict them with confidence.
- **Simulate** p_i, based on a beta distribution, so that we can test our ability to estimate them.
- **Use observed** AB_i, since we know the true values and there's no harm in re-using them.

12.2 Empirical Bayes estimation

The beta-binomial model is easy to simulate, with applications of the `rbeta` and `rbinom` functions.

```
# always set a seed when simulating
set.seed(2017)

career_sim <- career %>%
  mutate(p = rbeta(n(), alpha0, beta0),
         H = rbinom(n(), AB, p))

career_sim
```

```
## # A tibble: 10,388 × 4
##     playerID     H    AB       p
##        <chr> <int> <int>   <dbl>
## 1  aaronha01  3817 12364 0.29908
## 2  aaronto01   240   944 0.24889
## 3   abadan01     5    21 0.27364
## 4  abadijo01     9    49 0.19774
## 5  abbated01   755  3044 0.24911
## 6  abbeych01   449  1751 0.26453
## 7  abbotda01     2     7 0.19075
## 8  abbotfr01   138   513 0.25108
## 9  abbotje01   116   596 0.24351
## 10 abbotku01   570  2044 0.26121
## # ... with 10,378 more rows
```

Just like that, we've generated a "true" p_i for each player from the beta distribution, and then a new value of H from the binomial distribution according to that probability.[2]

Throughout this book, we've assumed that the raw H/AB estimates have had a large amount of noise when AB is small, and that

[2] Note that this means there is no relationship between how good a particular player is in our simulation and how good they are in reality.

empirical Bayes helps reduce that noise. Now, since we know the true value of p_i for each player, we can finally examine whether that's true: and we can see what the empirical Bayes method is giving up as well.

Let's visualize the true values of p_i versus the estimates, which we'll call $\hat{p}_i$, using either raw estimation or empirical Bayes shrinkage (Figure 12.2).

```
career_sim_eb <- career_sim %>%
  add_ebb_estimate(H, AB)

career_sim_gathered <- career_sim_eb %>%
  rename(Shrunken = .fitted, Raw = .raw) %>%
  gather(type, estimate, Shrunken, Raw)

career_sim_gathered %>%
  filter(AB >= 10) %>%
  ggplot(aes(p, estimate, color = AB)) +
  geom_point() +
  geom_abline(color = "red") +
  geom_smooth(method = "lm", color = "white", lty = 2, se = FALSE) +
  scale_color_continuous(trans = "log",
                         breaks = c(10, 100, 1000, 10000)) +
  facet_wrap(~ type) +
  labs(x = "True batting average (p)",
       y = "Raw or shrunken batting average")
```

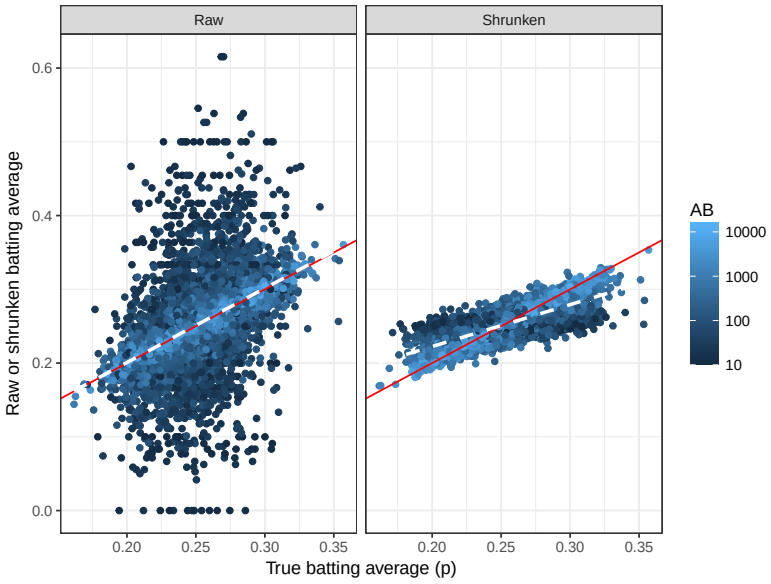

Figure 12.2: The relationship between the true batting average p and either the raw batting average H/AB, or the shrunken estimate $\hat{p}_i$.

This figure shows that the method works: the raw (H / AB) estimates have a *lot* more noise than the shrunken estimates (appearing

farther from the red line), just as we expected.[3]

However, notice the white dashed line representing the best-fit slope. One property that we'd prefer an estimate to have is that it's equally likely to be an overestimate or an underestimate (that is, that $E[\hat{p}] = p$), and that's true for the raw batting average: the white dashed line lines up with the red $x = y$ line. However, the shrunken estimate tends to be too high for low values of p, and too low for high values of p. The empirical Bayes method has introduced **bias** into our estimate, in exchange for drastically reducing the **variance**. This is a classic tradeoff in statistics and machine learning.

[3] We filtered out cases where $AB < 10$ in this graph: if we hadn't, the difference would have been even starker

12.2.1 Mean-squared error and bias relative to AB

Typically, when statisticians are facing a tradeoff between bias and variance, we use **mean squared error** (MSE) as a balance, which is computed as $\text{MSE} = \frac{1}{n}\sum_1^n (p - \hat{p})^2$ (thus, the average squared distance from the truth). We can easily compute that for both the raw and shrunken methods.

```
career_sim_gathered %>%
  group_by(type) %>%
  summarize(mse = mean((estimate - p) ^ 2))
```

```
## # A tibble: 2 × 2
##       type        mse
##      <chr>      <dbl>
## 1      Raw 0.01515533
## 2 Shrunken 0.00034977
```

The MSE of the shrunken estimate was *much* lower than the raw estimate, as we probably could have guessed by eyeballing the graph. So by this standard, the method succeeded!

We've seen in Figure 12.2 how the variance depends on AB, so we may want to compute the MSE within particular bins (Figure 12.3).

```
metric_by_bin <- career_sim_gathered %>%
  group_by(type, AB = 10 ^ (round(log10(AB)))) %>%
  summarize(mse = mean((estimate - p) ^ 2))

ggplot(metric_by_bin, aes(AB, mse, color = type)) +
  geom_line() +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Number of at-bats (AB)",
       y = "Mean-squared-error within this bin")
```

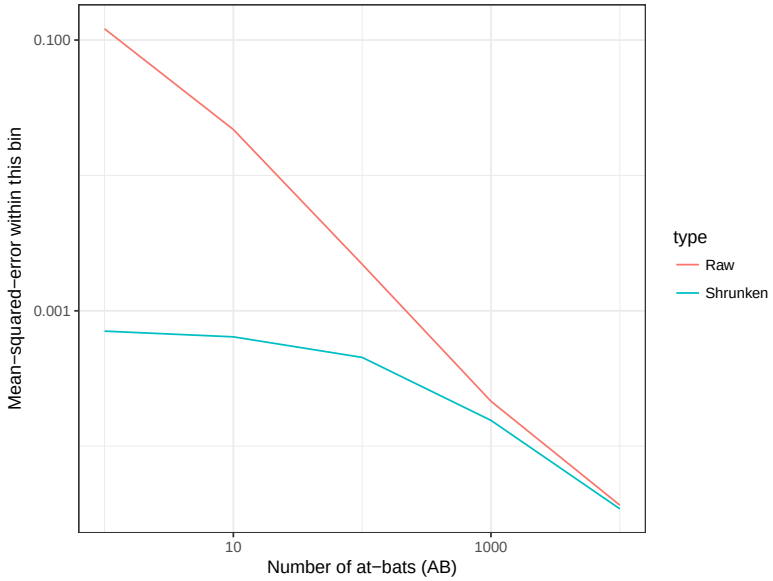

Figure 12.3: Mean-squared error within bins of AB, using either the raw average or the shrunken estimate. Note that both axes are on a log scale.

We note that the mean squared error is higher with the raw estimate, especially for low AB, and that the two methods become more and more similar in terms of MSE for higher AB. This makes sense, since when we have a large amount of evidence, both methods tend to be very accurate and empirical Bayes has only a small effect.

We could also examine the bias within each bin, measured as the slope between the estimate and the true value of p (Figure 12.4; code not shown). This shows that shrunken estimates introduce bias, especially when AB is low, while the raw estimates are generally unbiased.

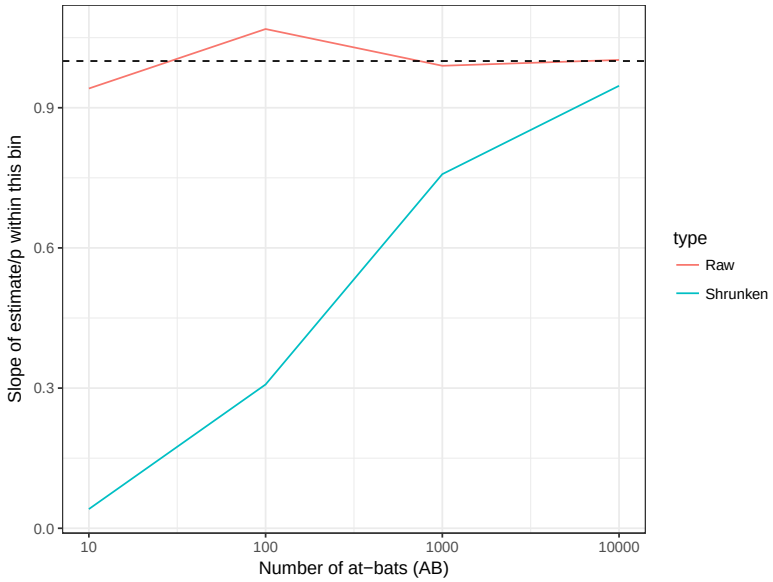

Figure 12.4: Bias within bins of AB, using either the raw average or the shrunken estimate. Note that an unbiased estimate would have a slope of 0 (shown as horizontal dashed line).

Another way to visualize how this tradeoff between bias and variance happens for varying AB is to recreate Figure 12.2 of the true batting average versus the estimate, this time binning by AB (Figure 12.5; code not shown).

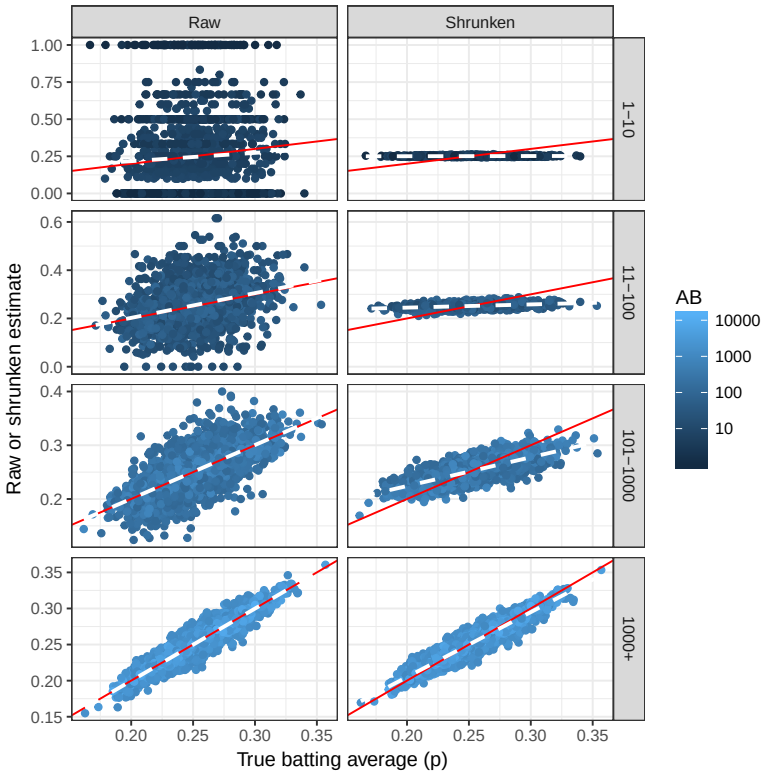

Figure 12.5: The relationship between the true batting average p and either the raw batting average H/AB, or the shrunken estimate $\hat{p}_i$, within particular bins of AB. The red line is x = y; the dashed white line is a linear fit.

Notice how the variance around the true (red) line shrinks in the raw estimate, and the bias (the flatness of the gray dashed line) in the shrunken estimate decreases, until both look quite similar in the 1000+ bin.

12.3 Credible intervals

Besides the shrunken empirical Bayes estimates, the `add_ebb_estimate` function also adds credible intervals (Chapter 12.3) for each of our players. With our simulated data, we now know the true batting averages, and can see whether the intervals are accurate in representing the uncertainty. For example, we could visualize the credible intervals for 20 random players, and consider how often the interval captured the true batting average (Figure 12.6).

```
set.seed(2017)

career_sim_eb %>%
```

```
  sample_n(20) %>%
  mutate(playerID = reorder(playerID, .fitted)) %>%
  ggplot(aes(.fitted, playerID)) +
  geom_point() +
  geom_point(aes(x = p), color = "red") +
  geom_errorbarh(aes(xmin = .low, xmax = .high)) +
  theme(axis.text.y = element_blank()) +
  labs(x = "Estimated batting average (w/ 95% credible interval)",
       y = "Player")
```

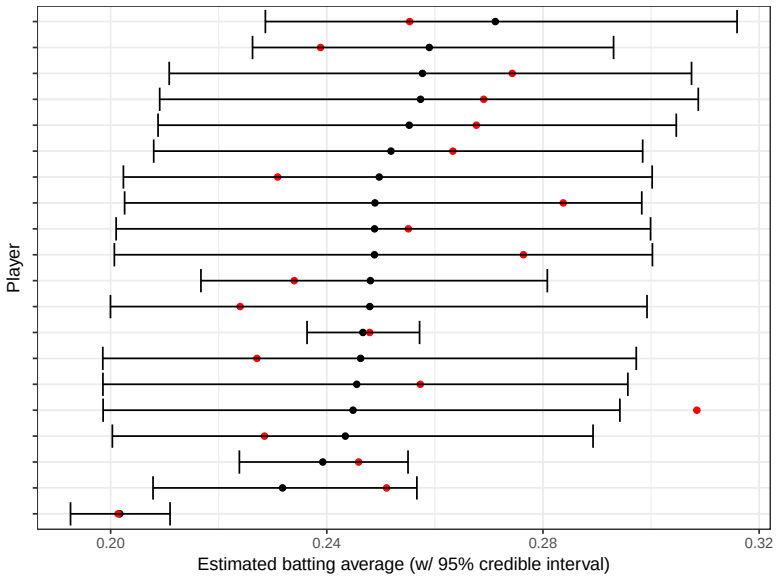

Figure 12.6: Credible intervals for 20 randomely selected players, with the true batting average of each player shown in red.

Notice that out of 20 randomly selected players, the credible interval contained the true batting average (shown in red) in 19 cases. This is a 95% coverage rate, which is exactly what we'd hoped for! Indeed, we can examine this across all players and see that 95% of the intervals contained the true probability.

```
career_sim_eb %>%
  summarize(coverage = mean(.low <= p & p <= .high))
```

```
## # A tibble: 1 × 1
##   coverage
##      <dbl>
## 1  0.94869
```

We could also have set the threshold of the credible interval to 90%, or 75%. Does the probability that the parameter is contained within the interval change accordingly along with the level? (Figure 12.7).

```
library(purrr)

# fit the prior once
sim_prior <- ebb_fit_prior(career_sim, H, AB)

# find the coverage probability for each level
estimate_by_cred_level <- data_frame(level = seq(.5, .98, .02)) %>%
  unnest(map(level, ~ augment(sim_prior, career_sim, cred_level = .)))


estimate_by_cred_level %>%
  group_by(level) %>%
  mutate(cover = .low <= p & p <= .high) %>%
  summarize(coverage = mean(cover)) %>%
  ggplot(aes(level, coverage)) +
  geom_line() +
  geom_abline(color = "red", lty = 2) +
  labs(x = "Level of credible interval",
       y = "Probability credible interval contains the true value")
```

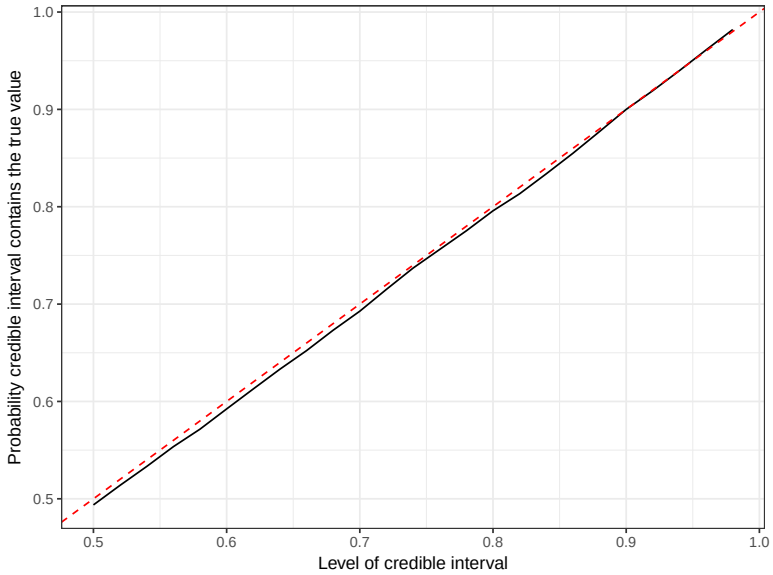

Figure 12.7: Comparison of the level of the credibility interval to the percentage of players where the credible interval contains the true value.

Notice that the probability (the points) hugs the red $x = y$ line almost precisely. This shows that in this method, the per-observation credible intervals are generally *well-calibrated*: if you ask for a X% credible interval, you get a region that contains the true parameter about X% of the time.

12.4 FDR control

In Chapter 11.4 we examined the problem of Bayesian hypothesis testing and FDR control. In particular, we considered the problem of constructing a list of players whose true batting average was above .300, and controlling such that only (say) 10% of the players on the list were included incorrectly.

The q-value, which controls FDR, can be calculated with the `add_ebb_prop_test` function:

```
pt <- career_sim_eb %>%
  add_ebb_prop_test(.3, sort = TRUE)

# Control for FDR of 10%
hall_of_fame <- pt %>%
  filter(.qvalue <= .1)
nrow(hall_of_fame)
```

```
## [1] 74
```

If the FDR control were successful, we'd expect 10% of the true batting averages (`p`) to be false discoveries, and therefore below .300. Did the method work?

```
mean(hall_of_fame$p < .3)
```

```
## [1] 0.094595
```

Yes- almost exactly 10% of the players included in this "hall of fame" were included incorrectly, indicating that the q-value succeeded in controlling FDR. We could instead try this for all q-values, comparing each q-value threshold to the resulting fraction of false discoveries (Figure 12.8).

```
pt %>%
  mutate(true_fdr = cummean(p < .3)) %>%
  ggplot(aes(.qvalue, true_fdr)) +
  geom_line() +
  geom_abline(color = "red") +
  labs(x = "q-value threshold",
       y = "True FDR at this q-value threshold")
```

Notice that the FDR was often a little bit higher than we aimed for with the q-value, which could be due to random noise. Later in this chapter, we'll perform many replications of this simulation and confirm whether the FDR method was successful on average.

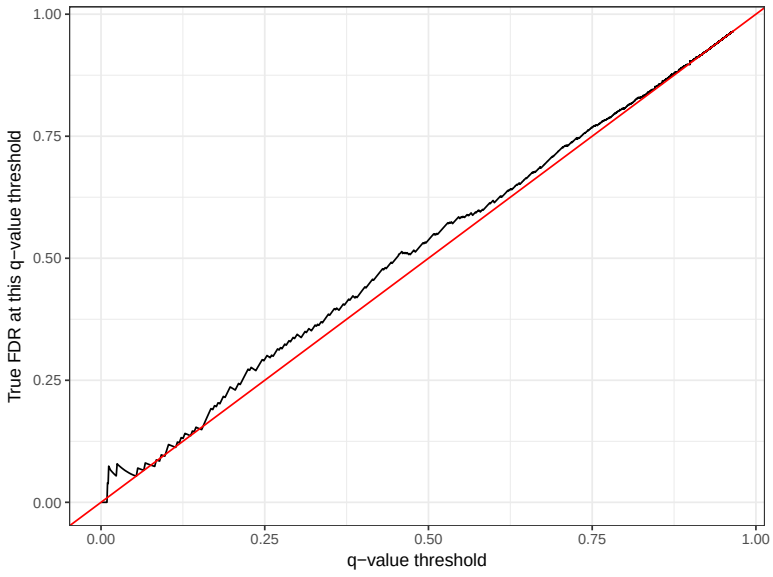

Figure 12.8: Comparison of the q-value threshold, meant to control false discovery rate, and the true FDR, defined as the number of players included where $p < 3$. The red line is x = y.

12.5 Beta-binomial regression

Most simulation analyses start with a simple model, than gradually add complications. In Chapter 7, we discovered that there is a relationship between AB_i and the true batting average p_i that we need to incorporate into our model. Let's add that complication to our simulation, and see if the method we used to account for it actually works.

The model described in that post had three hyperparameters: μ_0, μ_{AB} and σ_0. Then each of the probabilities p_i was computed according to the following generative process.

$$\mu_i = \mu_0 + \mu_{\text{AB}} \cdot \log(\text{AB})$$

$$\alpha_{0,i} = \mu_i / \sigma_0$$

$$\beta_{0,i} = (1 - \mu_i) / \sigma_0$$

$$p_i \sim \text{Beta}(\alpha_{0,i}, \beta_{0,i})$$

$$H_i \sim \text{Binom}(\text{AB}_i, p_i)$$

Much as we estimated α_0 and β_0 from the data before using them in the simulation, we would estimate μ_0, μ_{AB}, and σ_0 from the data:

```
bb_reg <- career %>%
  ebb_fit_prior(H, AB, method = "gamlss", mu_predictors = ~ log10(AB))
```

```
tidy(bb_reg)
```

```
## # A tibble: 3 × 6
##   parameter        term estimate std.error
##      <fctr>       <chr>    <dbl>     <dbl>
## 1        mu (Intercept) -1.68767 0.0090698
## 2        mu   log10(AB)  0.19232 0.0028014
## 3     sigma (Intercept) -6.29919 0.0231563
## # ... with 2 more variables:
## #   statistic <dbl>, p.value <dbl>
```

The simulation of p for each player is pretty straightforward with the `augment()` method of the beta-binomial prior. Since `augment()` adds `.alpha0` and `.beta0` columns with the per-player hyperparameters, we can simply generate `p` from that.

```
set.seed(2017)

career_sim_ab <- augment(bb_reg, career) %>%
  select(playerID, AB,
         true_alpha0 = .alpha0,
         true_beta0 = .beta0) %>%
  mutate(p = rbeta(n(), true_alpha0, true_beta0),
         H = rbinom(n(), AB, p))
```

12.5.1 Performance of beta-binomial regression method

We first might be wondering if we were able to extract the right hyperparameters through beta-binomial regression. To answer this, we'll fit the prior and then compare it to `bb_reg`, which were the parameters we used to generate the data.

```
career_ab_prior <- career_sim_ab %>%
  ebb_fit_prior(H, AB, method = "gamlss", mu_predictors = ~ log10(AB))

tidy(bb_reg)
```

```
## # A tibble: 3 × 6
##   parameter        term estimate std.error
##      <fctr>       <chr>    <dbl>     <dbl>
## 1        mu (Intercept) -1.68767 0.0090698
## 2        mu   log10(AB)  0.19232 0.0028014
## 3     sigma (Intercept) -6.29919 0.0231563
## # ... with 2 more variables:
## #   statistic <dbl>, p.value <dbl>
```

```
tidy(career_ab_prior)
```

```
## # A tibble: 3 × 6
##   parameter        term estimate std.error
##      <fctr>       <chr>    <dbl>     <dbl>
## 1        mu (Intercept) -1.68844 0.0090827
## 2        mu   log10(AB)  0.19285 0.0028071
## 3     sigma (Intercept) -6.27911 0.0256424
## # ... with 2 more variables:
## #   statistic <dbl>, p.value <dbl>
```

That's quite close! It looks like beta-binomial regression was able to estimate the true parameters accurately,[4] which suggests the resulting per-player prior (which depends on both those parameters AB_i) will be accurate.

[4] It makes sense that the estimation was as accurate as it was, since the standard error (`std.error`) representing the uncertainty is quite low.

How did this prior affect our shrunken estimates? Again, since we're working from a simulation we can compare the true values to the estimates, and do so separately for each model (Figure 12.9).

```
career_flat_prior <- career_sim_ab %>%
  ebb_fit_prior(H, AB)

data_frame(method = c("Flat prior", "Prior depending on AB"),
           model = list(career_flat_prior, career_ab_prior)) %>%
  unnest(map(model, augment, data = career_sim_ab)) %>%
  ggplot(aes(p, .fitted, color = AB)) +
  geom_point() +
  scale_color_continuous(trans = "log", breaks = c(1, 10, 100, 1000)) +
  geom_abline(color = "red") +
  facet_wrap(~ method) +
  labs(x = "True batting average (p)",
       y = "Shrunken batting average estimate")
```

Look at the bias when we don't take the AB to batting average relationship into account: batters with low AB and low averages were universally overestimated. This is exactly the issue we predicted in Chapter 7:

> Since low-AB batters are getting overestimated, and high-AB batters are staying where they are, we're working with a biased estimate that is systematically *overestimating* batter ability.

If you're interested, you could take this more complex model and perform the same examinations of credible intervals and priors that we did for the simple model. You could also incorporate some of the other trends that could affect your prior, such as year and handedness, that were considered in the hierarchical model in Chapter 11.3.

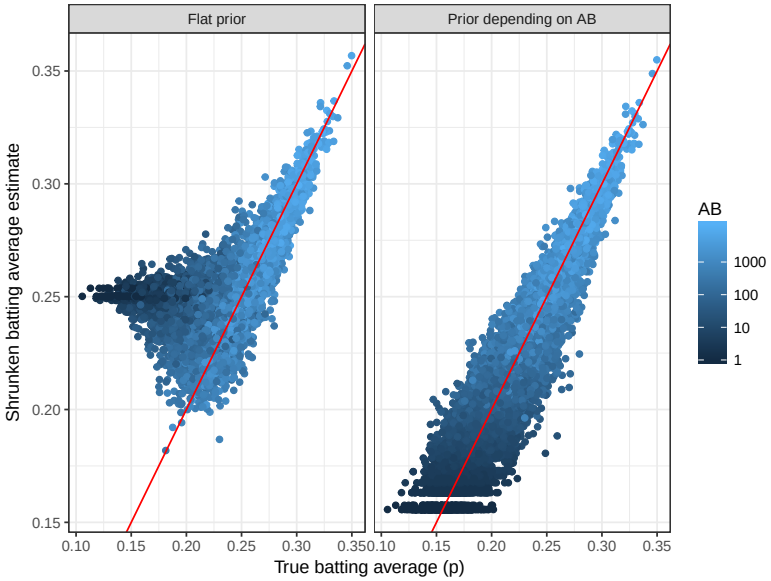

Figure 12.9: Comparison of the true batting average and the shrunken batting average, using either a single beta as the prior or a prior where p depends on AB.

13
Simulating replications

In Chapter 12, we ran a single simulation of our players' batting averages, used it to perform estimation, and then examined whether our results were accurate. This is a valuable way to sanity-check the accuracy of the empirical Bayes method.

But what if we just got lucky? What if empirical Bayes shrinkage works about half the time, and if the players had batted a bit differently it would have given terrible results? Similarly, even if the method worked on 10,000 players, can we tell if it would have worked on 1000, or 100? These are important concerns if we want to trust the method on our real data.

In this final chapter, we'll extend the simulation from Chapter 12. Rather than simulating a single example, we'll create **50 simulations**, and run the empirical Bayes method on each of them. We'll similarly learn how to vary an input parameter, the number of players, which will examine how the empirical Bayes approach is sensitive to the number of observations.

13.1 Setup

As usual, we start with code that sets up the variables analyzed in this chapter (in this case, the same code as Chapter 12.

```
library(Lahman)
library(dplyr)
library(tidyr)
library(purrr)

# Grab career batting average of non-pitchers
# (allow players that have pitched <= 3 games, like Ty Cobb)
pitchers <- Pitching %>%
  group_by(playerID) %>%
  summarize(gamesPitched = sum(G)) %>%
```

```
  filter(gamesPitched > 3)

# include the "bats" (handedness) and "year" column for later
career <- Batting %>%
  filter(AB > 0) %>%
  anti_join(pitchers, by = "playerID") %>%
  group_by(playerID) %>%
  summarize(H = sum(H), AB = sum(AB))

library(ebbr)
library(broom)

prior <- ebb_fit_prior(career, H, AB)
alpha0 <- tidy(prior)$alpha
beta0 <- tidy(prior)$beta
```

13.2 Replicating the beta-binomial simulation

The `crossing()` function from tidyr is very useful for performing multiple replications of a tidy simulation. Instead of performing repeating an operation in a loop, you can replicate your data within one data frame. [1]

[1] These simulations of 50 replications are the slowest-running code examples in the entire book. If you're following along and want to speed it up, you could decrease the number of replications.

```
set.seed(2017)

sim_replications <- career %>%
  crossing(replication = 1:50) %>%
  mutate(p = rbeta(n(), alpha0, beta0),
         H = rbinom(n(), AB, p))
```

After simulating values of p and H, we can then nest within each replication, and use the purrr package's `map()` function to fit the priors. The dataset is then stored with one row for each of the 50 replications, with the prior for each stored in a list column.[2]

[2] To learn more about the philosophy of storing models in a list column, check out Chapter 25 of the book R for Data Science.

```
library(ebbr)

sim_replication_models <- sim_replications %>%
  nest(-replication) %>%
  mutate(prior = map(data, ~ ebb_fit_prior(., H, AB)))

sim_replication_models
```

```
## # A tibble: 50 × 3
##    replication                 data
##          <int>               <list>
```

```
## 1            1 <tibble [10,388 × 4]>
## 2            2 <tibble [10,388 × 4]>
## 3            3 <tibble [10,388 × 4]>
## 4            4 <tibble [10,388 × 4]>
## 5            5 <tibble [10,388 × 4]>
## 6            6 <tibble [10,388 × 4]>
## 7            7 <tibble [10,388 × 4]>
## 8            8 <tibble [10,388 × 4]>
## 9            9 <tibble [10,388 × 4]>
## 10          10 <tibble [10,388 × 4]>
## # ... with 40 more rows, and 1 more
## #   variables: prior <list>
```

13.2.1 Estimations of hyperparameters

In each replication, we started by estimating a prior distribution, in the form of α_0 and β_0 hyperparameters. Since these estimated hyperparameters are the foundation of any empirical Bayes method, we'd like to know if they're consistently accurate.

```
sim_replication_priors <- sim_replication_models %>%
  unnest(map(prior, tidy), .drop = TRUE)

sim_replication_priors
```

```
## # A tibble: 50 × 4
##    replication  alpha   beta    mean
##          <int>  <dbl>  <dbl>   <dbl>
## 1            1 73.902 220.09 0.25138
## 2            2 71.324 212.02 0.25172
## 3            3 71.904 214.24 0.25128
## 4            4 68.094 203.69 0.25054
## 5            5 73.823 220.14 0.25114
## 6            6 76.333 227.72 0.25106
## 7            7 74.142 221.25 0.25100
## 8            8 72.276 216.33 0.25043
## 9            9 70.126 209.45 0.25083
## 10          10 72.269 215.90 0.25079
## # ... with 40 more rows
```

Figure 13.1 shows our estimations of α_0 and β_0 across all 50 replications, along with the true values shown as a dashed horizontal line. We notice that our estimates of are mostly unbiased: generally they're equally likely to be above or below the true parameter. We also note that the mean $\frac{\alpha_0}{\alpha_0+\beta_0}$ is almost always between .250 and .252. Since

this is what every player is being shrunk towards, it's good that the estimate is so precise.

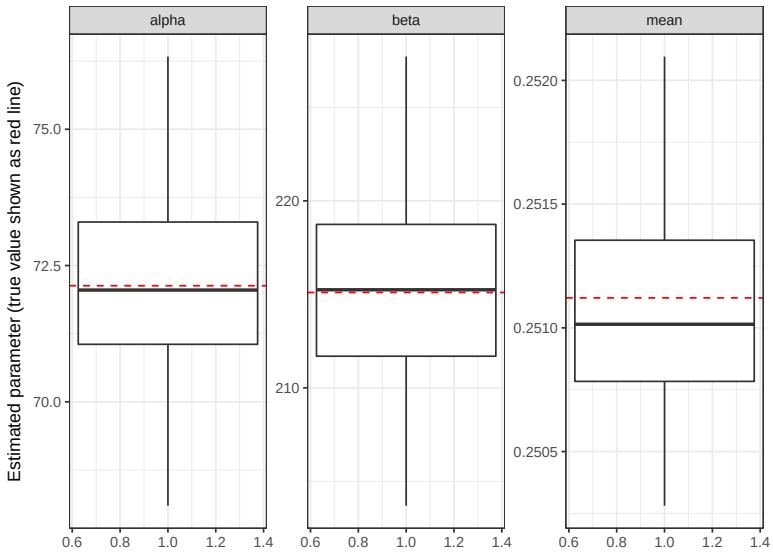

Figure 13.1: Estimated hyperparameters α_0, β_0, and the mean $\frac{\alpha_0}{\alpha_0+\beta_0}$ across 50 replications.

If we'd often estimated the hyperparameter poorly, then we should worry about using them as our prior. But our accuracy gives us confidence that we have enough data to apply the empirical Bayesian approach.

13.2.2 Estimates, intervals, and hypothesis testing across replications

We can then can examine whether the empirical Bayes shrinkage and credible intervals were consistently effective.

In Section 12.2.1, we used the mean squared error (MSE) between the estimate and the true batting average p as a metric for evaluating the method's performance, and for comparing it to the raw batting average H/AB. We can now repeat that comparison across the 50 replications (Figure 13.2).

```
sim_replication_au <- sim_replication_models %>%
  unnest(map2(prior, data, augment))

sim_replication_mse <- sim_replication_au %>%
  rename(Raw = .raw, Shrunken = .fitted) %>%
  gather(type, estimate, Raw, Shrunken) %>%
  group_by(type, replication) %>%
  summarize(mse = mean((estimate - p) ^ 2))
```

It looks like the MSE of empirical Bayes shrunken estimates was always much lower than the raw estimates, and was pretty consistent

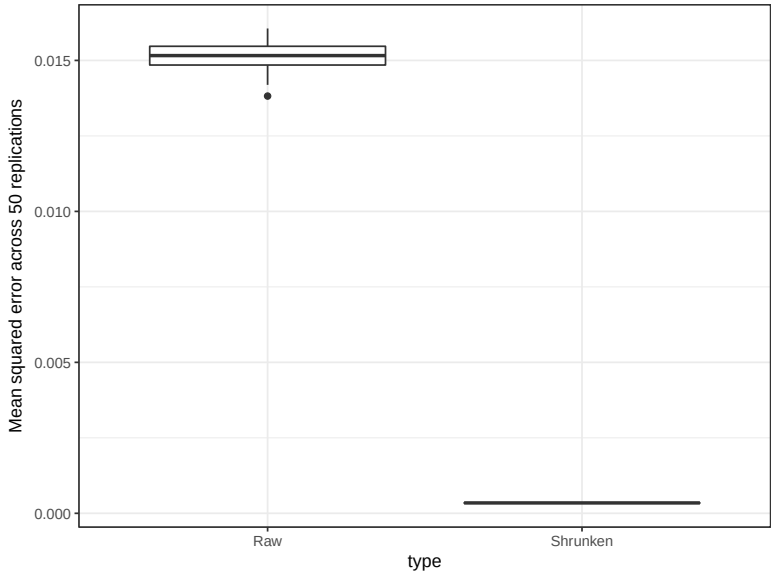

Figure 13.2: Comparison of the mean-squared error on 50 replications, using either the raw batting average or the shrunken batting average.

in its range. This is a good sign: even in 50 replications, it never fails "catastrophically." This is not true of all statistical methods!

In Section 12.3 we also saw that the credible intervals were well calibrated, where 95% credible intervals generally contained the true value about 95% of the time. We can now see if this is consistently true across replications (Figure 13.3). Indeed, it looks like the coverage of a 95% credible interval was generally between 94.4% and 95.5%.

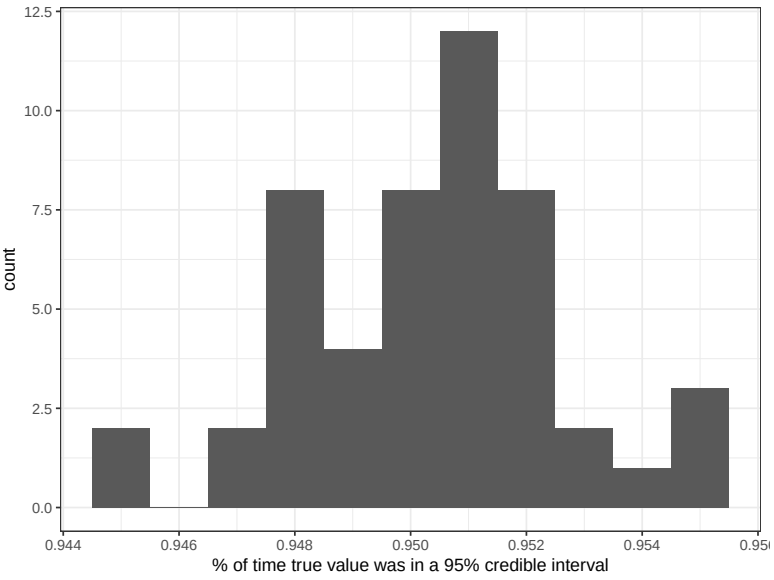

Figure 13.3: Distribution of the coverage probability of a 95% credible interval across simulations.

Is it well calibrated at other levels: does an 80% credible interval contain the true value about 80% of the time? Figure 12.7 from the last chapter tried varying the level of the credible interval, and exam-

ined how it affected the coverage probability. We can now recreate that plot, but do so across all fifty replications (Figure 13.4).

```
sim_replication_intervals <- sim_replication_models %>%
  crossing(cred_level = c(seq(.5, .9, .05), .95)) %>%
  unnest(pmap(list(prior, data, cred_level = cred_level), augment)) %>%
  select(replication, cred_level, p, .low, .high)
```

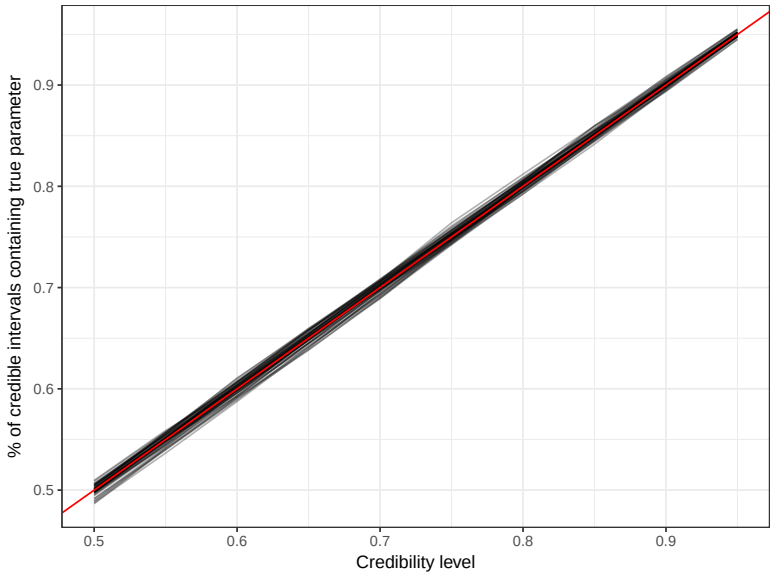

Figure 13.4: Comparison of the level of the credible interval to the fraction of players where the interval contains the true value. Each line represents one replication of the simulation; the red line represents $x = y$.

Each of these lines is one replication tracing from a "50% credible interval" to a "95% credible interval." Since all the replications are close to the red $x = y$ line, we can see that an X% credible interval contains the true value about X% of the time. This is an important lesson of tidy simulations: whenever you can make a plot to check one simulation to check accuracy or calibration, you can also recreate the plot across many replications.

We can also examine our method for false discovery rate control, and see whether we can trust a q-value of (say) .05 to keep the FDR below 5%, just as we did last chapter in Figure 12.8. The approach (code not shown) is similar to the one for credible interval coverage: group by each replication, then perform the same analysis we did on a single replication (Figure 13.5).

Each of these lines represents a replication tracing along every possible q-value threshold. We see that the proportion of false discoveries below a q-value is sometimes higher than the q-value promises, and sometimes lower. That's OK: the promise of FDR control isn't that the false discovery rate will always be exactly 5% (that would be impossible due to random noise), but that it is on average.

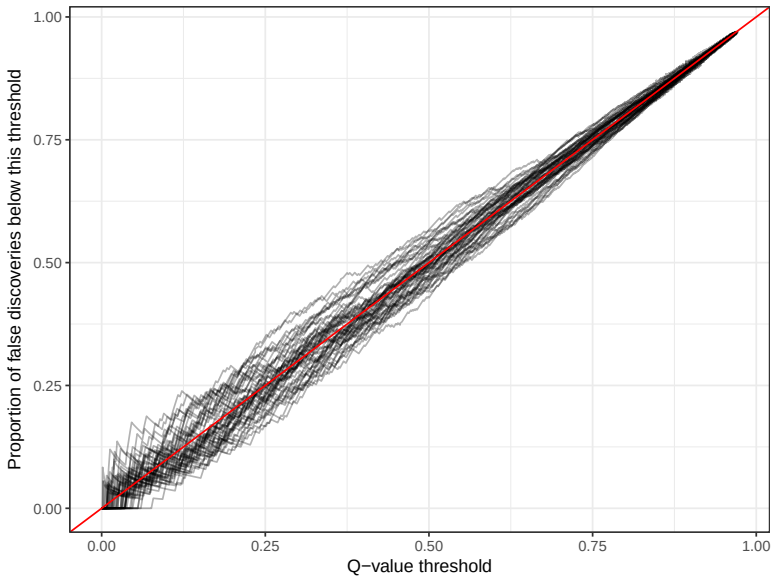

Figure 13.5: Comparison of the q-value threshold and the resulting false discovery rate. Each line represents one replication of the simulation; the red line represents $x = y$.

13.3 Varying sample size

In these two chapters, we've re-simulated our set of baseball players, and confirmed that empirical Bayes generally performed well. In what situations might the method *not* work?

One example of a case where empirical Bayes performs poorly is if we had fewer observations. If there were only three or four batters, we'd have no ability to estimate their prior beta distribution accurately, and therefore be shrinking the batting averages towards an arbitrary estimate. This is particularly dangerous because empirical Bayes doesn't account for the uncertainty in hyperparameter estimates, so our poor estimate of the prior would be all it had to go on.[3]

How many observations are *enough* to use these methods? 100? 1000? While this book was being developed in a series of online posts, I often received this question, and never had a good answer. But through simulation, we have the opportunity to examine the effect of the sample size on the performance of empirical Bayes estimation.

13.3.1 Simulating varying numbers of observations

Let's consider six possible sample sizes: 30, 100, 300, 1000, 3000, and 10,000 (10,000 is pretty close to our actual dataset size of 10388). We could perform simulations for each of these sample sizes, by randomly sampling from the set of players each time.[4]

```
set.seed(2017)

# nifty trick for sampling different numbers within each group
```

[3] Traditional Bayesian methods handle this by modeling the uncertainty about the beta distribution explicitly. That is, instead of estimating the α_0 and β_0 hyperparameters of the beta, they would have a *hyperprior* for the distributions of α_0 and β_0, which would get updated with the evidence. This is challenging but well-studied; see Chapter 5.3 of Bayesian Data Analysis for an example that also uses the beta-binomial (Gelman et al., 2003).

[4] To imitate the distribution that a new set of players might have, we resample with replacement, similarly to the process of bootstrapping.

```
# randomly shuffle with sample_frac(1), then filter
varying_size_sim <- career %>%
  select(-H) %>%
  crossing(size = c(30, 100, 300, 1000, 3000, 10000),
           replication = 1:50) %>%
  group_by(size, replication) %>%
  sample_frac(1, replace = TRUE) %>%
  filter(row_number() <= size) %>%
  ungroup()

varying_size_priors <- varying_size_sim %>%
  mutate(p = rbeta(n(), alpha0, beta0),
         H = rbinom(n(), AB, p)) %>%
  nest(-size, -replication) %>%
  mutate(prior = map(data, ~ ebb_fit_prior(., H, AB)))
```

The first step of empirical Bayes is to estimate the prior hyperparameters α_0 and β_0. How did the accuracy of these estimations depend on the sample size?

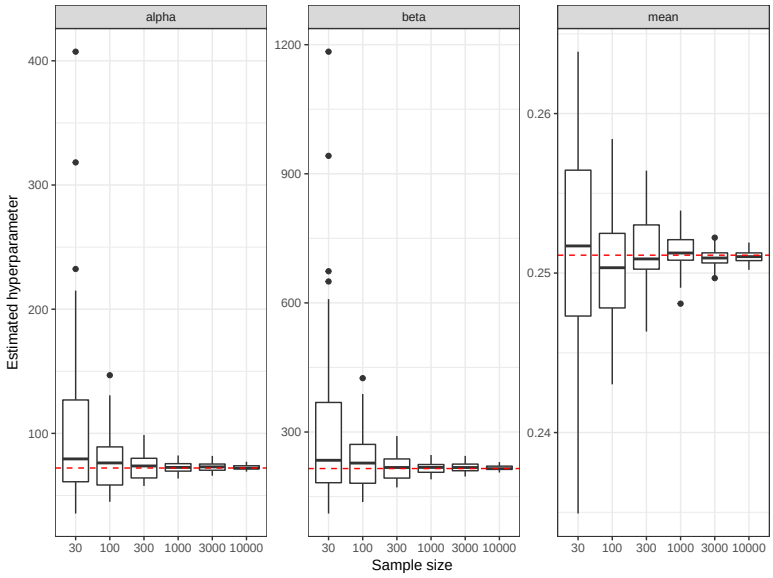

Figure 13.6: Estimated hyperparameters α_0, β_0, and the mean $\frac{\alpha_0}{\alpha_0+\beta_0}$ across 50 replications for each sample size. One condition with much higher α_0 and β_0 was removed for readability.

Figure 13.6 shows that for smaller numbers of observations, there was greater variance in the hyperparameter estimates. This makes sense: more data gives more evidence, and therefore a more consistently accurate maximum likelihood estimate.

The performance helps illustrate the danger of empirical Bayes on smaller samples. Rather than shrinking towards the true mean of 0.25112, the method may shrink everyone towards values below .24 or above .26. While this may seem like a small difference, they would af-

fect *everyone* in the dataset as a systematic error. In a number of the replications with 30 players, the algorithm also greatly overestimated both α and β, which means its prior would have too little variance (and the algorithm would "over-shrink").

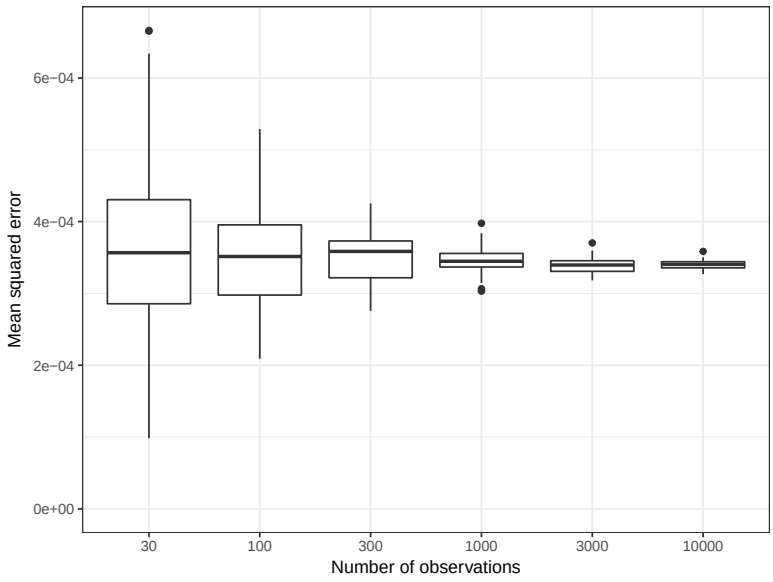

Figure 13.7: Distribution of the mean squared error (MSE) of empirical Bayes estimates across 50 replications, for simulated datasets of varying sizes.

How does this difference affect the accuracy of the empirical Bayes estimates? We can use the mean squared error (MSE) to quantify this. Figure 13.7 shows that the most dramatic difference across sizes was the variance of the MSE. That is, it was possible for empirical Bayes on lower sample sizes to show substantially less accurate estimates than on higher sample sizes, but also possible for the estimates to be *more* accurate.[5]

When we perform empirical Bayes on a real dataset, we don't have the luxury of knowing whether we were in a more accurate or less accurate "replication", so consistency matters. Thus, based on this MSE plot we might recommend sticking to datasets with at least 1,000 observations, or at least 100. Still, it's nice to see that even on 30 observations, we were often able to achieve accurate estimates.

[5] This may seem like a counter-intuitive result (how could the algorithm end up closer with less data?), but it's simply because with smaller samples one might get "lucky" with how the batters perform, with fewer that hit unusually high or low records relative to their true p. This is one complication of comparing across sample sizes.

13.3.2 Coverage of credible intervals

Besides the accuracy of the estimates, we can examine the credible intervals. Are the intervals still well-calibrated? Or are they sometimes systematically overconfident or underconfident?

Figure 13.8 shows the distribution of coverage proportions for each sample size. The credible intervals generally centered around 95%, especially for larger sample sizes. There is greater variation in the proportions for smaller sample sizes, but that's partly an artifact of

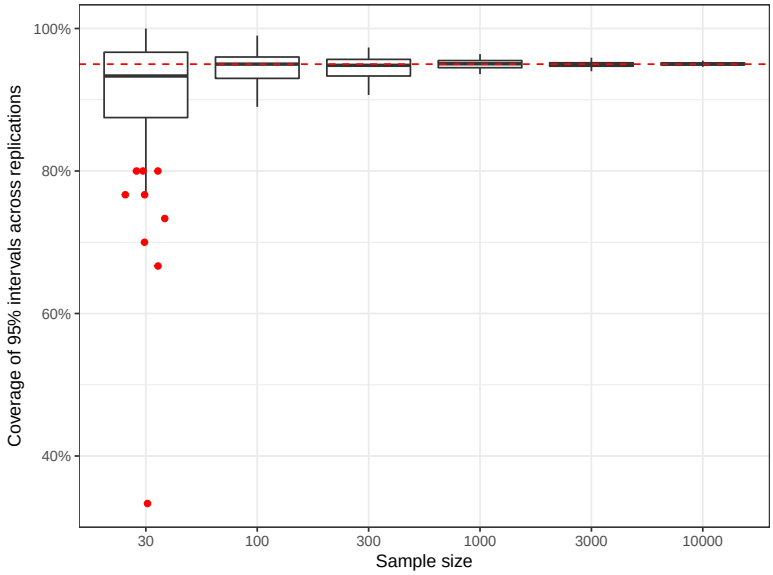

Figure 13.8: The coverage probabilities of 95% credible intervals, comparing 50 replications of each sample size. Cases where the coverage probability is lower than we'd expect by chance are shown as red points.

the noise present in smaller sizes (out of 30 95% intervals, it's easy for only 26 (86.7%) to contain the true p just by bad luck).

To separate this out, we show red points for replications where the coverage was low to a statistically significant extent.[6] We can see that there were 9 cases where we're confident the intervals were too narrow, all in the $n = 30$ replications. There was one particularly disastrous case replication where the credible intervals contained the true value only about 1/3 of the time. (More on that in a moment).

[6] Here, statistical significance was determined by computing a p-value with a binomial test, then by selecting those with less than a 5% false discovery rate using the Benjamini-Hochberg correction.

What causes credible intervals to be poorly calibrated? Generally it comes from when the prior is poorly estimated, and particularly when *when the variance of the prior is underestimated*. When the model underestimates the amount of variance, it tends to think the credible intervals are narrower than they are. The variance of the estimated prior can be represented by $\alpha_0 + \beta_0$: the higher that sum, the smaller the variance of the beta distribution.

Figure 13.9 compares the estimate of $\alpha_0 + \beta_0$ in each replication to the resulting credible interval coverage. When $\alpha_0 + \beta_0$ is underestimated relative to the true value (the vertical dotted line), the credible intervals tend to be too conservative and include the true value more than 95% of the time, and when $\alpha_0 + \beta_0$ is underestimated the intervals are too narrow.

We notice in particular the one replication of $n = 30$ where $\alpha_0 + \beta_0$ was dramatically overestimated, which resulted in only 1/3 of credible intervals containing the true value. This is the risk of empirical Bayes estimation for low sample sizes: we might estimate a very poor prior, and then treat it as though we're certain of its value.

This trend holds true across all sample sizes (as can be seen by the

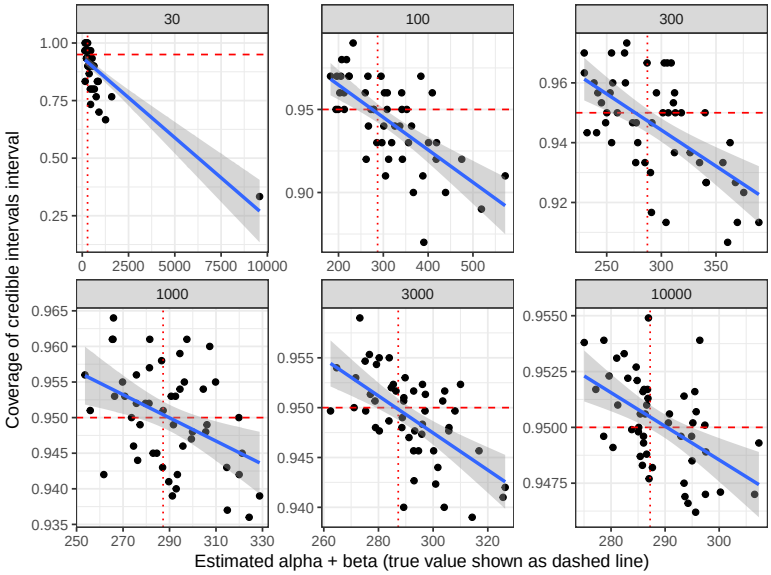

Figure 13.9: The relationship between the estimated value of $\alpha_0 + \beta_0$ in a particular replication and the resulting coverage probabilities. 95% shown as a horizontal dashed line, and the true value of $\alpha_0 + \beta_0$ is shown as a vertical dotted line. Best fit lines are shown in blue.

best-fit lines in blue), but it is trivial in sample sizes like 10,000, where coverage probabilities range from 94.5% to 95.5%. This is because the initial estimates of α_0 and β_0 were generally accurate, as we'd also seen in Figure 13.6. Still, this shows how our accuracy in estimating the prior can affect the performance of the rest of our methods.

This chapter shows just a few examples of simulations we could perform. What if each player had half as many at-bats? What if we varied the algorithm used to estimate our hyperparameters, using the (much faster) method of moments to compute the beta prior, rather than maximum likelihod? I encourage you to explore other simulations you might be interested in.

13.4 Conclusion: "I have only proved it correct, not simulated it"

Computer scientist Donald Knuth has a famous quote: **"Beware of bugs in the above code; I have only proved it correct, not tried it."** I feel the same way about statistical methods.

When I look at mathematical papers about statistical methods, the text tends to look something like:

> Smith et al (2008) proved several asymptotic properties of empirical Bayes estimators of the exponential family under regularity assumptions i, ii, and iii. We extend this to prove the estimator of Jones et al (2001) is inadmissable, in the case that $\hat{\theta}(x)$ is an unbiased estimator and $g(y)$ is convex…

This kind of paper is an important part of the field, but it does almost nothing for me. I'm not particularly good at manipulating

equations, and I get rustier every year out of grad school. If I'm considering applying a statistical method to a dataset, papers and proofs like this won't help me judge whether it will work. ("Oh- I should have known that my data didn't follow regularity assumption ii!") What does help me is the approach we've used here, where I can see for myself just how accurate the method tends to be.

For example, I recently found myself working on a problem of logistic regression that I suspected had mislabeled outcomes (some zeroes turned to ones, and vice versa), and read up on some robust logistic regression methods, implemented in the robust package. But I wasn't sure they would be effective on my data, so I did some random simulation of mislabeled outcomes and applied the method. The method didn't work as well as I needed it to, which saved me from applying it to my data and thinking I'd solved the problem.

For this reason, no matter how much math and proofs there are that show a method is reliable, I really only feel comfortable with a method once I've worked with it simulated data. It's also a great way to teach myself about the statistical method. I hope in these simulation chapters, and indeed throughout this book, that you have found my approaches to concrete examples and simulation as useful as I have.

14
Bibliography

Agresti, A. and Hitchcock, D. B. (2005). Bayesian inference for categorical data analysis. *Statistical Methods and Applications*, 14(3):297–330.

Friendly, M. (2015). *Lahman: Sean Lahman's Baseball Database*. R package version 4.0-1.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2003). *Bayesian Data Analysis*. Chapman and Hall/CRC.

Käll, L., Storey, J. D., MacCoss, M. J., and Noble, W. S. (2008). Posterior error probabilities and false discovery rates: Two sides of the same coin. *Journal of Proteome Research*, 7(1):40–44. PMID: 18052118.

Morgan, M. (2016). *DirichletMultinomial: Dirichlet-Multinomial Mixture Model Machine Learning for Microbiome Data*. R package version 1.14.0.

Stasinopoulos, M. and Rigby, B. (2016). *gamlss: Generalised Additive Models for Location Scale and Shape*. R package version 5.0-1.

Storey, J. D. (2002). A direct approach to false discovery rates. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):479–498.

Storey, J. D. (2003). The positive false discovery rate: a bayesian interpretation and the q-value. *Ann. Statist.*, 31(6):2013–2035.

Storey, J. D. and Tibshirani, R. (2003). Statistical significance for genomewide studies. *Proceedings of the National Academy of Sciences*, 100(16):9440–9445.

Yee, T. W. (2016). *VGAM: Vector Generalized Linear and Additive Models*. R package version 1.0-2.