resampling

J. Di Iorio, F. Chiaromonte

3/10/2021

Libraries

We are going to use **tidyverse** and **ggplot2**.

```r
library(tidyverse) # for data manipulation and visualization
```

```
## -- Attaching packages --------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(ggplot2) # for plots
library(boot) # for bootstrapping
library(coin) # for permutation tests
```

```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:boot':
##
##     aml
```

Data

We will not use already available datasets but we will simulate them.

Bootstrapping

In principle, there are two different ways of obtaining and evaluating bootstrap estimates:

1. non-parametric;
2. parametric;

- Goal: we have a set of n observations from which we have calculated some statistic θ, for which we have no formula to estimate a standard error, but to which we wish to attach (ordinary 2-tailed 95%) confidence interval.

Non-parametric Bootstrapping

- Why non-parametric? we cannot reasonably assume our sample to represents a known frequency distribution, but we can assume it adequately reflects the wider population from which it was drawn.

By hand

We have a sample from a binomial distribution with parameters $(15, 0.71)$.

```
set.seed(123)
x <- rbinom(n=30, size=15, prob=0.71)
x
```

```
##  [1] 12  9 11  9  8 13 11  8 10 11  8 11 10 10 13  8 12 13 11  8  8 10 10  6 10
## [26] 10 11 10 12 12
```

Now, Let us assume we do not know the distribution. We want to find out the 90th percentile. We use non-parametric bootstrapping.

```
sample_size <- length(x)
B <- 2000 #number of samples
tempdata <- c()
for(i in 1:B)
  tempdata <- c(tempdata, sample(x,sample_size,replace=TRUE))

bootstrapsample = matrix(tempdata, nrow = sample_size, ncol = B)
```

Now we can compute on each of the 2000 samples the statistic - producing B bootstrap values. In our case the statistic we want to compute is the 90th percentile.

```
B_values <- apply(bootstrapsample, 2, quantile, prob=0.9)
head(B_values)
```

```
## [1] 12.0 13.0 12.0 12.1 11.1 12.0
```

So we have the following estimate and standard error:

```
mean(B_values)
```

```
## [1] 12.3326
```

```
sd(B_values)
```

```
## [1] 0.5549606
```

Using boot

We can automatically perform non-parametric bootstrapping using the **boot** package. The main bootstrapping function is boot( ) and has the following format:

```
help(boot)
```

1. **data:** The data as a vector, matrix or data frame. If it is a matrix or data frame then each row is considered as one multivariate observation;
2. **statistic:** A function which when applied to data returns a vector containing the statistic(s) of interest;
3. **R:** The number of bootstrap replicates;

In the **statistic** field it is mandatory to pass an estimation function. In the case of the 90th percentile, our estimation function is:

```
sampleperc <- function(x, d) {
  return(quantile(x[d], prob=0.9))
}
```

The estimation function (that you write) consumes data x and a vector of indices d. This function will be called many times, one for each bootstrap replication. Every time, the data x will be the same, and the bootstrap sample d will be different.

Once we have written a function like this, here is how we would obtain bootstrap estimates of the standard deviation of the distribution of the 90th percentile:

```
b = boot(x, sampleperc, R=2000)
print(b)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = x, statistic = sampleperc, R = 2000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     12.1  0.2271   0.5598145
```

It is also easy to get a confidence interval (be careful!) using the function **boot.ci** that requires an object of class "boot" (computed using **boot**). The function generates 5 different types of equi-tailed two-sided nonparametric confidence intervals. These are the first order normal approximation, the basic bootstrap interval, the studentized bootstrap interval (bootstrap variance needed), the bootstrap percentile interval, and the adjusted bootstrap percentile (BCa) interval.

```
boot.ci(b, conf=0.95)
```

```
## Warning in boot.ci(b, conf = 0.95): bootstrap variances needed for studentized
## intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, conf = 0.95)
##
## Intervals :
## Level      Normal              Basic
## 95%   (10.78, 12.97 )   (11.20, 13.10 )
##
## Level     Percentile            BCa
## 95%   (11.1, 13.0 )   (11.0, 13.0 )
## Calculations and Intervals on Original Scale
```

Parametric Bootstrapping

- Why parametric? we can reasonably assume our sample to represent a known frequency distribution.

By hand

We have a sample from a binomial distribution with parameters $(15, 0.71)$.

```
set.seed(123)
N <- 15
x <- rbinom(n = 30, size = N, prob = 0.71)
x
```

```
##  [1] 12  9 11  9  8 13 11  8 10 11  8 11 10 10 13  8 12 13 11  8  8 10 10  6 10
## [26] 10 11 10 12 12
```

Now, Let us assume we know the distribution but we do not know the parameter p and we want to estimate the 90th percentile. We use MLE ($\hat{p} = \frac{\sum_{i=1}^{n} x_i}{nN}$)

```
bin_size <- 15
sample_size <- length(x)
p_hat <- mean(x)/N
```

We use parametric bootstrapping and we sample B samples of $sample_size$ observations from the known distribution.

```
B <- 2000 #number of samples
tempdata <- rbinom(B*sample_size, size = bin_size, prob = p_hat)
bootstrapsample = matrix(tempdata, nrow = sample_size, ncol = B)
```

Now we can compute on each of the 2000 samples the statistic - producing B bootstrap values. In our case the statistic we want to compute is the 90th percentile.

```
B_values <- apply(bootstrapsample, 2, quantile, prob=0.9)
head(B_values)
```

```
## [1] 12.0 12.0 12.0 12.1 12.0 11.1
```

So we have the following estimate and standard error:

```
mean(B_values)
```

```
## [1] 12.2717
```

```
sd(B_values)
```

```
## [1] 0.5362862
```

Permutation Test

Permutation tests are particularly relevant in experimental studies, where we are often interested in the sharp null hypothesis of no difference between treatment groups.

Let's generate our data divided in treatment group (1) and control group (0).

```r
set.seed(1)
n <- 100
tr <- rbinom(n, 1, 0.5)
y <- 1 + tr + rnorm(n, 0, 3)
```

Let us compute the difference in mean between the two groups. The difference in means is, as we would expect (given we made it up), about 1:

```r
means <- by(y, tr, mean)
diff0 <- diff(means)
```

To obtain a single permutation of the data, we simply resample without replacement and calculate the difference again:

```r
s <- sample(tr, length(tr), FALSE)
by(y, s, mean)
```

```
## s: 0
## [1] 0.8112026
## ------------------------------------------------------------
## s: 1
## [1] 2.094659
```

```r
diff(by(y, s, mean))
```

```
## [1] 1.283456
```

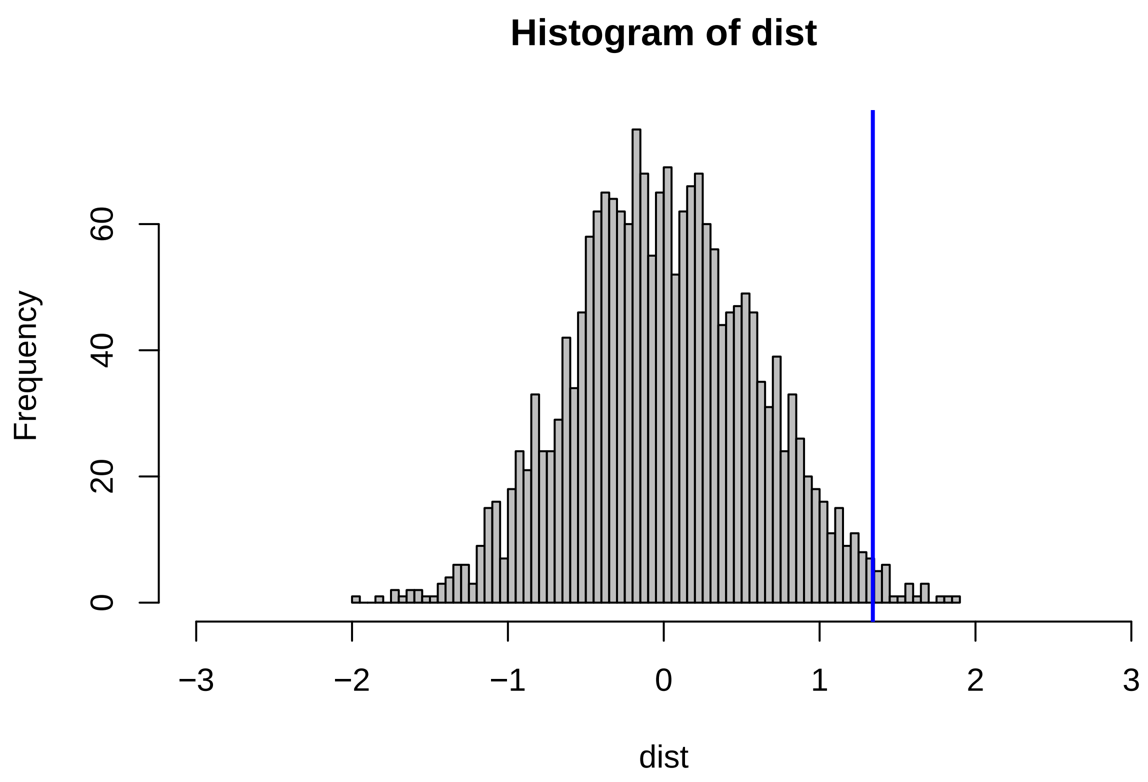
If we repeat this process a large number of times, we can build our approximate permutation distribution (i.e., the sampling distribution for the mean-difference). We'll use **replicate** to repeat our permutation process. The result will be a vector of the differences from each permutation (i.e., our distribution):

```r
dist <- replicate(2000, diff(by(y, sample(tr, length(tr), FALSE), mean)))
head(dist)
```

```
##          1          1          1          1          1          1
## -0.5879742  0.6944840  0.8145251 -0.4230718 -0.1443668  0.1122164
```

We can look at our distribution using an histogram indicating a vertical line for our observed difference:

```r
hist(dist, xlim = c(-3, 3), col = "grey", breaks = 100)
abline(v = diff(by(y, tr, mean)), col = "blue", lwd = 2)
```



Now, we can use the distribution to obtain a p-value for our mean-difference by counting how many permuted mean-differences are larger than the one we observed in our actual data. We can then divide this by the number of items in our permutation distribution (i.e., 2000 from our call to replicate, above):

```
sum(dist > diff0)/2000 # one-tailed test
```

```
## [1] 0.012
```

```
sum(abs(dist) > abs(diff0))/2000  # two-tailed test
```

```
## [1] 0.021
```

Permutation Test with coin

Even if we showed as to create your own permutation distributions, R provides a package to conduct permutation tests called **coin**. We can compare our result from above with the result from **coin**:

```
library(coin)
independence_test(y ~ tr, alternative = "greater")  # one-tailed
```

```
##
##  Asymptotic General Independence Test
##
## data:  y by tr
## Z = 2.3154, p-value = 0.01029
## alternative hypothesis: greater
```

```
independence_test(y ~ tr)  # two-tailed
```

```
##
##  Asymptotic General Independence Test
##
```

```
## data:  y by tr
## Z = 2.3154, p-value = 0.02059
## alternative hypothesis: two.sided
```

Almost anything that you can address in a parametric framework can also be done in a permutation framework otherwise you can create your own proper permutation test!