Power Analysis by simulation: t-test

Crispin Jordan

07/06/2021

This file demonstrates how to conduct a power-analysis for a t-test via simulation.

Establishing parameters

Our first step is to establish the parameters (essential components) of our power analysis.

In this file, we demonstrate power analysis for a standard (i.e., 'Students') t-test. Therefore, we will create a dataset with two groups, which we'll name `group.1` and `group.2` (inspired, eh?).

We'll imagine that our control group (`group.1`) has a mean of `5`. As well, we'll imagine that we want our experiment to detect a change in the mean of 1 unit, and we expect that this change will increase the mean. Therefore, we'll set the mean of the manipulated group (`group.2`) to have a mean of `6`.

When programming (which is what we're doing here), it is often useful to assign a value to an object, and then use the object in the main body of the program. This approach allows us to keep all parameters in one area of the code; if we wish to change the value of something (e.g., the mean of `group.1`) we can do so by changing its value at one location in our program (rather than hunting throughout our program for every place that we use that value). For example, we'll set the mean values for `group.1` and `group.2`, respectively, as:

```
mean.1 <- 5
mean.2 <- 6
```

Now, we similarly need to assign a value for the standard deviation for residual variation; we'll assume that this variation is equal in our two groups, so we'll only specify one value (`sd.both`; we'll imagine that the standard deviation equals `0.5`).

```
sd.both <- 0.5
```

Next, we'll set the sample size for our experiment, for which we wish to determine the statistical power:

```
sample.size <- 5
```

Finally, we need a way to keep track of how many times we've had a 'successful' experiment. What do we mean by 'successful'? When we conduct a power analysis, we imagine that an effect truly exists (in our case, we did this by setting different mean values for the two groups (`mean.1 <- 5`, `mean.2 <- 6`)). We then establish the **power** of our experiment by determining the *probability* of our being able to detect this effect (given our assumptions about the experiment, e.g., the value of `sd.both`) with a given sample size. To determine this *probability*, we run our experiment many times (via simulations) and count the number of times that we could detect an effect. The standard way of saying we've detected an effect is to obtain `p < 0.05` (but see lecture materials for warnings about this approach). Now, back to the matter at hand: we need a way to count the proportion of our simulated 'experiments' that are successful. We'll do this by setting the total number of simulated experiments that we'll run and by creating a counter that will increment each time a simulated 'experiment' yields `p < 0.05`:

```
nsims <- 10000
counter <- 0
```

Create a dataset

Our parameters, established above, describe the conditions for our experiment. Now that we know the conditions, we can use them to simulate a dataset.

We'll use the function `rnorm()` to simulate our data. `rnorm()` takes three arguments, **in this order**:

- The number of random numbers to be drawn from a normal distribution;
- The mean value for the normal distribution from which the numbers are drawn;
- The standard deviation for the normal distribution from which the numbers are drawn.

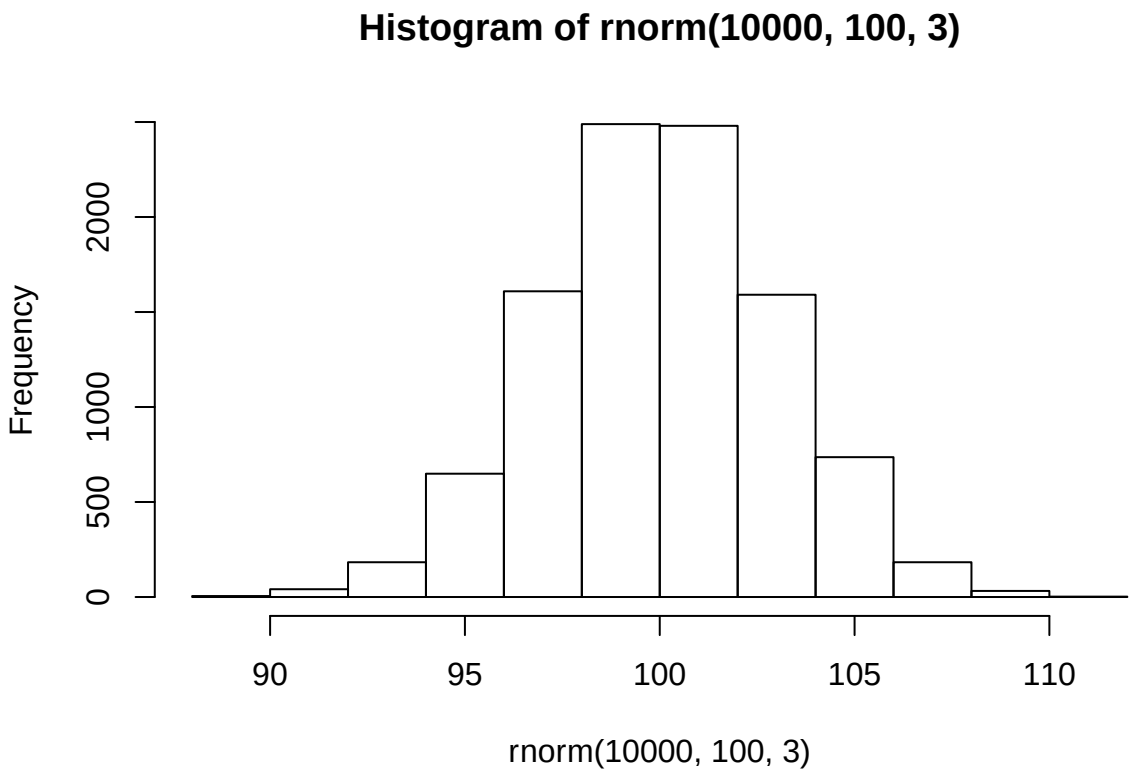
Therefore, we can draw 10 random numbers from a normal distribution with a mean of 100 and a standard deviation of 3, like this:

```
rnorm(10,100,3)
```

```
##  [1]  95.58886  97.35292 105.16731  95.23943 101.51461  99.70763 101.82953
##  [8]  99.36850  96.25744 106.10426
```

We can see the shape of the distribution by drawing lots of random numbers and plotting them in a histogram:

```
hist(rnorm(10000,100,3))
```



Now that you see how we can use `rnorm()`, you'll see that we can draw 5 random numbers (remember, we set `sample.size <- 5`, above) from each group, using the mean values and standard deviation we specified, above, as:

```
group.1 <- rnorm(sample.size, mean.1,sd.both)
group.2 <- rnorm(sample.size, mean.2,sd.both)
group.1
```

```
## [1] 5.084425 5.622888 5.298120 5.811789 6.085595
```

```
group.2
```

```
## [1] 5.344521 6.943372 6.389124 5.532942 6.622975
```

Analysing the data and counting the number of 'successes'

Now that we've created the data, we need to analyse it to see whether we've detected a difference between the two groups (based on `p < 0.05`). We'll use the function for a t-test, where we specify that we assume the standard deviation is equal in the two groups (this is an assumption of a Student's t-test); we'll also save the output of our t-test in an object we'll call, `t.out`:

```
t.out <- t.test(group.1,group.2,var.equal = TRUE)
```

`t.test` produces a lot of different output, stored in `t.out`.

For the sake of our power analysis, we only need to know the p-value from this output. We can obtain the p-value like this:

```
t.out <- t.test(group.1,group.2,var.equal = TRUE)
t.out$p.value
```

```
## [1] 0.1410511
```

Remember that we wanted to determine whether each simulated experiment was a 'success'? In this example, a 'success' means that we obtain `p < 0.05`; therefore, we can determine whether we have a successful experiment by comparing the p-value vs. `0.05`:

```
t.out$p.value < 0.05
```

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

(We're almost done...) Now, we need to count all the times that an experiment was a success. We'll do that by asking whether `p < 0.05` and increment our counter if it is true. We can do this with an `if()` statement, which will assess a statement (in our case, whether `t.out$p.value < 0.05`) and perform a task if that statement is `TRUE`:

```
if(t.out$p.value < 0.05){counter <- counter + 1}
```

Notice how we increment the counter: `counter <- counter + 1`. This means that we take the original value of `counter`, add 1 to it, and then re-assign this sum to the object `counter`. i.e., we replace the value of `counter` with its value incremented by 1.

Let's look at the value of `counter` now: ask yourself whether its value makes sense, given the p-value we obtained, above:

```
counter
```

```
## [1] 0
```

And that is how we analyse the data and keep track of whether the simulated experiment was a 'success'!

Repeating the simlated experiments many times.

It would be very cumbersome if we had to repeat the process, above, many times by hand. We need a way to automate this process to we can conduct many (i.e., `nsims <- 10000`) times. We can use a `for` loop for this purpose. A `for` loop looks like this:

```
my.numbers <- 0
for(i in 1:10){
  my.numbers <- my.numbers + 1
}
my.numbers
```

```
## [1] 10
```

The code, above stars by creating an object, `my.numbers` and setting it equal to zero. Then, we enter the `for` loop. The loop starts with this: `for(i in 1:10)`. This code creates an object, `i`. `i` will equal 1 the first time the `for` loop runs. It will equal 2 the next time the `for` loop runs, an so on, until `i` equals 10. You can think of `i` as a counter, and the `for` loop runs until `i` equals the final value (10, in this toy example). Each time the `for` loop runs, it will implement the code within the curly brackets: `my.numbers <- my.numbers + 1`. As explained above for our object, `counter`, this code increments `my.numbers` by one each time the code is run. As the code runs 10 times, we see that, when we exit the `for` loop, `my.numbers` equals 10.

Below, you'll see how we can use the `for` loop for the purposes of our simulations.

Bringing the code together

Now that you've learned the pieces, we can put them together, like this:

```
#It can be a good idea to run this code before the simulations; it deletes all objects
#from R, leaving you with a 'clean' working environment
rm(list=ls())

mean.1 <- 5
mean.2 <- 6
sd.both <- 0.5
sample.size <- 5
counter <- 0
nsims <- 10000

for(i in 1:nsims){
  group.1 <- rnorm(sample.size, mean.1,sd.both)
  group.2 <- rnorm(sample.size, mean.2,sd.both)
  t.out <- t.test(group.1,group.2,var.equal = TRUE)
  if(t.out$p.value < 0.05){counter <- counter + 1}
}

counter/nsims
```

```
## [1] 0.7903
```

Note the final line of code, which occurs outside the `for` loop: `counter/nsims`. This code divides the number of times that `p < 0.05` in our simulations (stored in `counter`) by the total number of simulated experiments (`nsims`); the result equals the proportion of simulated experiments in which we detected an 'effect', based on `p < 0.05`. This proportion equals the statistical **power** of our experiment.

Organizing code as a function

Perhaps you dislike how the code is organized, above. An alternative is to create a function that will perform a power analysis for a t-test: you would simply input the sample size in each group (we assume the sample size is the same in this example), the mean value for each group, and the standard deviation, run the code, and obtain the answer.

We can do so like this:

```
#First, we create our function, like this:

power.sim.t.test <- function(sample.size,mean.1,mean.2,sd.both){
  #Notice that we enter the sample size, mean values and sd when we call the function,
  #and these values are used in the code, below.
  counter <- 0
  nsims <- 10000
```

```
  for(i in 1:nsims){
    group.1 <- rnorm(sample.size, mean.1, sd.both)
    group.2 <- rnorm(sample.size, mean.2, sd.both)
    t.out <- t.test(group.1,group.2,var.equal = TRUE)
    if(t.out$p.value < 0.05){counter <- counter + 1}
  }
  return(counter/nsims)
}

#To run the function, we provide the sample size (5), mean of group 1 (5), mean of
#group 2 (6), and the sd (0.5):
power.sim.t.test(5,5,6,0.5)
```

```
## [1] 0.7923
```

Why might we want to organize our data as a function? Well, imagine that we wanted to run a power analysis for a variety of sample sizes. Can you think of code that would use our function, above, and iterate it for a series of sample sizes? Try it!