The Bootstrap

John Clements

2/26/2022

Requirements

- `parallel`: for parallel computation
- `tidyquant`: for access to financial data
- `ggplot2`: for data visualization

```
suppressMessages(library(parallel))
suppressMessages(library(tidyquant))
suppressMessages(library(ggplot2))
```

I. Introduction

II. Technical Implementation

A. Single Bootstrap Function

```
one_bootstrap <- function(my_vec, my_func){
  ###
  # This function retrieves one bootstrapped sample and returns the statistic
  # of interest.
  #
  # Args
  # ----
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  #
  # Returns
  # -------
  # double
  #   The statistic of interest computed on the bootstrapped sample.
  #
  ###
  bootstrapped_sample <- sample(my_vec, size=length(my_vec), replace=TRUE)
  return(my_func(bootstrapped_sample))
}
```

B. Bootstrap Loop

```
bootstrap_loop <- function(my_vec, my_func, B){
  ###
  # This function takes in a data vector, function, and the number of bootstrap
  # iterations and returns a list holding the mean and standard deviation of the
  # bootstrap estimates as well, as the vector of the bootstrap estimates. It
  # utilizes a for loop.
  #
  # Args
  # ----
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  # B : int
  #   The number of bootstrapped samples to return.
  #
  # Returns
  # -------
  # output : list
  #   A list of the mean, and standard deviation of the estimates and a vector
  #   of the estimates.
  #
  ###
  estimates <- rep(NA, B)
  for (i in 1:B){
    estimates[i] <- one_bootstrap(my_vec, my_func)
  }
  output <- list(
    'mean' = mean(estimates),
    'se' = sd(estimates),
    'estimates' = estimates
  )
  return(output)
}
```

C. Bootstrap Replicate

```
bootstrap_replicate <- function(my_vec, my_func, B){
  ###
  # This function takes in a data vector, function, and the number of bootstrap
  # iterations and returns a list holding the mean and standard deviation of the
  # bootstrap estimates as well, as the vector of the bootstrap estimates. It
  # utilizes the replicate function for optimized looping.
  #
  # Args
  # ----
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
```

```
  #   Function which computes the statistic of interest.
  # B : int
  #   The number of bootstrapped samples to return.
  #
  # Returns
  # -------
  # output : list
  #   A list of the mean, and standard deviation of the estimates and a vector
  #   of the estimates.
  #
  ###
  estimates <- replicate(B, one_bootstrap(my_vec, my_func))
  output <- list(
    'mean' = mean(estimates),
    'se' = sd(estimates),
    'estimates' = estimates
  )
  return(output)
}
```

D. Parallelized Bootstrapping

```
bootstrap_replicate_par <- function(B, my_vec, my_func){
  ###
  # This function is a helper function for the parallized bootstrapping function.
  # It takes in a vector whose length determines the number of bootstrap samples
  # to take, a data vector, and a function. It returns the list of bootstrapped
  # estimates from my_func.
  #
  # Args
  # ----
  # B : vecot
  #   A vector whose length determines of bootstrapped samples to return.
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  #
  # Returns
  # -------
  # estimates : vector
  #   A vector of the estimates.
  #
  ###
  estimates <- replicate(length(B), one_bootstrap(my_vec, my_func))
  return(estimates)
}
```

```
bootstrap_parallel <- function(my_vec, my_func, B){
  ###
  # This function takes in a data vector, function, and the number of bootstrap
```

```
# iterations and returns a list holding the mean and standard deviation of the
# bootstrap estimates as well, as the vector of the bootstrap estimates. It
# utilizes parallel computing.
#
# Args
# ----
# my_vec : numeric vector
#   A vector of numbers of which to compute the statistic of interest.
# my_func : function
#   Function which computes the statistic of interest.
# B : int
#   The number of bootstrapped samples to return.
#
# Returns
# -------
# output : list
#   A list of the mean, and standard deviation of the estimates and a vector
#   of the estimates.
#
###

# Count the cores and make a cluster from leaving one core free.
cores <- detectCores()
cluster <- makeCluster(cores - 1)

# Create a vector that will be split up and determine the number of bootstrap
# samples to get on each core.
boot_vec <- 1:B

# Export functions to the cluster.
clusterExport(
  cluster,
  list("boot_vec", "one_bootstrap", "bootstrap_replicate_par", "my_vec",
       "my_func"),
  envir=environment()
  )

estimates <- parSapply(
  cluster,
  boot_vec,
  FUN=bootstrap_replicate_par,
  my_vec=my_vec,
  my_func=my_func
  )

stopCluster(cluster)

output <- list(
  'mean' = mean(estimates),
  'se' = sd(estimates),
  'estimates' = estimates
)
```

```
  return(output)
}
```

Applications

A. Data Set-up

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.

## [1] "^GSPC"
```

B. Sharpe Ratio

```
sharpe_ratio <- function(returns){
  return(mean(returns)/sd(returns))
}
```

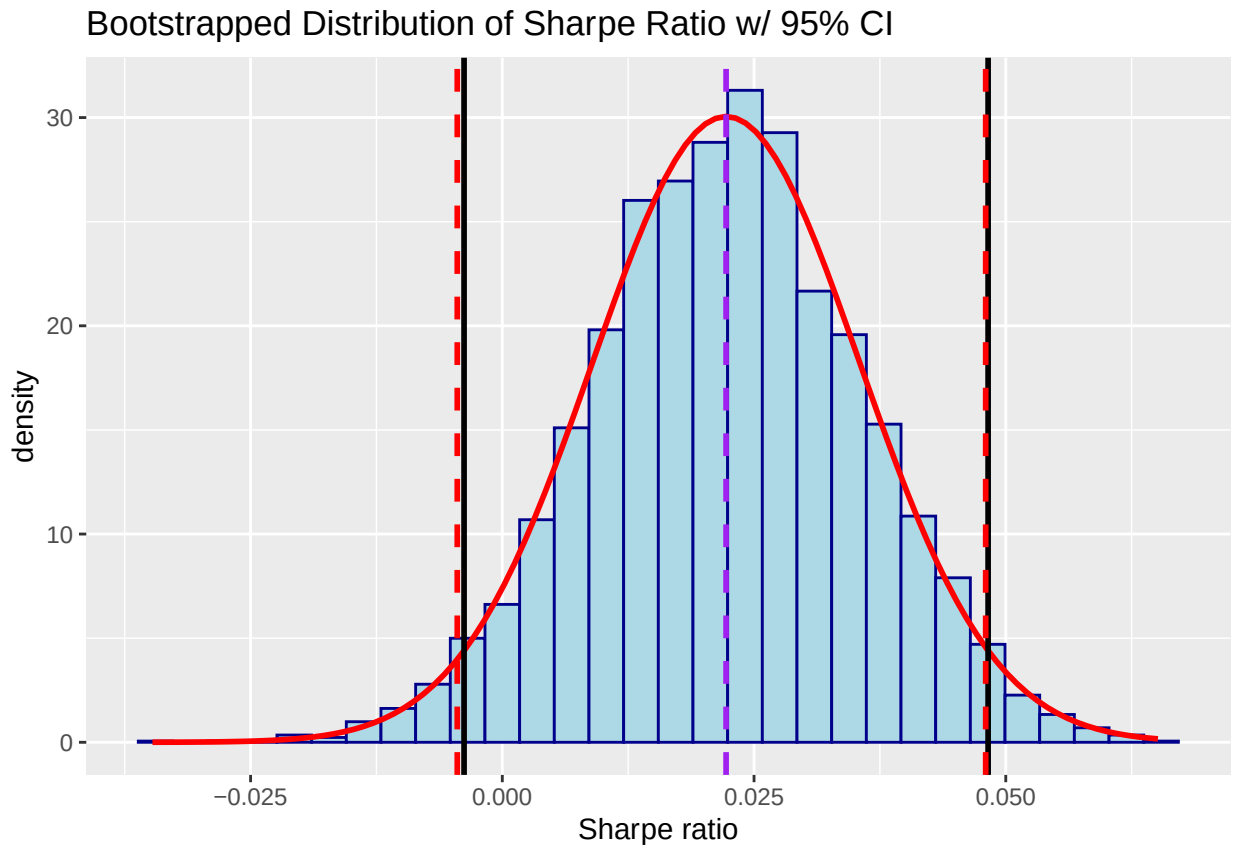
```
sharpe_est <- sharpe_ratio(returns)
sharpe_est
```

```
## [1] 0.02219397
```

```
set.seed(1)
B <- 5000

ptm_loop <- proc.time()
sharpe_est_loop <- bootstrap_loop(returns, sharpe_ratio, B)
loop_time <- proc.time() - ptm_loop

ptm_repl <- proc.time()
sharpe_est_repl <- bootstrap_replicate(returns, sharpe_ratio, B)
repl_time <- proc.time() - ptm_repl

ptm_par <- proc.time()
sharpe_est_par <- bootstrap_parallel(returns, sharpe_ratio, B)
par_time <- proc.time() - ptm_par
```

```
bootstrap_sharpe_df <- as.data.frame(list('sharpe'=sharpe_est_par$estimates))

df <- trading_days - 2
```

```r
# Define a Student t distribution with shape (nu) and location (mu)
# source: https://stackoverflow.com/questions/46848998/superimposing-asymmetric-t-distribution-using-gg
dt2 <- function(x, mu, nu, df, ncp) {
  dt((x-mu)/nu, df, ncp) / nu
}

ggplot(bootstrap_sharpe_df, aes(x=sharpe)) +
  geom_histogram(
    aes(y=(..density..)),
    color='darkblue',
    fill='lightblue',
    position = 'identity',
    bins=30) +
  stat_function(
    fun=dt2,
    args=list(mu=sharpe_est_par$mean, nu=sharpe_est_par$se, df=df),
    color='red',
    lwd=1
    ) +
  labs(
    title='Bootstrapped Distribution of Sharpe Ratio w/ 95% CI',
    x='Sharpe ratio',
    y='density'
    ) +
  geom_vline(xintercept=sharpe_est_par$mean, linetype='dashed',
             color='purple',
             size=1) +
  geom_vline(xintercept=(sharpe_est_par$mean + sharpe_est_par$se
                         * qt(0.025, df=df)),
             linetype='solid',
             color='black',
             size=1) +
  geom_vline(xintercept=(sharpe_est_par$mean + sharpe_est_par$se
                         * qt(0.975, df=df)),
             linetype='solid',
             color='black',
             size=1) +
  geom_vline(xintercept=quantile(sharpe_est_par$estimates, 0.025),
             linetype='dashed',
             color='red',
             size=1) +
  geom_vline(xintercept=quantile(sharpe_est_par$estimates, 0.975),
             linetype='dashed',
             color='red',
             size=1)
```



B. Value at Risk and Expected Shortfall

```
non_parametric_var <- function(returns, prob=0.05, holdings=1000000){
  return(as.numeric(-holdings*quantile(returns, probs=prob)))
}

non_parametric_es <- function(returns, prob=0.05, holdings=1000000){
  below_quantile <- returns < quantile(returns, probs=prob)
  numer <- sum(returns[below_quantile])
  denom <- sum(below_quantile)
  return(holdings * (numer / denom))
}

var_est <- bootstrap_parallel(returns, non_parametric_var, 100000)
es_est <- bootstrap_parallel(returns, non_parametric_es, 100000)
```