Binomial data from the Bayesian perspective

Patrick Breheny

September 22, 2016

In last week's lab, we were introduced to the function `binom.test` and we learned how it worked by recreating the analyses that it performs "manually" in `R`. Unfortunately, there is no simple equivalent function in `R` that does the same kind of analysis from the Bayesian perspective. So today, we're going to create one for ourselves.

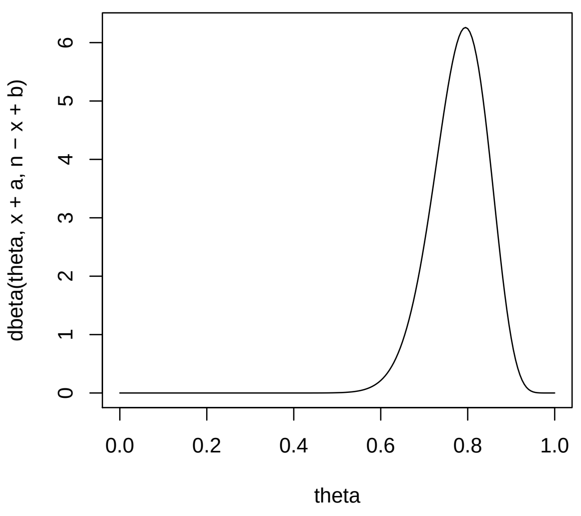
1 Starting out: doing the analysis outside of a function

Before we can create a function to do the analysis, we should make sure that we can actually do the analysis in the first place. So let's check that we can re-create the results from class in `R`. Let's work with the 25-week survival example and a uniform prior:

```
> x <- 31
> n <- 39
> a <- 1  ## Prior alpha
> b <- 1  ## Prior beta
```

Now, we know from lecture that the posterior will follow a Beta$(x + a, n - x + b)$ distribution. As with the binomial distribution last week, `R` has `dbeta`, `pbeta`, `qbeta`, and `rbeta` functions, and that's all we need to perform our analyses. So, for example, we can plot the posterior density:

```
> theta <- seq(0, 1, 0.005)
> plot(theta, dbeta(theta, x+a, n-x+b), type="l")
```



We can find out, say, the posterior probability that $\theta < 0.6$:

```
> pbeta(0.6, x+a, n-x+b)

[1] 0.006064628
```

And we can construct the central 95% interval:

```
> qbeta(0.025, x+a, n-x+b)

[1] 0.643522

> qbeta(0.975, x+a, n-x+b)

[1] 0.8916034

> ## Or:
> qbeta(c(0.025, 0.975), x+a, n-x+b)

[1] 0.6435220 0.8916034
```

2 How should our function work?

Basically, what we're going to do now is just put that stuff above into a function. But let's give a little thought first as to what our function should look like and accomplish. Let's call our function `binom.bayes` and have it work somewhat analogously to `binom.test`. In other words, I'd like to be able to run `binom.bayes(31,39)` and get my results. So obviously, the function is going to have to have two *required arguments*, `x` and `n`. But we'd also like to have a number of options to adjust the analysis, such as:

- `a`, `b`: So we can change the prior
- `level`: In case we want something other than a 95% posterior interval

- `plot`: It would be nice to optionally plot the posterior density

So, the basic outline of our function would look like this:

```
> binom.bayes <- function(x, n, a=1, b=1, level=0.95, plot=FALSE, ...) {
+ }
```

A few remarks:

- In the above, `x` and `n` are required arguments to `binom.bayes`. If we fail to pass `x` and `n`, the function will produce an error. This is as it should be – without knowing `x` and `n`, there is nothing the function can possibly do.
- The other arguments, though, have *default values*. We *can* specify `a` if we want to, but if we don't, the function will just use the default value `a=1`.
- When calling `binom.bayes` (or any `R` function), we can refer to the arguments either by name or by position. For example, `binom.bayes(31,39)` will work fine, because `R` will assume that the first argument, `31`, is `x`, and the second argument, `39`, is `n`. However, `binom.bayes(39, 31)` will not work, because the arguments are in the wrong order. We can, however, pass arguments in any order if we call them by name: `binom.bayes(n=39, x=31)` works. Also, it is fine to mix the two, as in `binom.bayes(31, 39, level=0.9)`.
- The `...` argument is a special argument in `R` that allows any number of additional arguments to be passed along to other functions called by `binom.bayes`. We'll see how this works and why it is useful in the next section.

3 Writing our function

OK, let's put the stuff from Section 1 inside the function we set up in Section 2:

```
> binom.bayes <- function(x, n, a=1, b=1, level=0.95, plot=FALSE, ...) {
+   theta <- seq(0, 1, 0.005)
+   plot(theta, dbeta(theta, x+a, n-x+b), type="l")
+   qbeta(c(0.025, 0.975), x+a, n-x+b)
+ }
```

This obviously doesn't really work as written, because the `level` and `plot` arguments don't actually do anything yet. Let's start with `plot`. We need to make the first two lines *conditional expressions*, meaning that they only get run if `plot=TRUE`. This can be accomplished with an `if` statement:

```
> binom.bayes <- function(x, n, a=1, b=1, level=0.95, plot=FALSE, ...) {
+   if (plot) {
+     theta <- seq(0, 1, 0.005)
+     plot(theta, dbeta(theta, x+a, n-x+b), type="l")
+   }
+   qbeta(c(0.025, 0.975), x+a, n-x+b)
+ }
```

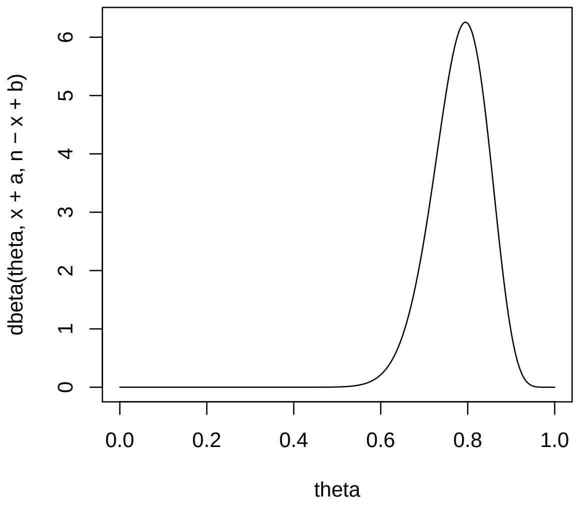
Note the syntax:

- `if` is followed by something in parentheses that is either `TRUE` or `FALSE` – in this case, that is `plot` itself, but it could also be an expression such as `x==5`

- This is then followed by one or more expressions or assignments that will only be run if the condition is `TRUE`. If you just have one line, you don't need the brackets, for example `if (x > 5) print("x is bigger than 5!")`. However, if you have multiple lines that need to be run conditionally, you need to bunch them together with curly brackets, as in the above.

Let's check to see that this actually works:

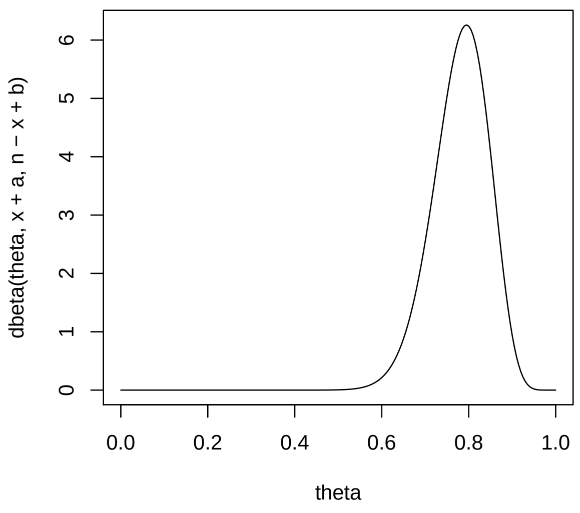
```
> binom.bayes(31,39)

[1] 0.6435220 0.8916034

> binom.bayes(31,39, plot=TRUE)
```



```
[1] 0.6435220 0.8916034
```

Unfortunately, however, we can't actually change anything about the plot yet:

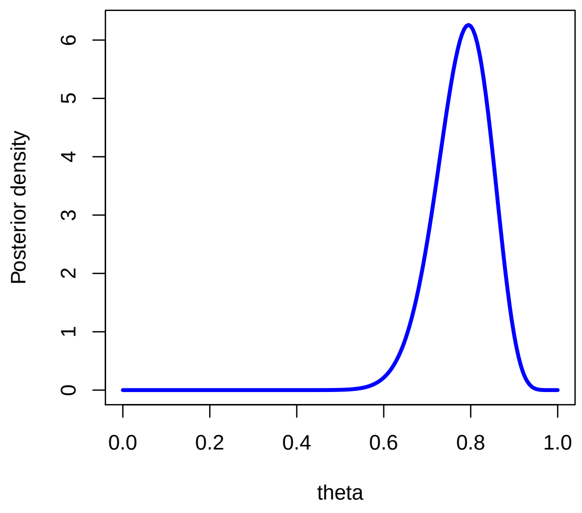
```
> binom.bayes(31,39, plot=TRUE, col="blue")
```



```
[1] 0.6435220 0.8916034
```

This is where those dots come in. By passing along any extra arguments to the `plot` function, we can take advantage of all the usual plotting options in `R`:

```
> binom.bayes <- function(x, n, a=1, b=1, level=0.95, plot=FALSE, ...) {
+   if (plot) {
+     theta <- seq(0, 1, 0.005)
+     plot(theta, dbeta(theta, x+a, n-x+b), type="l", ...)
+   }
+   qbeta(c(0.025, 0.975), x+a, n-x+b)
+ }
> binom.bayes(31,39, plot=TRUE, col="blue", lwd=3, ylab="Posterior density")
```



```
[1] 0.6435220 0.8916034
```

We still need to make the `level` option work, though:

```
> binom.bayes <- function(x, n, a=1, b=1, level=0.95, plot=FALSE, ...) {
+   if (plot) {
+     theta <- seq(0, 1, 0.005)
+     plot(theta, dbeta(theta, x+a, n-x+b), type="l", ...)
+   }
+   lp <- (1-level)/2
+   up <- 1-(1-level)/2
+   qbeta(c(lp, up), x+a, n-x+b)
+ }
> binom.bayes(31, 39, level=0.9)

[1] 0.6679723 0.8772883

> binom.bayes(31, 39, level=0.8)

[1] 0.6955027 0.8594132
```

Let's add one last feature. As you will show on the homework, the posterior mode of a beta distribution is $(\alpha - 1)/(\alpha + \beta - 2)$. Let's return the posterior mode along with the interval:

```
> binom.bayes <- function(x, n, a=1, b=1, level=0.95, plot=FALSE, ...) {
+   if (plot) {
+     theta <- seq(0, 1, 0.005)
+     plot(theta, dbeta(theta, x+a, n-x+b), type="l", ...)
+   }
+   lp <- (1-level)/2
+   up <- 1-(1-level)/2
+   interval <- qbeta(c(lp, up), x+a, n-x+b)
+   mode <- (x+a-1) / (x+a+n-x+b-2)
+   list(mode=mode, interval=interval)
+ }
> binom.bayes(31, 39)

$mode
[1] 0.7948718

$interval
[1] 0.6435220 0.8916034
```

In case you hadn't noticed, by default an `R` function returns its final evaluated expression (i.e., whatever you put in the last line will be returned by the function). You could change this using the `return()` function if you wished.

We now have a pretty useful function. One nice additional feature would be if it calculated the highest posterior density interval for us. Sounds like a nice homework problem. . . .

4 A little glimpse at Monte Carlo methods

Suppose there was some other hospital at which 10 out of 20 infants born at 25 weeks gestation survived. Let θ_1 denote the probability of a 25-week infant surviving at Johns Hopkins and θ_2 denote the probability of a 25-week infant surviving at this other hospital. In the actual samples, the Johns Hopkins infants were more likely to survive, but there is considerable uncertainty about the true long-run survival probabilities at each hospital. We might be interested in asking, what is the probability that $\theta_1 > \theta_2$ (conditional on the data, of course)?

This certainly seems like a reasonable question, and yet, nothing we've talked about yet provides an answer. Although we know the posterior distributions of θ_1 and θ_2, this is a question about their joint distribution, and we don't have any convenient functions in `R` to help us with multivariate beta distributions. However, if we can generate random numbers from the posterior distributions (and of course we can in this case), we can use that to obtain good numerical approximations of $P(\theta_1 > \theta_2 | x_1, x_2)$.

Let's try this out with 10,000 draws from each posterior, assuming a uniform prior for each hospital:

```
> theta1 <- rbeta(10000, 32, 9)
> theta2 <- rbeta(10000, 11, 11)
> mean(theta1 > theta2)

[1] 0.9879
```

In other words, we can be 99% certain that a 25-week infant is more likely to survive at Johns Hopkins than the other hospital. This way of obtaining numerical approximations to otherwise intractable problems is known as *Monte Carlo integration*. As data becomes more complex, we rapidly lose the abiliy to obtain nice closed-form solutions to Bayesian posteriors like we have for binomial data. For this reason, modern Bayesian data analysis relies heavily on Monte Carlo approaches like the above to calculate quantities of interest from posterior distributions.