The binomial distribution

Patrick Breheny

September 17, 2014

So, today we're going to learn about `R`'s suite of functions for the binomial distribution. In addition, we'll see how to carry out hypothesis tests and construct confidence intervals for binomially distributed data.

1   The binomial distribution

`R` has four functions related to the binomial distribution:

- `dbinom`: Probability mass function
- `pbinom`: Cumulative distribution function
- `qbinom`: Quantile function (inverse of CDF)
- `rbinom`: Generates random variables from a binomial distribution

This is a common pattern in `R`; as we will see, there is are `dnorm`, `pnorm`, `qnorm`, and `rnorm` functions that do the same thing for the normal distribution, and so on for many other distributions.

The `dbinom` function evaluates the binomial formula, returning $P(X = k)$:

```
> dbinom(31, size=39, prob=0.7)

[1] 0.06369

> ## Same as:
> choose(39,31) * 0.7^31 * 0.3^8

[1] 0.06369
```

As an aside, note the use of the `choose` function; you could use factorials directly, but this can lead to overflow inaccuracies if the numbers are large enough:

```
> ## This is OK:
> print(choose(39,31), digits=15)

[1] 61523748

> print(factorial(39)/(factorial(31)*factorial(8)), digits=15)

[1] 61523748

> ## This is a problem:
> print(choose(139,131), digits=15)

[1] 2815120424853

> print(factorial(139)/(factorial(131)*factorial(8)), digits=15)

[1] 2815120424853.04
```

The `pbinom` function evaluates the binomial CDF:

```
> pbinom(31, size=39, prob=0.7)

[1] 0.9336

> ## Same as:
> sum(dbinom(0:31, size=39, prob=0.7))

[1] 0.9336
```

Note that in the second line of code, I passed a vector of values to `dbinom`; this is fine, and is much more convenient than writing a for loop. For example, it allows you to plot the whole distribution with a single line of code:

```
> barplot(dbinom(0:39, 39, 0.7), names=0:39, border=FALSE)
```
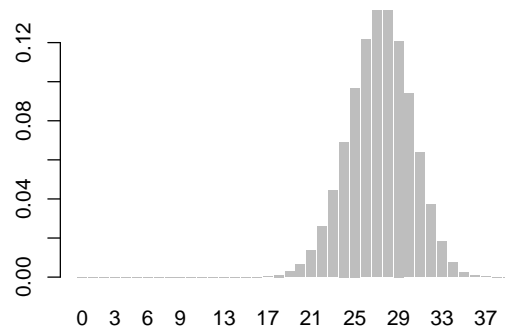

Finally, `rbinom` generates a vector of random variables from the binomial distribution (yes, I know I skipped `qbinom`; it's not very useful):

```
> x <- rbinom(100, size=39, prob=0.7)
> x

 [1] 29 28 26 25 30 28 29 28 28 26 29 25 31 24 24 28 22 31 25 30 26 29 26
[24] 27 25 26 26 28 25 29 28 24 22 23 25 24 26 29 26 27 26 27 25 23 21 30
[47] 21 29 27 29 25 27 27 27 31 26 28 26 27 24 28 28 26 26 24 30 23 32 27
[70] 26 28 26 29 27 29 24 26 29 26 27 28 27 27 31 28 26 31 31 30 29 27 25
[93] 32 32 26 22 25 28 28 27

> barplot(table(x), border=FALSE)
```
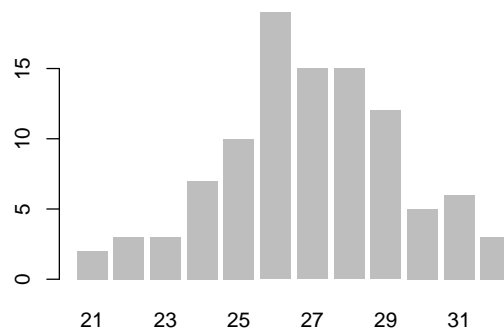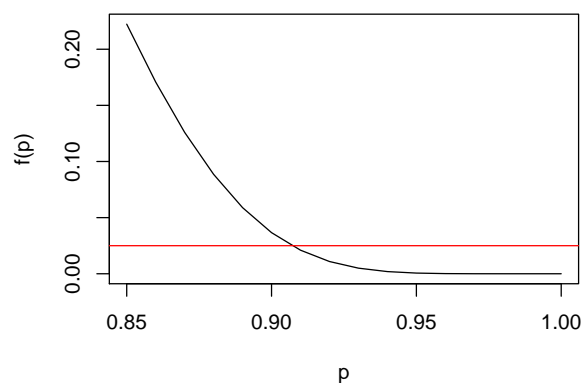
2   Constructing the Clopper-Pearson interval

As we discussed in class, we need a way to find the values p_L and p_U such that we would reject the hypothesis concerning p at the $\alpha = 0.025$ level for all the values outside $[p_L, p_U]$. Let's draw a picture of what's going on:

```
> f <- function(p) pbinom(31, 39, p)
> p <- seq(0.85,1,0.01)
> plot(p, f(p), type="l")
> abline(h=0.025, col="red")
```


   This is our first example of a user-created function in R. R of course comes with a lot of functions, but here, on the first line, we created a new function `f` that didn't exist before. Once we create it, we can call it, as we do on the third line with `f(p)`. This is a pretty simple function; we'll create more complicated ones later in the course.

   The illustration gives us a good idea of where p_u is, but how can we find the exact value of p_u? This is what is known in numerical analysis as a *root-finding problem*; i.e., it can be restated as finding the zero, or root, of a function. Strictly speaking, we want to find where our function intersects 0.025, not where it

intersects zero, but this can be trivially fixed by subtracting off 0.025 from `f(p)`. R's function for root-solving in one dimension is called `uniroot`. Here's what it looks like in action:

```
> f.u <- function(p) pbinom(31, 39, p) - 0.025
> uniroot(f.u, 0:1)$root

[1] 0.907
```

Note that we need to supply an interval to `uniroot`; here, $[0, 1]$ is an obvious interval for p. Now let's find p_L:

```
> f.l <- function(p) 1 - pbinom(30, 39, p) - 0.025
> uniroot(f.l, 0:1)$root

[1] 0.6354
```

So, by inverting the hypothesis test, we have obtained the 95% confidence interval $[0.635, 0.907]$ for p. Of course, this is a pretty common task in statistics, so it shouldn't be surprising that R has a function to calculate the confidence interval directly:

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


	Exact binomial test

data:  31 and 39
number of successes = 31, number of trials = 39, p-value =
0.0002941
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.6354 0.9070
sample estimates:
probability of success
              0.7949
```

Note that we get the same interval from `binom.test` that we got "manually" – in practice, of course, you'd use `binom.test`, but it's important to see where that interval comes from.