Distributions in R: Exercises

[Your Name Here]

2020-07-28

Below are some exercises for you to complete on your own. Make sure to add your name in the above YAML where it says "[Your Name Here]." When you're done with this problem set. Just hit "Knit" and email your pdf output to milesdw2@illinois.edu

Exercise 1: ***A colleague needs help making a p.d.f.!***

You have a colleague that says they don't trust the `dnorm` function. "How can I know it's actually returning the correct probabilities?!" they ask.

Fortunately, `R` is a functional programming language. *That means you can create your own custom functions!*

Knowing this, you tell your colleague that they can make their own version of `dnorm`. They can then compare the output from their custom function to that provided by `dnorm` to learn whether the base `R` function is trustworthy.

As it turns out, they've already started working on such a function. However, the function is returning results that don't equal what `dnorm` is returning. *That shouldn't be!* Can you help them figure out where they've gone wrong?

Here's what they have:

```
my.dnorm = function(x,mean=0,sd=1) { # start bracket
  pdf = # equation for p.d.f.
    1/(sd*sqrt(2*pi)) * exp(-0.5 * ((x - mean)/sd))
  return(pdf) # tell the function to return the pdf
} # close brackets
```

The following code will compare the results to determine if both functions return the same thing. Right now, it's coming back `FALSE`.

```r
x = seq(-5,5,0.01) # vector from -5 to 5 in 0.01 increments
base.dnorm = dnorm(x=x)
my.results = my.dnorm(x=x)
all(base.dnorm==my.results) # if output is TRUE, then you can trust dnorm!
```

```
## [1] FALSE
```

Hint: Did they use the right equation for the normal distribution p.d.f.?

Exercise 2: ***Binomial distributions***

The Binomial distribution analogues for the `dnorm`, `pnorm`, etc. functions in `R` are:

- `dbinom`,
- `pbinom`,
- `qbinom`,
- `rbinom`.

You can learn more about these functions by typing `?dbinom` into your console.

Using these functions, can you plot the p.m.f., c.d.f., and empirical distribution for a binomial distribution with 5 trials where the probability of success is 0.25?

This would be like a scenario where you have a "trick" coin where the likelihood of Heads, rather than being 50%, is 25%, and you want to know the probability of getting a certain number of heads after 5 tosses.

As a quick example, this is how to get the p.m.f. for a Binomial distribution with 2 trials and probability of success equal to 0.6:

```r
n.trials = 2
dbinom(x = 0:n.trials, size = n.trials, prob = 0.6)
```

```
## [1] 0.16 0.48 0.36
```

Exercise 3: ***Conditional Probability Distributions***

You will almost always work with *conditional* probabilities or conditional *means*—or a conditional *something*—in your research.

The function for the normal distribution is an example of a conditional probability function:

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

The "|" in $f(\cdot)$ signifies that the function is evaluated at x *given* μ and σ. That is, the mean and standard deviation of a random variable *condition* the probability of observing a given value of x.

Say you have a friend who wants to learn some intuition behind conditional probabilities.

You decide to use the example of the conditional probability of pulling out up to 10 red jelly beans in 10 draws *without replacement* from a bowl of 100 total blue and red jelly beans. You provide the following code to demonstrate the idea more intuitively:

```
# jelly beans
beans = c('red','blue')

# number of red jelly beans
N1 = 20
N2 = 80

# total number of beans
tot.beans = 100

# make a bowl of jelly beans
bowl1 = c(
  rep(beans[1],N1),
  rep(beans[2],tot.beans-N1)
)
bowl2 = c(
  rep(beans[1],N2),
  rep(beans[2],100-N2)
)

# use loop to iterate times you pull from the bowl:
# use the foreach package
library(foreach)
foreach(i = 1:1000, .combine = 'c') %do% {
  sum(sample(bowl1,size=10,replace=F)=='red')
} -> out1
foreach(i = 1:1000, .combine = 'c') %do% {
  sum(sample(bowl2,size=10,replace=F)=='red')
} -> out2

# print out frequencies per number of times
table(out1)
```

```
## out1
##   0   1   2   3   4   5   6
##  95 259 313 214  90  26   3
```

```
table(out2)
```

```
## out2
##   3   4   5   6   7   8   9  10
##   1   3  15  79 244 293 253 112
```

You tell your friend that `table(out1)` returns a summary of the number of times you get x many reds in 10 draws from the bowl when there are 20 reds total in the bowl. Meanwhile, `table(out2)` returns a summary of the number of times you get x many reds in 10 draws from the bowl when there are 80 total red jelly beans. “As you can see,” you tell them, “a bigger N of reds leads to a greater likelihood that you’ll get up to 10 red jelly beans in 10 draws from the bowl.” You continue, "this is what we mean by ‘conditional’ probability—the probability of pulling out x reds from the bowl is conditioned by the total number of reds in the bowl.

Part (A)

But then your friend asks, “What’s the minimum N such that I could get at least 10 reds in 10 pulls *at least once*?”

How would you modify your code to answer this question?

Note: You can either summarize in words how you would modify the code, or write out the code. **You don’t have to do both!**

Part (B)

You friend also asks, “Does the effect of the total number of reds in the bowl change depending on the total number of beans in the bowl?”

Play arround with values for `tot.beans` in the above code to see if it makes a difference.

Part (C)

Finally, your friend says, “One more Question. I promise!”

“What happens if we put each bean we draw from the bowl back in before we draw again?”

Change the code to answer their question.