SLLD - Module 1

Resampling

S.Tonini, F.Chiaromonte

Sant'Anna School of Advanced Study - Pisa

22/2/2024

```
library(boot) #Bootstrap Functions
library(coin) #Conditional Inference Procedures in a
# Permutation Test Framework
library(ggplot2) #Create Elegant Data Visualisations Using
# the Grammar of Graphics
```

Data

We generate a sample from a binomial distribution with parameters (25, 0.6).

```
options(width = 60)
set.seed(123)
x <- rbinom(n=50,       # sample size
            size=25,    # num. of trials
            prob=0.6)  # prob. of success per trial
x
```

```
##  [1] 16 13 16 12 11 19 15 12 15 15 11 15 14 15 18 12 17 19
## [19] 16 11 12 14 14  9 14 14 15 14 16 18 11 12 14 13 20 15
## [37] 13 17 16 17 18 16 16 16 18 18 17 15 17 12
```

```
n <- length(x)
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   13.00   15.00   14.86   16.75   20.00
```

Let's pretend that we do not know the true underlying distribution.

Our goal is to estimate the 95th percentile, and we rely on **non-parametric bootstrapping**.

```
set.seed(123)
# number of bootstrap samples
B <- 1000
# initialize an empty "container" for each of these
bootstrapsample <- matrix(NA,
nrow = n,
ncol = B)
dim(bootstrapsample)
```

```
## [1]   50 1000
```

```
bootstrapsample[1:3, 1:3]
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
```

We use the function **sample** that takes a sample of the specified size from the elements of x using either with or without replacement.

```
for(i in 1:B){
# assign (column-wise) B draws with replacement
bootstrapsample[, i] <- sample(x,n,replace=TRUE)
}
summary(bootstrapsample[, 1:3])
```

```
##        V1              V2              V3
##  Min.   :11.00   Min.   :11.00   Min.   : 9.00
##  1st Qu.:14.00   1st Qu.:14.00   1st Qu.:13.00
##  Median :15.00   Median :15.00   Median :15.00
##  Mean   :14.88   Mean   :15.04   Mean   :14.94
##  3rd Qu.:16.00   3rd Qu.:17.00   3rd Qu.:17.00
##  Max.   :20.00   Max.   :20.00   Max.   :20.00
```

Now we can compute our statistic of interest on each of these 1000 samples – producing B bootstrap values. In our case, the statistic we are interested in is the 95th percentile.

We use the function **apply** that allows us to apply some function column-wise, and the function **quantile** that produces sample quantiles corresponding to the given probabilities.

```
B_values <- apply(bootstrapsample, # data
                  2, # dimension (i.e. column-wise)
                  quantile, # function to apply
                  probs=0.95) # input for the function
head(B_values)
```

```
## [1] 18.00 19.00 18.00 18.00 18.55 19.00
```

So we have the following point estimate and its standard error:

```
results <- data.frame("mean" = mean(B_values),
"SD" = sd(B_values))
rownames(results) <- "Manual"
results
```

```
##           mean        SD
## Manual 18.5187 0.5442946
```

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

- **data:** The data as a vector, matrix or data frame.
- **statistic:** A function which when applied to data returns a vector containing the statistic(s) of interest.
- **R:** The number of bootstrap replicates.

Remark: it is mandatory to pass a"user-defined function" in the field statistic. In the case of the 95th percentile, our estimation function is:

```
# x: vector
# d: set of indexes
# prob: quantile
sampleperc <- function(x, d, prob=0.95) {
return(quantile(x[d], probs=prob))
}
```

Once we have written a function like this, here is how we obtain bootstrap estimates of the standard error for the 95th percentile of the distribution:

```
set.seed(123)
b = boot(x, sampleperc, R=1000)
print(b)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = x, statistic = sampleperc, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*    18.55 -0.03435   0.5642082
```

notice how close this is to our previous computation

```
results <- rbind.data.frame(results,
data.frame("mean" = b$t0,
"SD" = sd(b$t)))
rownames(results)[2] <- "boot"
results
```

```
##           mean        SD
## Manual 18.5187 0.5442946
## boot   18.5500 0.5642082
```

```
#bias calculation
mean(b$t)-b$t0
```

```
##      95%
## -0.03435
```

It is also easy to get a confidence interval (but be careful) using the function **boot.ci** that requires an object of class “boot” (i.e. computed using boot).

This function generates (by default) 5 different types of equi-tailed two-sided nonparametric confidence intervals. These are:

- **first order normal approximation:** uses the standard deviation for calculation of CI. Use when statistic is unbiased. Is normally distributed.
- **basic bootstrap interval:** uses percentile to calculate upper and lower limit of test statistic. Use when statistic is unbiased and homoscedastic. The bootstrap statistic can be transformed to a standard normal distribution
- **studentized bootstrap interval (bootstrap variance needed):** resamples the bootstrap sample to find a second-stage bootstrap statistic and use it to calculate the CI. Use when statistic is homoscedastic. The standard error of bootstrap statistic can be estimated by second-stage resampling.
- **bootstrap percentile interval:** uses quantiles eg 2.5%, 5% etc. to calculate the CI. Use when statistic is unbiased and homoscedastic. The standard error of your bootstrap statistic and sample statistics are the same.
- **adjusted bootstrap percentile (BCa) interval:** uses percentile limits with bias correction and estimate acceleration coefficient corrects the limit and find the CI. The bootstrap statistic can be transformed to a normal distribution. The normal-transformed statistic has a constant bias.

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

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, conf = 0.95)
##
## Intervals :
## Level      Normal              Basic
## 95%   (17.48, 19.69 )   (17.55, 19.55 )
##
## Level     Percentile            BCa
## 95%   (17.55, 19.55 )   (17.55, 19.55 )
## Calculations and Intervals on Original Scale
```

Parametric Bootstrapping

We have a sample of size n = 50 from a binomial distribution with parameters (N = 25, p = 0.6).

```
set.seed(123)
n = 50
N <- 25
x <- rbinom(n = n, size = N, prob = 0.6)
```

Let's assume that we know the underlying distribution, where the actual parameter p is unknown, and we want to estimate the 95th percentile as before.

```
p_hat <- mean(x)/N
p_hat
```

```
## [1] 0.5944
```

We use parametric bootstrap and compute B samples of size n from the known distribution.

```
B <- 1000 # number of bootstrap samples
tempdata <- rbinom(B*n,
size = N,
prob = p_hat)
bootstrapsample <- matrix(tempdata, nrow = n, ncol = B)
dim(bootstrapsample)
```

```
## [1]   50 1000
```

```
bootstrapsample[1:3, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   19   14   12   17   16
## [2,]   15   16   15   10   17
## [3,]   13   15   16   14   14
```

Now we can compute the statistic of interest on each of these 1000 samples – producing B bootstrap values. In our case, we are interested in the 95th percentile.

```
B_values <- apply(bootstrapsample, 2, quantile, prob=0.95)
summary(B_values)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   18.00   18.55   18.60   19.00   20.55
```

So we have the following estimate and standard error:

```
resultsParam <- data.frame("mean" = mean(B_values),
"SD" = sd(B_values))
rownames(resultsParam) <- "Manual_Parametric"
resultsParam
```

```
##                          mean        SD
## Manual_Parametric 18.60495 0.6487479
```

Let's compare it with the boot function:

```
set.seed(123)
B = 1000
p.rg <- function(data, mle, N=25) {
out <- rbinom(length(data),
size = N,
prob = mle)
out
}
bBoot <- boot(x, sampleperc, R = B, sim = "parametric",
ran.gen = p.rg, mle = (mean(x)/N))
```

```
bBoot
```

```
##
## PARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = x, statistic = sampleperc, R = B, sim = "parametric",
##     ran.gen = p.rg, mle = (mean(x)/N))
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    18.55  0.0555   0.6484676
```

```
# notice how close this is to our previous computation
resultsParam <- rbind.data.frame(resultsParam,
data.frame("mean" = bBoot$t0,
"SD" = sd(bBoot$t)))
rownames(resultsParam)[2] <- "boot_Parametric"
resultsParam
```

```
##                       mean        SD
## Manual_Parametric 18.60495 0.6487479
## boot_Parametric   18.55000 0.6484676
```

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 a dataset divided into treatment (1) and control group (0).

```
# they have a difference in mean equal to 1
set.seed(1)
n <- 80
tr <- rbinom(n, 1, 0.3)
y <- 1 + tr + rnorm(n, 0, 2)
```

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

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

```
## [1] 1.198222
```

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

```
s <- sample(tr, length(tr), FALSE) # shuffle the labels
by(y, s, mean) # compute mean in vector y according to class s
```

```
## s: 0
## [1] 1.732339
## ------------------------------------------------
## s: 1
## [1] 1.147594
```

```
diff(by(y, s, mean)) # difference between 2 means
```

```
## [1] -0.584745
```

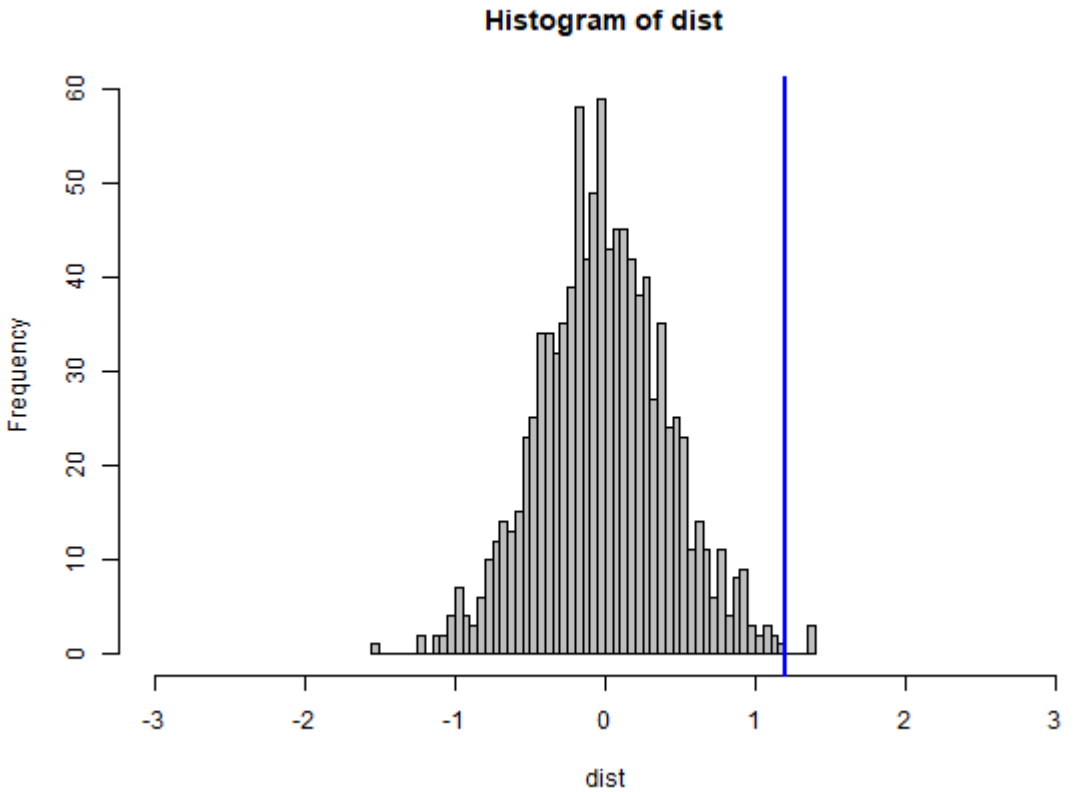
If we repeat this process a large number of times, we can build our approximate permutation distribution (i.e., the sampling distribution for the mean-difference). We use the function **replicate** to repeat our permutation process.

```
set.seed(123)
nperm = 1000
# nperm = 200 # try this too :)
# nperm = 5000
dist <- replicate(nperm, diff(by(y, sample(tr, length(tr),
                                           FALSE), mean)))
head(dist)
```

```
##           1           1           1           1
##  0.158136857 -0.128001168 -0.455300572 -1.004146337
##           1           1
##  0.007316022  0.094566325
```

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

```
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., nperm=1000 from our call to replicate, above):

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

```
## [1] 0.003
```

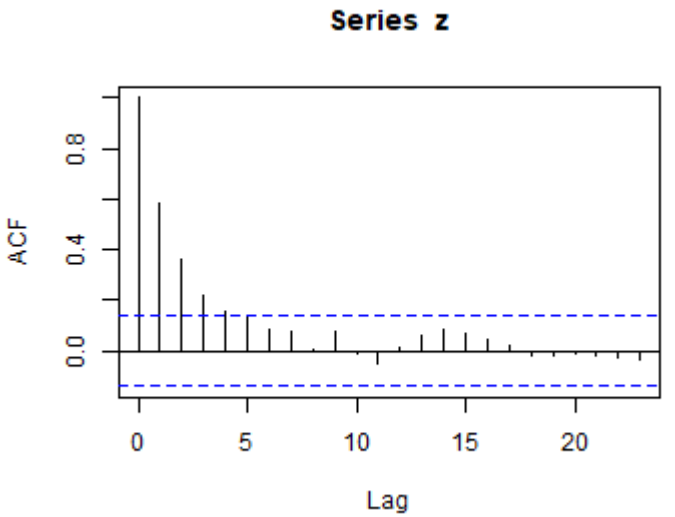
```
sum(abs(dist) > abs(diff0))/nperm # two-tailed test
```

```
## [1] 0.006
```

Time Series Bootstrap

The function **tsboot** (in the **boot package**) generates R bootstrap replicates of a statistic applied to a time series. The replicate time series can be generated using fixed or random block lengths or can be model based replicates.

```
# they have a difference in mean equal to 1
set.seed(1)
z <- arima.sim(n = 200, list(ar = 0.6))
acf(z)
```



```
ar.fun <- function(fun) {
     ar.fit <- ar(fun, order.max = 1)
     c(ar.fit$ar[1])
}
prova <- tsboot(z, ar.fun, R = 1000, l = 6, sim = "fixed")
prova$t0
```

```
## [1] 0.5804858
```

```
ar.fun(z)
```

```
## [1] 0.5804858
```

```
prova$t[1:5]
```

```
## [1] 0.4245009 0.4276346 0.4033080 0.4816728 0.4838219
```

```
sd(prova$t)
```

```
## [1] 0.05240388
```

Now it's your turn!!!