Bootstrap with Library boot

Stats 102A

Miles Chen

Department of Statistics

Week 9 Friday

UCLA

Section 1

Package boot

Package boot

The R package `boot` provides some useful utilities beyond the basics of creating a bootstrapped sampling distribution.

The primary function is `boot()` which can perform both non-parametric and parametric bootstrap.

```
# install.packages("boot")
library(boot)
```

Help for boot()

Description

Generate R bootstrap replicates of a statistic applied to data. Both parametric and nonparametric resampling are possible. For the nonparametric bootstrap, possible resampling methods are the ordinary bootstrap, the balanced bootstrap, antithetic resampling, and permutation. For nonparametric multi-sample problems stratified resampling is used: this is specified by including a vector of strata in the call to boot. Importance resampling weights may be specified.

Usage

```
boot(data, statistic, R, sim = "ordinary", stype = c("i", "f", "w"),
     strata = rep(1,n), L = NULL, m = 0, weights = NULL,
     ran.gen = function(d, p) d, mle = NULL, simple = FALSE, ...,
     parallel = c("no", "multicore", "snow"),
     ncpus = getOption("boot.ncpus", 1L), cl = NULL)
```

Help for boot()continued

`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.

`statistic`

A function which when applied to data returns a vector containing the statistic(s) of interest. ... `statistic` must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices, frequencies or weights which define the bootstrap sample.

`R`

The number of bootstrap replicates. Usually this will be a single positive integer.

The function `boot()`

The function `boot()` requires the user to provide at least three arguments:

- the data that will be resampled
- the statistic to be calculated (a function that we define)
- the number of replicates R

You can specify other arguments, but they all have default values associated with them.

`boot()` resamples indices

`boot()` performs bootstrap resampling is a clever way. Rather than resampling the actual values in the dataset, `boot()` resamples only the **indices** of the data.

It then subsets the data based on these resampled indices.

On the next two slides I provide some code that illustrates how resampling the values and resampling the indices are equivalent.

Resampling values vs resampling indices

To illustrate bootstrap resampling, let's pretend our population consists of the 26 letters of the alphabet.

In the following code chunk I draw 10 letters with replacement from the population of 26 letters.

```
population <- letters
set.seed(5)
sample(population, 10, replace = TRUE)
```

```
##  [1] "b" "k" "y" "o" "k" "y" "i" "u" "g" "w"
```

In the following code chunk I draw 10 numbers with replacement from the vector 1:26. I then use the numbers as indices for subsetting the letters. With the same starting seed, the results are the same.

```
set.seed(5)
indices <- sample(1:26, 10, replace = TRUE)
print(indices)
```

```
##  [1]  2 11 25 15 11 25  9 21  7 23
```

```
print(population[indices])
```

```
##  [1] "b" "k" "y" "o" "k" "y" "i" "u" "g" "w"
```

Section 2

Example

Iris data

The iris data set is very popular data set. It has 150 observations, 4 numeric predictor variables, and one categorical variable with three categories (species) that can be predicted.

Statistician Ronald Fisher (after whom the Fisher exact test and F distribution are named) used the iris flower data set in a 1936 paper on linear discriminant analysis.

It has been a staple in statistics since.

```
data(iris)
print(iris)
```

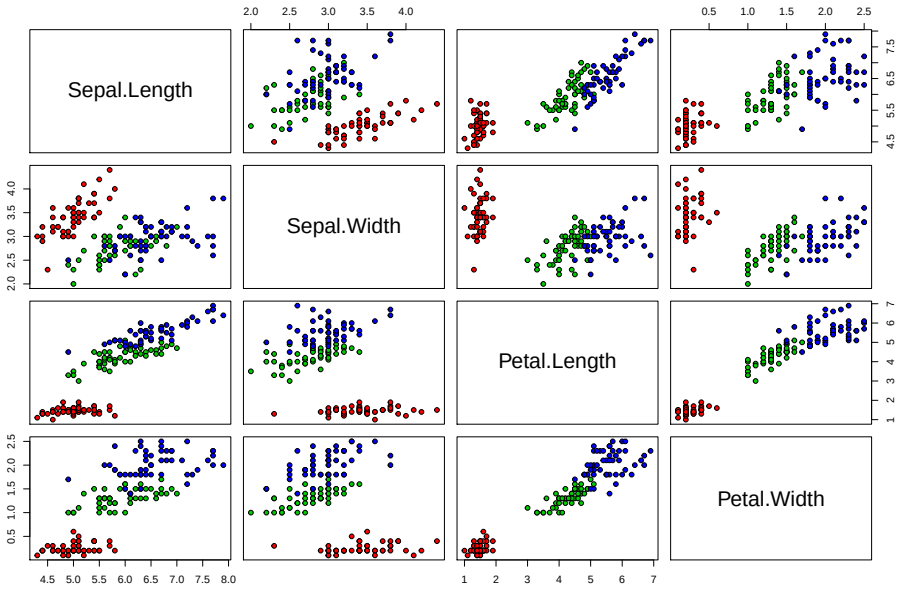
```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
```

Summary stats

```
iris %>%
  group_by(Species) %>%
  summarise(mSepalLen = mean(Sepal.Length),
            sSepalLen = sd(Sepal.Length),
            mSepalWid = mean(Sepal.Width),
            sSepalWid = sd(Sepal.Width),
            mPetalLen = mean(Petal.Length),
            sPetalLen = sd(Petal.Length),
            mPetalWid = mean(Petal.Width),
            sPetalWid = sd(Petal.Width),
            n = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 10
##   Species mSepalLen sSepalLen mSepalWid sSepalWid mPetalLen sPetalLen mPetalWid sPetalWid     n
##   <fct>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl> <int>
## 1 setosa       5.01     0.352      3.43     0.379      1.46     0.174     0.246     0.105    50
## 2 versic~      5.94     0.516      2.77     0.314      4.26     0.470     1.33      0.198    50
## 3 virgin~      6.59     0.636      2.97     0.322      5.55     0.552     2.03      0.275    50
```



Hypothesis Test

Let's say we want to see if the sepal length differs for the species setosa and the species versicolor. In the dataset, the first setosa is the first 50 observations and versicolor are observations 51-100.

We can conduct a simple t-test to compare the mean length of the first 50 sepal length values (Setosa) and the next 50 sepal length values (Versicolor)

The results show that there is a significant difference between the means.

```
t.test(iris$Sepal.Length[1:50], iris$Sepal.Length[51:100])
```

```
##
##  Welch Two Sample t-test
##
## data:  iris$Sepal.Length[1:50] and iris$Sepal.Length[51:100]
## t = -10.521, df = 86.538, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.1057074 -0.7542926
## sample estimates:
## mean of x mean of y
##     5.006     5.936
```

Code Contrast: resampling the values directly

For a bootstrap test, the null hypothesis is that the values come from the same population. To create our bootstrap "population" we would take the first 100 values and duplicate them infinitely many times, or equivalently we would resample the first 100 values with replacement.

If we want to resample the sepal.length values in the Iris data directly, we could do the following:

```
set.seed(1)
differences <- rep(NA, 10^4)
for(i in seq_along(differences)){
  boot_population <- iris$Sepal.Length[1:100]
  groupA <- sample(boot_population, 50, replace = TRUE)
  groupB <- sample(boot_population, 50, replace = TRUE)
  differences[i] <- mean(groupA) - mean(groupB)
}
summary(differences)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.4700000 -0.0840000  0.0000000  0.0002698  0.0865000  0.5520000
```

Code Contrast: resampling the indices and subsetting

In contrast, we can also choose to resample indices instead.

In this setup, we sample the index values 1:100 with replacement and use those values to "subset" the iris data. From the resampled data, we put the first 50 values of the sepal length column into groupA and the next 50 values in to groupB.

```
set.seed(1)
differences_b <- rep(NA, 10^4)
for(i in seq_along(differences_b)){
  resampled_indices <- sample(1:100, size = 100, replace = TRUE)
  resampled_data <- iris[resampled_indices, ]
  groupA <- resampled_data$Sepal.Length[1:50]
  groupB <- resampled_data$Sepal.Length[51:100]
  differences_b[i] <- mean(groupA) - mean(groupB)
}
summary(differences_b)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.4700000 -0.0840000  0.0000000  0.0002698  0.0865000  0.5520000
```

```
all.equal(differences, differences_b) # resampling indexes produces the same results
```

```
## [1] TRUE
```

Using `boot()`: the `statistic` function

The bootstrap function in library `boot` performs bootstrap by resampling indices. To use the function `boot()`, you must write a function that calculates the statistic of interest with resampled indices.

This statistic function we write takes in two arguments: the data and the indices. Function `boot()` performs the resampling of indices and passes them along as an argument in the function.

In our case, we wish to calculate the difference between the mean of sepal length of 50 setosa flowers vs the mean sepal length of 50 versicolor flowers.

Our `statistic` function for use with `boot()`

Here is the function we would write to use with `boot()`. We can see the code is quite similar to the loop we wrote when we resampled indices. Our function **does not need to perform the resampling of indices**. The `boot()` function will perform the resampling of indices and will pass the index values to this function we write.

```
dif_func <- function(data, index){
  resampled_data <- data[index, ]
  groupA <- resampled_data$Sepal.Length[1:50]
  groupB <- resampled_data$Sepal.Length[51:100]
  difference <- mean(groupA) - mean(groupB)
  difference
}
```

Bootstrap resampling with boot()

```
set.seed(1)
boot_results <- boot(data = iris[1:100, ], statistic = dif_func, R = 10000)
```

Running `boot()` is simple.

- We provide the name of the data frame that will be resampled. In our case, we use the first 100 rows of the data frame `iris`. We limit ourselves to the first 100 rows because we are comparing only the sepal length of setosa with versicolor.
- We provide the name of the statistic function that we wrote, `dif_func`
- We provide how many replications to perform

The results of boot()

```
str(boot_results)
```

```
## List of 11
##  $ t0       : num -0.93
##  $ t        : num [1:10000, 1] -0.092 -0.348 -0.146 -0.096 0.066 ...
##  $ R        : num 10000
##  $ data     :'data.frame':   100 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:100] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:100] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:100] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:100] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ seed     : int [1:626] 10403 624 -169270483 -442010614 -603558397 -222347416 1489374793 865871222 1
##  $ statistic:function (data, index)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 1 13 7 1 13 1 1 7
##   .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x00000000229e9ac8>
##  $ sim      : chr "ordinary"
##  $ call     : language boot(data = iris[1:100, ], statistic = dif_func, R = 10000)
##  $ stype    : chr "i"
##  $ strata   : num [1:100] 1 1 1 1 1 1 1 1 1 1 ...
##  $ weights  : num [1:100] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 ...
##  - attr(*, "class")= chr "boot"
##  - attr(*, "boot_type")= chr "boot"
```

Individual bootstrap results are stored in $t

Despite using the same starting seed, these individual values will not match the values from our own bootstrap results. The `boot()` function does resample with replacement, but does so in a different way and thus the randomized results will not match exactly.

```
summary(boot_results$t)
```

```
##        V1
##  Min.   :-0.434000
##  1st Qu.:-0.086000
##  Median : 0.002000
##  Mean   : 0.001596
##  3rd Qu.: 0.088000
##  Max.   : 0.506000
```

The empirical p-value

We can estimate the p-value of our boot results using the vector $t in the boot_results. This would be the equivalent of a vector that stores all of the resampled differences.

Note the observed difference can be found in the boot_results as $t0, which is the result if the original indexes were used directly (original data) and not the resampled indexes. In our case, it is the difference between the mean of indexes 1:50 and indexes 51:100

```
observed_dif <- 5.006 - 5.936 # from summary table
observed_dif
```

```
## [1] -0.93
```

```
boot_results$t0
```

```
## [1] -0.93
```

In 10,000 repetitions, none of the samples produced by random sampling had a difference greater than 0.93. Our empirical p-value is thus 0, which aligns closely with the results we saw in the t-test.

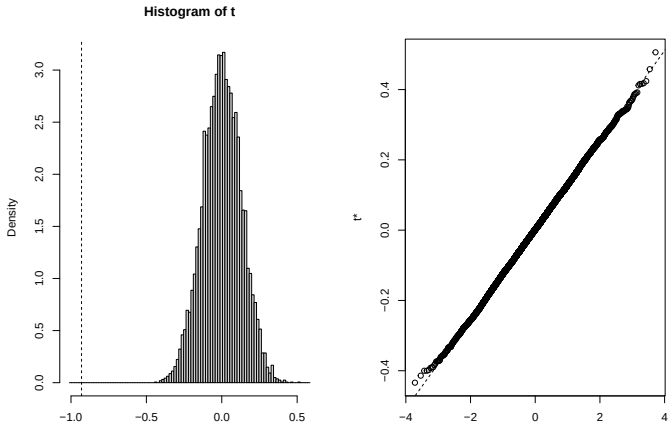
```
mean(abs(boot_results$t)  >= abs(observed_dif))
```

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

plot.boot()

The object returned by boot has its own plot method.

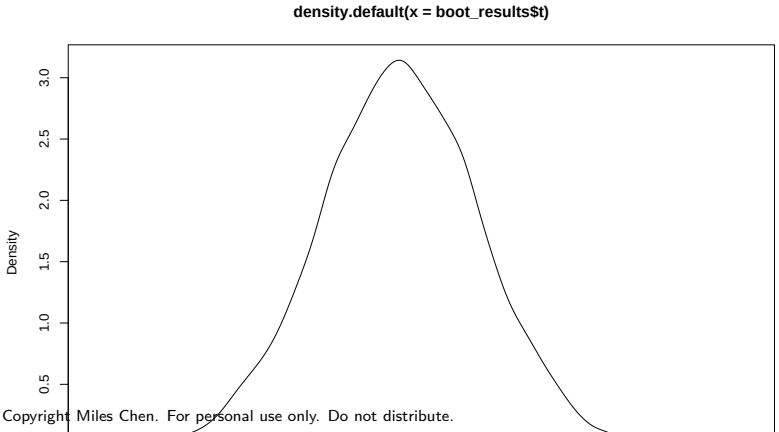
```
plot(boot_results)
```



Density curve of results

We can also produce an estimated density curve of the sampling distribution of the statistic that would be seen if the null hypothesis were true. As expected, the sampling distribution of $(\bar{y}_1 - \bar{y}_2)$ is centered at 0.

```
plot(density(boot_results$t))
```



Bootstrap Confidence Intervals

With `boot`, we can also create bootstrap confidence intervals. This page provides some interpretations for the different confidence intervals produced by `boot()`.
https://www.datacamp.com/community/tutorials/bootstrap-r

```
boot.ci(boot_results)
```

```
## Warning in boot.ci(boot_results): bootstrap variances needed for studentized intervals
```

```
## Error in bca.ci(boot.out, conf, index[1L], L = L, t = t.o, t0 = t0.o, : estimated adjustmen
```

Unfortunately, we get an error because our empirical p-value is 0.

Advantage of using boot

The advantage of using `boot` is that there are several functions and libraries that take advantage of the output of `boot`. Also, we can perform bootstrap for other, slighlty more complicated statistics.

Correlation

Let's say we wanted to do a bootstrap study on the correlation between sepal length and sepal width for all of the species in the iris data.

```
# sample correlation
cor(iris$Sepal.Length, iris$Sepal.Width)
```

```
## [1] -0.1175698
```

```
correlation <- function(data, indices){
  resampled_data <- data[indices,]
  cor(resampled_data[, 1], resampled_data[, 2]) # statistic that gets returned
}

boot_results <- boot(iris, correlation, R = 10000)
```

Boot results

```
str(boot_results)
```

```
## List of 11
##  $ t0       : num -0.118
##  $ t        : num [1:10000, 1] -0.2159 -0.0991 -0.0975 -0.2132 0.062 ...
##  $ R        : num 10000
##  $ data     :'data.frame':   150 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ seed     : int [1:626] 10403 251 251729535 -551786264 -922207485 1338312343 -731495824 1442613069 2
##  $ statistic:function (data, indices)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 4 16 7 1 16 1 4 7
##   .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x0000000014db2c80>
##  $ sim      : chr "ordinary"
##  $ call     : language boot(data = iris, statistic = correlation, R = 10000)
##  $ stype    : chr "i"
##  $ strata   : num [1:150] 1 1 1 1 1 1 1 1 1 1 ...
##  $ weights  : num [1:150] 0.00667 0.00667 0.00667 0.00667 0.00667 ...
##  - attr(*, "class")= chr "boot"
##  - attr(*, "boot_type")= chr "boot"
```

Boot results

```
summary(boot_results$t)
```

```
##        V1
##  Min.   :-0.36160
##  1st Qu.:-0.16936
##  Median :-0.11889
##  Mean   :-0.11874
##  3rd Qu.:-0.06961
##  Max.   : 0.19877
```

Bootstrapped confidence intervals

```
boot.ci(boot_results)
```

```
## Warning in boot.ci(boot_results): bootstrap variances needed for studentized intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_results)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-0.2608,  0.0280 )   (-0.2630,  0.0234 )
##
## Level     Percentile            BCa
## 95%   (-0.2586,  0.0278 )   (-0.2504,  0.0421 )
## Calculations and Intervals on Original Scale
```

Bootstrapped confidence intervals

The `boot.ci()` function provides 4 types of intervals

- The Normal CI uses the normal distribution. It uses 1.96 (for a 95% CI) times the estimated SE. The SE is estimated from the bootstrap results.
- The Percentile CI reports the 2.5th and 97.5th percentile value. In this case, it takes the 10,000 results, puts them in order, and reports the 250th and 9750th results.
- The Basic CI computes the differences between the results of each bootstrap replication and `t0`. It then reports percentiles based on that.
- The BCa interval is the bias-corrected accelerated interval from B. Efron. Details at: https://projecteuclid.org/journals/statistical-science/volume-11/issue-3/Bootstrap-confidence-intervals/10.1214/ss/1032280214.full