Running GUI-less: an introduction

Brian Williamson

BIOST 561: Computational Skills For Biostatistics I

16 November 2017

Running GUI-less

GUIs are picture books — they let you move things to other things, or right click on things and select options.

But you still type a post when you want to ask a question.

Command lines are much the same thing... if what you're trying to do is simple, then pictures and words are about the same. If what you're trying to do is complex, then words allow you to explain better.

Stack Exchange user `Sobrique` (2015)

Motivation

A typical R learning curve (on a personal machine):

1. Run commands by typing them line by line into a GUI (RStudio, RGUI, etc.), e.g.

```
> exp(1)
[1] 2.718282
> cos(pi)
[1] -1
> x <- 55
```

2. Use the GUI's editor (or, e.g., Notepad++) to create scripts and save to use/edit/etc. later

The first is actually the same as executing R from the command line, while the second is instrumental in writing long programs or simulation studies.

Motivation

Why not keep running R using a GUI and (R) scripts?

- Requires keeping R/RStudio open while the script is running
- Can take up significant resources on a personal machine
- May need to open multiple R/RStudio windows to run multiple jobs (there are some ways around this)

Many courses (e.g., 570s) and research will require a more sophisticated approach.

Example: permutation test

Permutation tests construct a sampling distribution for a test statistic, like the bootstrap.

However, they are used for different things in inference: permutation test for hypothesis testing, bootstrap for confidence intervals (usually).

In permutation testing, we resample in a manner consistent with the *null hypothesis*. These tests allow us to compute the sampling distribution for any test statistic (most useful when we do not actually know it), and obtain approximate p-values.

Example: permutation test

Consider SNP data — we want to measure difference in mean outcome in a dominant model for a single SNP. In this case, we could just use a t-test, but it is a nice illustration.

Some code to set this up: (from Ken's SISG course)

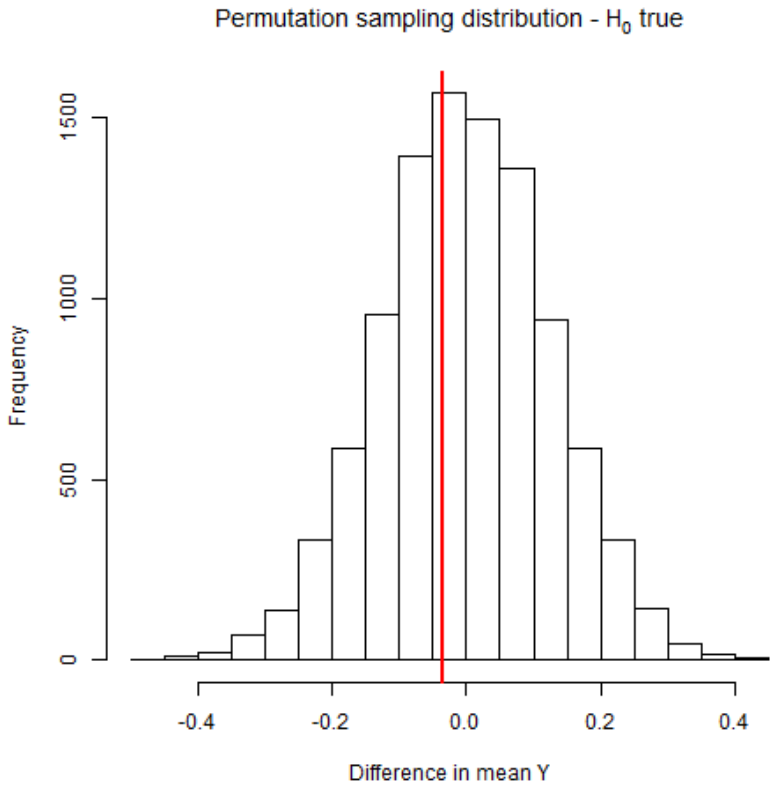
```
## make up some data (thanks Ken!)
carrier <- rep(c(0,1), c(100,200))
## outcome under the null and alternative
null.y <- rnorm(300)
alt.y <- rnorm(300, mean = carrier/2)
```

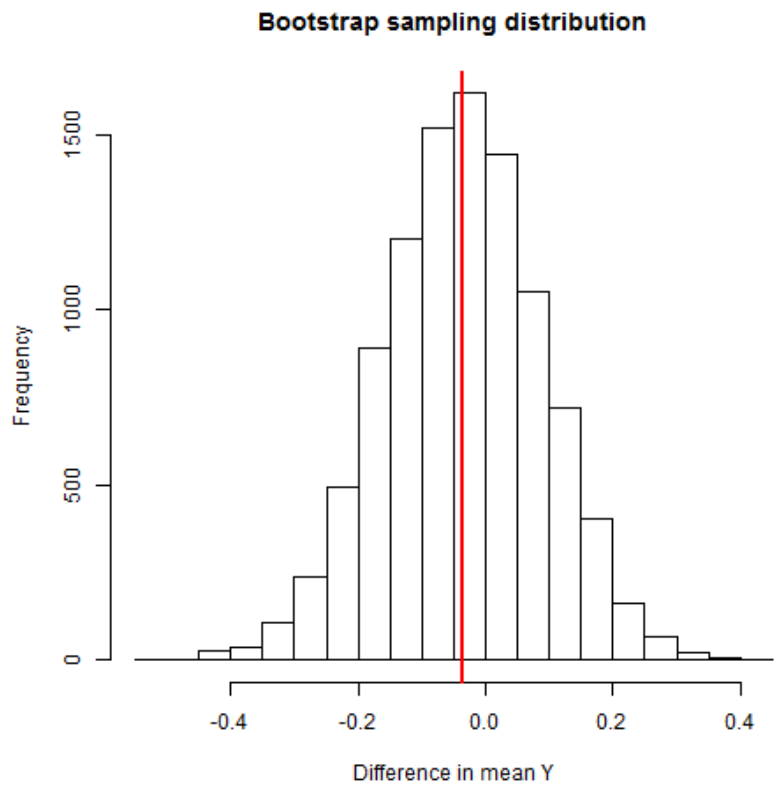
Example: permutation test (more code)

```
oneTest <- function(x, y) {
  ## get a new bootstrap sample of x
  xstar <- sample(x)

  ## return the difference in means
  ret <- mean(y[xstar == 1]) - mean(y[xstar == 0])

  return(ret)
}
## set a seed first!
set.seed(4747)
system.time(output.truenull <- replicate(10000, oneTest(carrier,
             null.y)))
system.time(output.falsenull <- replicate(10000, oneTest(carrier
             alt.y)))
null.diff <- mean(null.y[carrier == 1]) -
              mean(null.y[carrier == 0])
alt.diff <- mean(alt.y[carrier == 1]) -
              mean(alt.y[carrier == 0])
```

With 10000 replications, this takes approx. 2 seconds to run on Box.

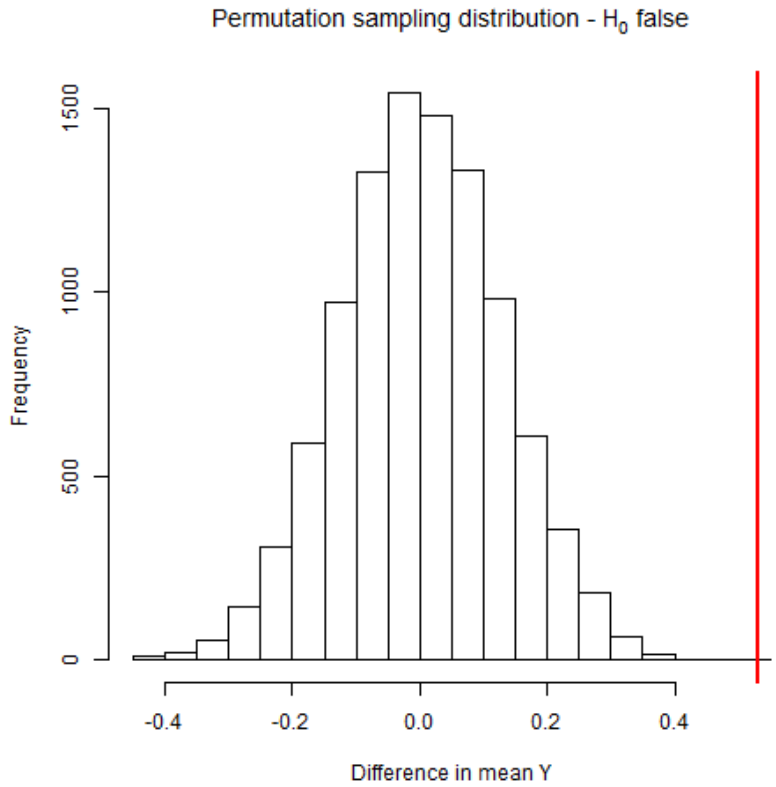
Example: permutation test vs bootstrap (H_0 true)

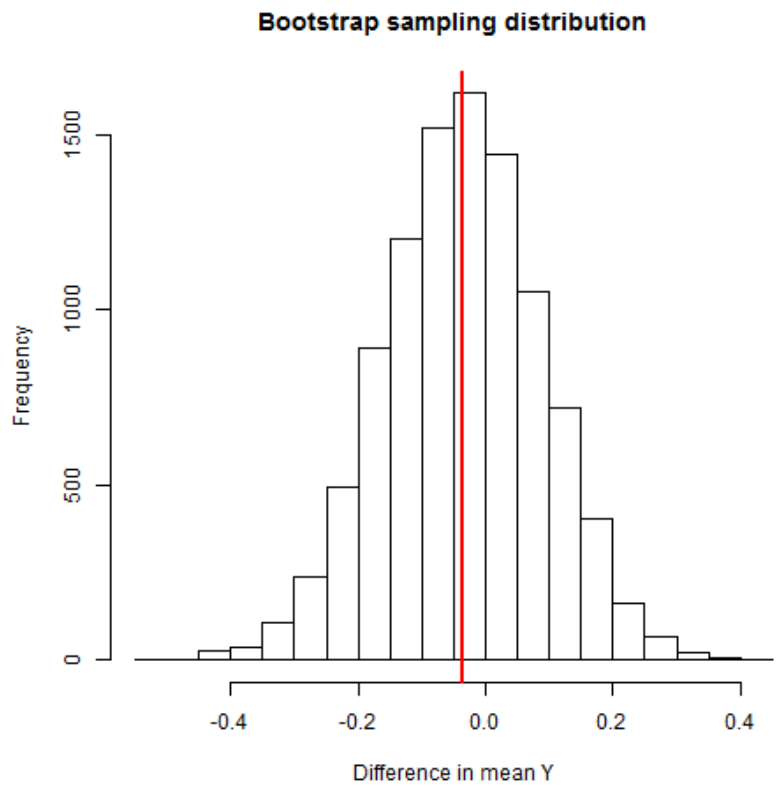




p-value (computed as proportion greater than test statistic in original data) ≈ 0.77

Example: permutation test vs bootstrap (H_0 false)





p-value (computed as proportion greater than test statistic in original data) < 0.005

Running GUI-less

Three basic things to remember:

- You have to set a seed!
  - Without using `set.seed()`, you will not be able to reproduce output if necessary; either you won't know where the random number generator started, or you will get the same output for each script
- R will not automatically save any objects created during the session — you have to do this manually
- You have to tell the computer where to look for the executable file to run your code

Bottom line: **Set the seed in a logical, memorable way such that each script gets a different seed. Also, save output that you care about to memory.**

Running GUI-less: Rscript

While there are a few ways to run R GUI-less, `Rscript.exe` is one of the most common.

First, include where `Rscript.exe` resides on your machine. On Box (and most machines), it is in `/usr/local/bin/Rscript`.

Put `#!/usr/local/bin/Rscript` at the top of your `.R` file so that the computer knows how to run the code.

The "shebang", `#!`, is common in GUI-less programming. This type of statement at the top of a file tells the computer where to look to run the file (more on this in shell scripting).

Running GUI-less: running the script

Doing this on Mac or Linux is easy — simply navigate to the file and run it, e.g.,

```
cd "C:/Users/brianw26/Dropbox/Courses/2016-2017 Third Year/Presentations/BIOST 561 - Lecture"
Rscript permutation_test_example.R
```

On Windows:

- Need to set the PATH (only once) to tell the machine where to look for `Rscript.exe`
- Typing `path` in the command line brings up the PATH, typing `C:\Program Files\R\R-3.2.1\bin\x64;%PATH%` (or wherever your R install is) will add this to the PATH

Then run with

```
cd "C:\Users\brianw26\Dropbox\Courses\2016-2017 Third Year\Presentations\BIOST 561 - Lecture"
Rscript permutation_test_example.R
```

Running GUI-less: parameters

In the permutation test example, we had three parameters: n (sample size), B (number of Monte Carlo replications), and the seed.

That is more clear in this function:

```
permTestSNP <- function(n = 300, B = 10000) {
carrier <- rep(c(0, 1), c(n/3, 2*n/3)) ## make the genotypes (0/1)
  null.y <- rnorm(n) ## make y under the null hypothesis
  alt.y <- rnorm(n, mean = carrier/2) ## make y under the alternative
  output.truenull <- replicate(B, oneTest(carrier, null.y)) ## run the test B t
  output.falsenull <- replicate(B, oneTest(carrier, alt.y))
  ## get the observed difference in means
  null.diff <- mean(null.y[carrier == 1]) - mean(null.y[carrier == 0])
  alt.diff <- mean(alt.y[carrier == 1]) - mean(alt.y[carrier == 0])
  ## return the sampling distribution and the p-values
  return(list(samp.truenull = output.truenull, samp.falsenull = output.falsenul
              p.truenull = mean(abs(output.truenull)) > abs(null.diff),
              p.falsenull = mean(abs(output.falsenull)) > abs(alt.diff))
         )
}
```

Running GUI-less: setting the seed

Rscript lets us pass arguments to our R session when we run a script.

However, we need a bit of pre-processing to read that in:

```
#!/usr/local/bin/Rscript
## read in the parameter(s)
arg <- commandArgs(TRUE)
## gets read in as a character, change to numeric
myseed <- as.numeric(arg)

permTest <- function(n = 300, B = 10000) {
 <code omitted, see earlier slides>
}
set.seed(myseed)
output <- permTest(n = 300, B = 10000)
```

At the command line, enter e.g., `Rscript permTest.R 47` to run `permTest` with $n = 300$, $B = 10000$, and seed $= 47$.

Running GUI-less: setting multiple parameters

Sometimes we have multiple parameters, e.g., n, B, and seed.

We can read in multiple from Rscript:

```
args <- commandArgs(TRUE)
if (length(args) == 0) {
  print("No arguments supplied.")
} else {
  for (i in 1:length(args)) {
    eval(parse(text = args[i]))
  }
}
if (!exists("myn")) {
  myn <- 100
  cat("Note: assuming n = 100")
}
if (!exists("myseed")) {
  myseed <- 47
  cat("Note: setting seed to be 47")
}
if (!exists("myB")) {
  myB <- 1000
  cat("Note: running 1000 replications")
}
```

Running GUI-less: setting multiple parameters

We then pass Rscript multiple parameters, e.g., `Rscript permTest.R myn=300 myseed=47 myB=10000`

The code on the previous slide makes sure that You can debug Your output.

Remember that `commandArgs()` reads in characters! This is a common source of error — we don't want to make R angry.

Running GUI-less: saving output

`write.table()` and `write.csv` are useful for large output datasets.

`save()` writes an R-friendly version of an object (or objects) to a designated `file`; read these in with `load()`.

"How much/what to save?" — depends on what you need!

- More output can make debugging easier, but wastes memory and/or time
- Thinking through which output you want to save can make your results cleaner

Running GUI-less: saving output

We might have, at the bottom of our script file,

```
<...>
set.seed(myseed)
output <- permTest(n = myn, B = myB)
save(output, paste("perm_test_output_n_", myn,
"_s_", myseed, "_B_", myB, ".Rdata", sep = ""))
```

This creates a `.Rdata` file in the current working directory (where you called `Rscript` from), and is called e.g., `perm_test_output_n_300_s_47_B_10000.Rdata`

Running GUI-less: some advice

Work up code slowly!

- Make sure your base code (e.g., `oneTest`) works correctly
- Make sure the loop works (usually in GUI) (e.g., `permTest` with small B)
- Make sure the looping and parameter setting works in Rscript, with small B (check output!)

Once your code passes all of these checks, *then* run with a large B.