Boostrap Example

Mladen Kolar

Minimum Variance Portfolio

We revisit example from Section 5.2 of ISLR.

Given two assets X and Y, we want to invest α fraction of our money in X and the remaining $1-\alpha$ in Y. We want to choose α so that the portfolio

$$P = \alpha X + (1-\alpha)Y$$

has the minimum variance. The choice of α that makes the variance $\mathrm{Var}(P)$ the smallest is

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}},$$

where $\sigma_X^2 = \mathrm{Var}(X)$, $\sigma_Y^2 = \mathrm{Var}(Y)$, and $\sigma_{XY} = \mathrm{Cov}(X, Y)$.

We will estimate the unknown quantitites in the equation above to obtain $\hat{\alpha}$. We would also like to estimate the standard deviation for $\hat{\alpha}$.

The following function will generate pairs of observations from X and Y. We set $\sigma_X^2 = 1$, $\sigma_Y^2 = 1.25$, and $\sigma_{XY} = 0.5$.

```
library(MASS)
generate.data = function(n=100){
  # Parameters for bivariate normal distribution
  mu <- c(0,0) # Mean
  sigma <- matrix(c(1, 0.5, 0.5, 1.25), 2) # Covariance matrix
  data = mvrnorm(n, mu = mu, Sigma = sigma)
  colnames(data) <- c('x', 'y')
  as.data.frame(data)
}
```

The following function computes $\hat{\alpha}$ based on observations in the data frame `data`. The vector `index` indicates which rows of the data should be used for computation.

```
alpha.fn = function (data, index) {
  X = data$x[ index ]
  Y = data$y[ index ]
  var.x = var(X)
  var.y = var(Y)
  cov.xy = cov(X, Y)
  alpha = ( var.y - cov.xy ) / ( var.x + var.y - 2 * cov.xy )
  return(alpha)
}
```

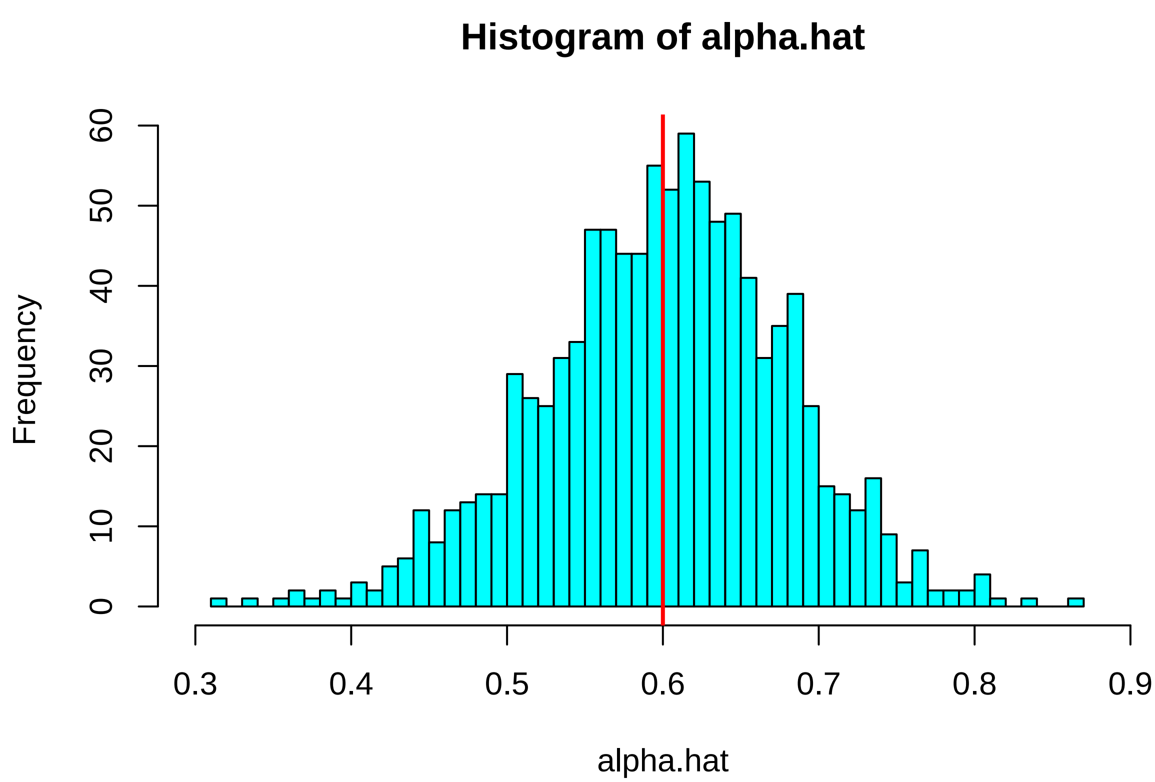
Let us now generate, a sample and compute $\hat{\alpha}$. For reproducibility, we set the seed.

```
set.seed(132)
data.init = generate.data(100)
alpha.fn(data.init, 1:100)
```

```
## [1] 0.601
```

Now let us repeat this process 1000 times.

```
alpha.hat = double(1000)
for(i in 1:1000) {
  data = generate.data(100)
  alpha.hat[i] = alpha.fn(data, 1:100)
}
hist(alpha.hat, breaks=50, col='cyan', xlim = c(0.3, 0.9))
abline(v=0.6, col='red', lwd=2)
```



The problem is that we only get one sample, and can compute only one estimate of $\hat{\alpha}$.

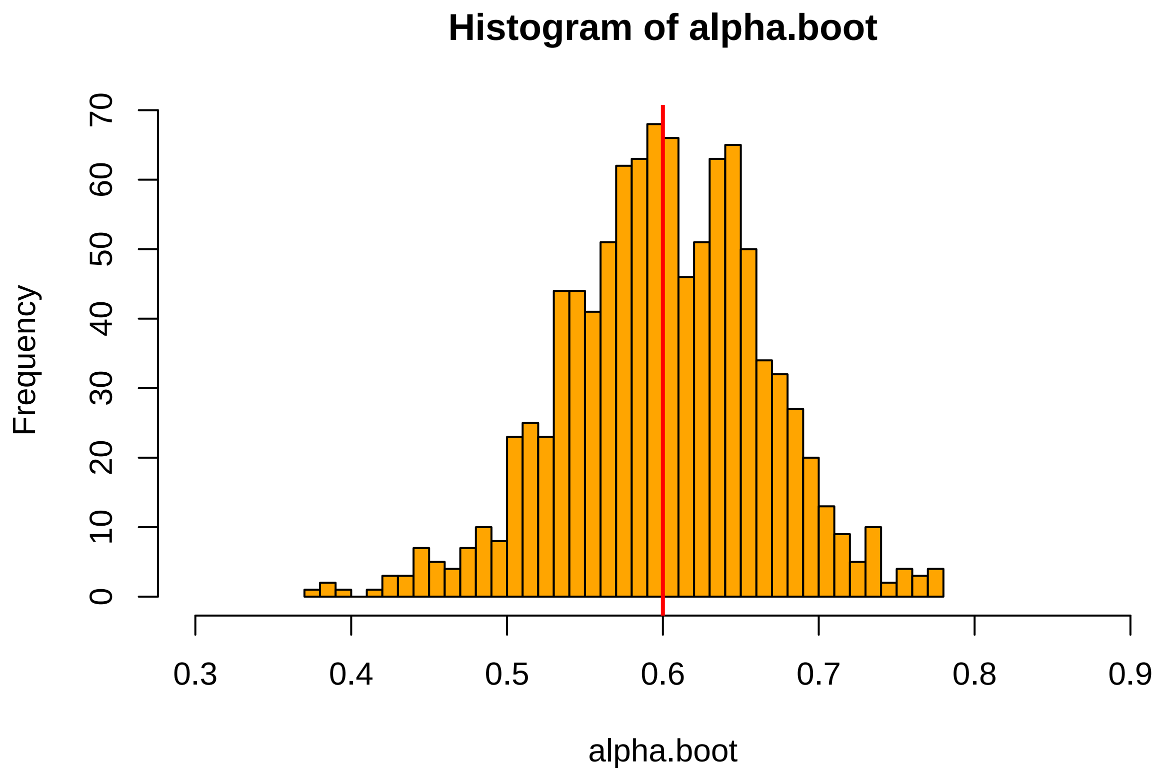
Let us now run bootstrap. Here is an estimate from one bootstrap sample.

```
alpha.fn(data.init, sample(100, 100, replace=T))
```

```
## [1] 0.664
```

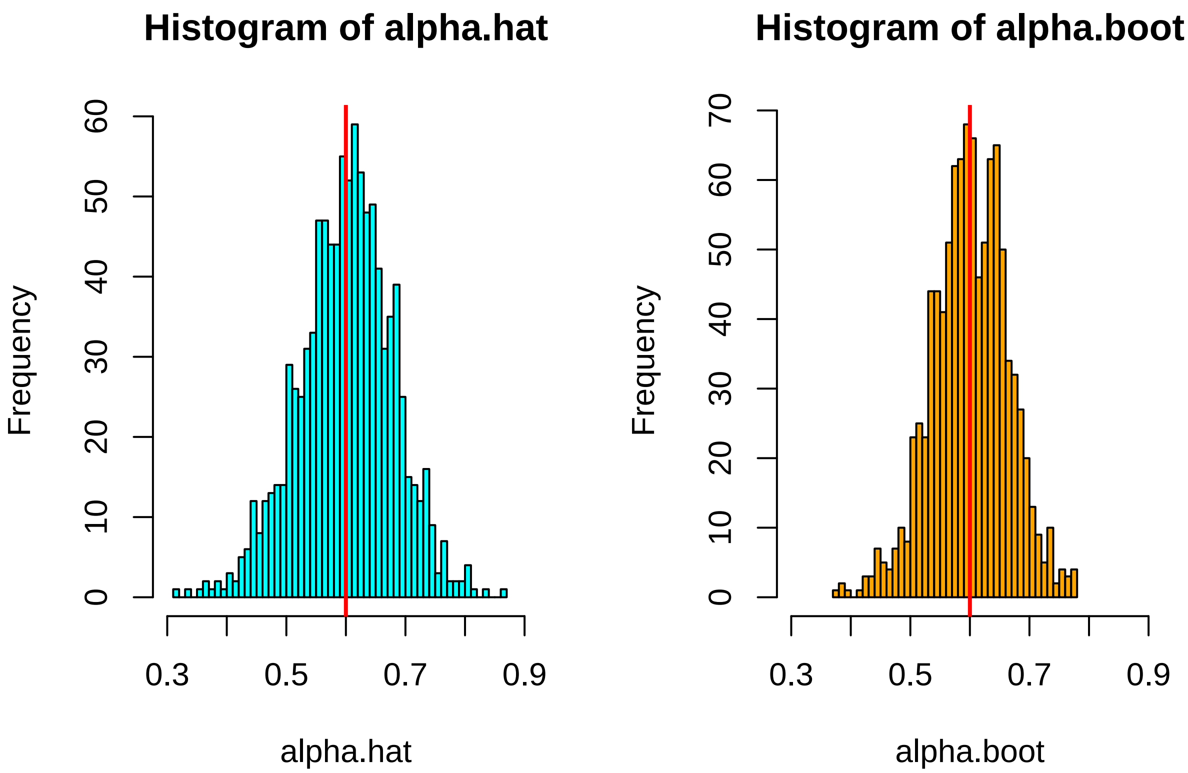
```
alpha.boot = double(1000)
for(i in 1:1000) {
  alpha.boot[i] = alpha.fn(data.init, sample(100, 100, replace=T))
}
hist(alpha.boot, breaks=50, col='orange', xlim = c(0.3, 0.9))
```

```
abline(v=0.6, col='red', lwd=2)
```

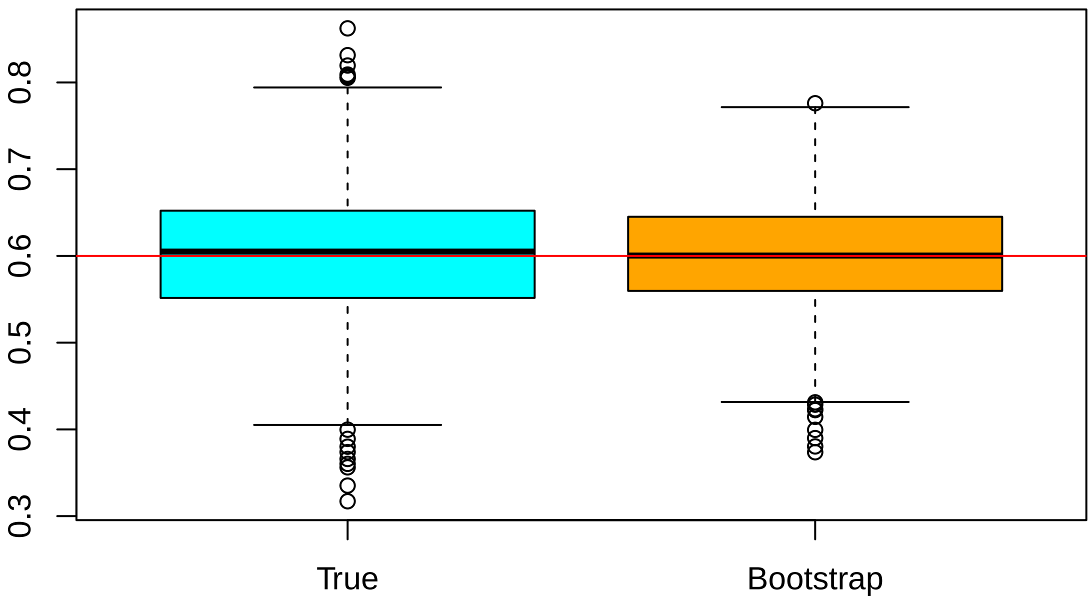


We can see that histograms are quite similar, illustrating that the bootstrap distribution well approximates the sampling distribution.

```
par(mfrow=c(1,2))
hist(alpha.hat, breaks=50, col='cyan', xlim = c(0.3, 0.9))
abline(v=0.6, col='red', lwd=2)
hist(alpha.boot, breaks=50, col='orange', xlim = c(0.3, 0.9))
abline(v=0.6, col='red', lwd=2)
```




```
boxplot(data.frame(True=alpha.hat, Bootstrap=alpha.boot), col=c('cyan','orange'))
abline(h=0.6, col='red')
```



Based on the bootstrap distribution, we can compute standard deviation of $\hat{\alpha}$

```
c( sd(alpha.hat), sd(alpha.boot) )
```

```
## [1] 0.0790 0.0638
```

or obtain confidence intervals.

CI based on the sampling distribution:

```
quantile(alpha.hat, probs = c(.025, .975))
```

```
##  2.5% 97.5%
## 0.442 0.747
```

Bootstrap CI:

```
quantile(alpha.boot, probs = c(.025, .975))
```

```
##  2.5% 97.5%
## 0.467 0.727
```

We do not need to manually write a for-loop to run the bootstrap.

```
library(boot)
out = boot(data.init, alpha.fn, R=1000)
print(out)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
```

```
##
## Call:
## boot(data = data.init, statistic = alpha.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*    0.601 0.000864      0.0632
```

```
boot.ci(out)
```

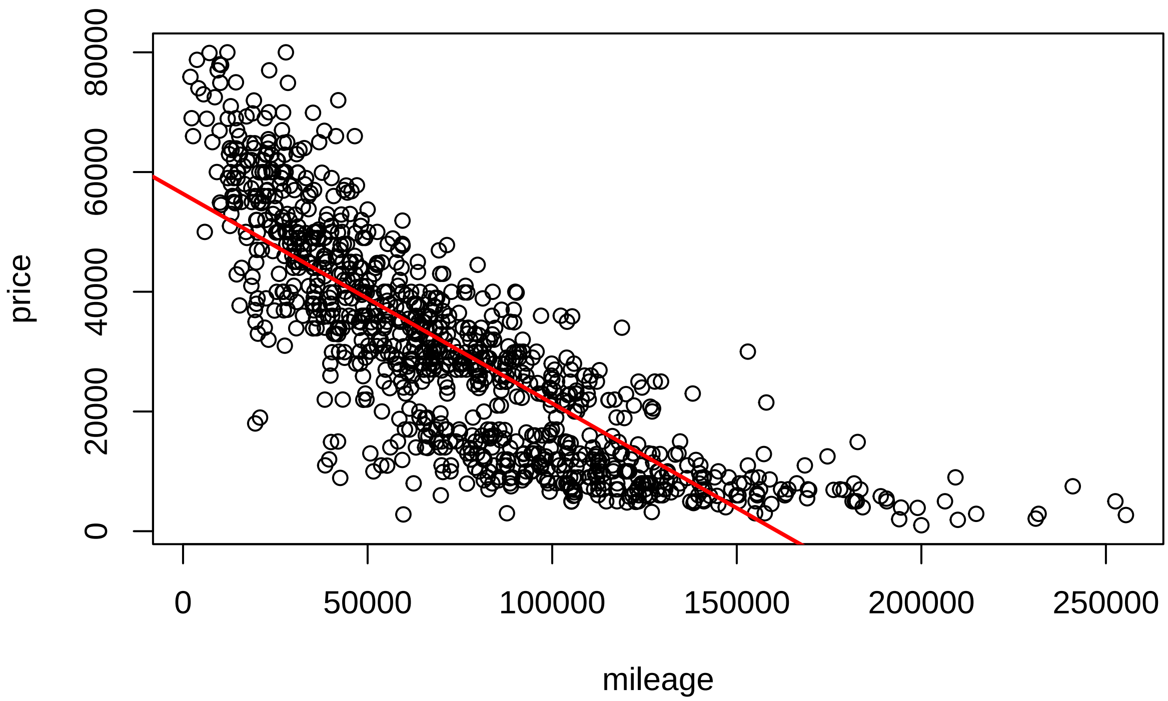
```
## Warning in boot.ci(out): bootstrap variances needed for studentized intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = out)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.476,  0.724 )   ( 0.481,  0.730 )
##
## Level     Percentile            BCa
## 95%   ( 0.471,  0.720 )   ( 0.471,  0.720 )
## Calculations and Intervals on Original Scale
```

Linear Regression

```
data = read.csv('UsedCars_small.csv')

plot(data$mileage, data$price, xlab='mileage', ylab='price')
lm.fit = lm(price~mileage, data)
abline(lm.fit, col='red', lwd=2)
```




```
summary(lm.fit)$coef
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 56359.78   6.71e+02    84.0  0.00e+00
## mileage        -0.35   7.87e-03   -44.5 5.37e-239
```

Bootstrap estimate

```
usedcars.coef.fn = function(data, index) {
  return( coef(lm(price~mileage, data=data, subset=index)) )
}

boot(data, usedcars.coef.fn, R=1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = usedcars.coef.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1* 56359.78 32.26013    891.3450
## t2*    -0.35 -0.00071      0.0111
```