Statistics 506- Problem Set #6

Garrett Pinkston

Link to GitHub

github: https://github.com/garrettpinkston2015/Computational-Methods

Stratified Bootstrapping

If a sample has a categorical variable with small groups, bootstrapping can be tricky. Consider a situation where $n = 100$, but there is some categorical variable g where $g = 1$ has only 2 observations.

In a single bootstrap resample of that data, there is a

$$\binom{98}{100} \approx 13\%$$

chance that the bootstrap sample does not include either observation from $g = 1$. This implies that if we are attempting to obtain a bootstrap estimate in group $g = 1$, 13% of the bootstrapped samples will have no observations from that group and thus will be unable to produce an estimate.

A way around this is to carry out stratified bootstrap: Instead of taking a sample with replacement of the whole sample, take separate bootstrap resamples within each strata, then combine those resamples to generate the bootstrap sample.

Use the "lahman" data that we first introduced in SQL. In the statistical analysis of baseball statistics, one metric used to measure a player's performance is their **Range Factor**:

$$RF = 3\frac{PO + A}{InnOuts}$$

Here, "PO" is putouts, "A" is assists, and "InnOuts" is the number of outs they were on the field for.

a) Calculate the average RF for each team in the `Fielding` table. Then, since we don't have a closed form for the standard deviation of this statistic, carry out a stratified bootstrap *by team* to estimate it. Do this out three ways:

1. Without any parallel processing

```r
library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(DBI)
library(RSQLite)

db_path <- "/Users/garrettpinkston/Desktop/Michigan/STAT506/Data/lahman_1871-2022.sqlite"

# connect to database, read fielding table
lahman <- dbConnect(RSQLite::SQLite(), dbname = db_path)
fielding <- dbReadTable(lahman, "Fielding")

# filter and clean and calculate RF
fielding <- fielding %>%
  # remove missing or null rows
  filter(!is.na(PO), !is.na(A), !is.na(InnOuts), InnOuts > 0) %>%
  # add column for rf
  mutate(RF = 3 * (PO + A) / InnOuts)

# find the average RF for each team
team_avg_rf <- fielding %>%
  group_by(teamID) %>%                      # group data by teamID
  summarise(avg_RF = mean(RF, na.rm = TRUE)) # find the mean RF for each team

#' Perform Stratified Bootstrap
```

```r
#'
#' This performs a stratified bootstrap by resampling within groups
#'
#' @param data A data frame containing the data to be bootstrapped
#' @param strata_col The name of the column used for grouping.
#' @param n_boot The amount of bootstrap samples to generate.
#'
#' @return A data frame with bootstrap results with the bootstrap iteration and calculated s
#' @export
#'
#' @examples
#' # Example:
#' boot_results <- stratified_bootstrap(data = my_data, strata_col = "teamID", n_boot = 1000)
stratified_bootstrap <- function(data, strata_col, n_boot) {
  results <- vector("list", n_boot)         # initialize list to store results

  for (i in seq_len(n_boot)) {              # loop over number of bootstrap iterations
    # resample within each group and combine
    boot_sample <- data %>%
      group_by_at(strata_col) %>%           # group by column
      group_modify(~ .x[sample(nrow(.x), replace = TRUE), ]) %>% # resample with replacement
      ungroup()

    # find average RF for each team in the bootstrap sample
    boot_avg_rf <- boot_sample %>%
      group_by(teamID) %>%
      summarise(avg_RF = mean(RF, na.rm = TRUE))

    # store results
    results[[i]] <- boot_avg_rf
  }

  # combine all results together
  bind_rows(results, .id = "bootstrap_iteration")
}

# set number of bootstrap iterations and perform bootstap
n_boot <- 1000
boot_results <- stratified_bootstrap(fielding, "teamID", n_boot)

# find standard deviation of bootstrap means for each team
boot_sd <- boot_results %>%
  group_by(teamID) %>%                     # group by teamID
  summarise(sd_RF = sd(avg_RF, na.rm = TRUE)) # compute the standard deviation of RF

# combine all results into dataframe
final_results_normal <- team_avg_rf %>%
  left_join(boot_sd, by = "teamID")        # match by teamID to combine results

# print and disconnect
print(final_results_normal)
```

```
# A tibble: 140 x 3
   teamID avg_RF   sd_RF
   <chr>   <dbl>   <dbl>
 1 ALT     0.387 0.0503
 2 ANA     0.415 0.0150
 3 ARI     0.366 0.00846
 4 ATL     0.388 0.00596
 5 BAL     0.393 0.00533
 6 BFN     0.451 0.0194
 7 BFP     0.464 0.0532
 8 BL1     0.442 0.0287
 9 BL2     0.401 0.0155
10 BL3     0.444 0.0413
# i 130 more rows
```

```
dbDisconnect(lahman)
```

2. Using parallel processing with the parallel package.

```
library(DBI)
library(parallel)

# connect ot database
db_path <- "/Users/garrettpinkston/Desktop/Michigan/STAT506/Data/lahman_1871-2022.sqlite"
lahman <- dbConnect(SQLite(), dbname = db_path)

# read fielding and calculate range factor
fielding <- dbReadTable(lahman, "Fielding") %>%
  # filter out missing or invalid data
  filter(!is.na(PO), !is.na(A), !is.na(InnOuts), InnOuts > 0) %>%
  # calculate rf using provided formula
  mutate(RF = 3 * (PO + A) / InnOuts)

# find average rf for each team
team_avg_rf <- fielding %>%
  group_by(teamID) %>%                     # groups data by team ID
  summarise(avg_RF = mean(RF, na.rm = TRUE)) # find mean rf for each team

#' Perform single bootstrap iteration
#'
#' This function performs a single bootstrap by resampling within defined groups
#'
#' @param data A data frame containing data for the bootstrap operation
#' @param strata_col Name of column used for grouping.
#' @param n_boot The amount of bootstrap samples to generate.
#'
#' @return A data frame with bootstrap results with the bootstrap iteration and calculated st
#' @export
#'
#' @examples
#' # Example usage:
#' boot_results <- stratified_bootstrap(data = my_data, strata_col = "teamID", n_boot = 1000)
bootstrap_iter <- function(data) {
  library(dplyr)                             # reload dplyr (this fixed it)
  data %>%
    group_by(teamID) %>%                     # group data by team
    # resample data within each group
    group_modify(~ .x[sample(nrow(.x), replace = TRUE), ]) %>%
    # find average RF for each resampled group
    summarise(avg_RF = mean(RF, na.rm = TRUE))
}


#' Perform Parallel Bootstrap
#'
#' This function performs a parallel bootstrap.
#'
#' @param data A data frame which contains the data for bootstrap
#' @param n_boot The number of iterations to perform
#'
#' @return A data frame with bootstrap results with the bootstrap iteration and calculated st
#' @export
```

```r
#'
#' @examples
#' # Example usage:
#' boot_results <- parallel_bootstrap(data = my_data, n_boot = 1000)
parallel_bootstrap <- function(data, n_boot) {
  n_cores <- detectCores() - 1              # use most cores for parallel processing
  cl <- makeCluster(n_cores)                # create cluster

  # give data and bootsteap to all workewrs
  clusterExport(cl, c("data", "bootstrap_iter"), envir = environment())
  clusterEvalQ(cl, library(dplyr))

  # perform bootstrap iterations in parallel
  results <- parLapply(cl, 1:n_boot, function(i) bootstrap_iter(data))

  stopCluster(cl)                           # stop cluster when done

  bind_rows(results, .id = "bootstrap_iteration") # combine all results into one table
}

# define bootstrap number and run parallel process
n_boot <- 1000
boot_results <- parallel_bootstrap(fielding, n_boot)

# find sd of bootstrap results for each team
boot_sd <- boot_results %>%
  group_by(teamID) %>%                     # groub by team id
  summarise(sd_RF = sd(avg_RF, na.rm = TRUE)) # find sd of RD

# combine original average with new results
final_results_parallel <- team_avg_rf %>%
  left_join(boot_sd, by = "teamID")        # match rows using team ID

# print and disconnect
print(final_results_parallel)
```

```
# A tibble: 140 x 3
   teamID avg_RF   sd_RF
   <chr>   <dbl>   <dbl>
 1 ALT     0.387 0.0504
 2 ANA     0.415 0.0159
 3 ARI     0.366 0.00866
 4 ATL     0.388 0.00591
 5 BAL     0.393 0.00552
 6 BFN     0.451 0.0198
 7 BFP     0.464 0.0537
 8 BL1     0.442 0.0291
 9 BL2     0.401 0.0159
10 BL3     0.444 0.0412
# i 130 more rows
```

```
dbDisconnect(lahman)
```

3. Using futures with the future package.

```
library(dplyr)
library(DBI)
library(RSQLite)
library(future)
library(furrr)

# connect to database
db_path <- "/Users/garrettpinkston/Desktop/Michigan/STAT506/Data/lahman_1871-2022.sqlite"
lahman <- dbConnect(SQLite(), dbname = db_path)

# read fielding and calculate range factor
fielding <- dbReadTable(lahman, "Fielding") %>%
  # filter out missing or invalid data
  filter(!is.na(PO), !is.na(A), !is.na(InnOuts), InnOuts > 0) %>%
  # calculate rf using provided formula
  mutate(RF = 3 * (PO + A) / InnOuts)

# find average rf for each team
team_avg_rf <- fielding %>%
  group_by(teamID) %>%                    # groups data by team ID
  summarise(avg_RF = mean(RF, na.rm = TRUE)) # find mean rf for each team

#' Perform single bootstrap iteration
#'
#' This function performs a single bootstrap by resampling within defined groups
#'
#' @param data A data frame containing data for the bootstrap operation
#' @param strata_col Name of column used for grouping.
#' @param n_boot The amount of bootstrap samples to generate.
```

```r
#'
#' @return A data frame with bootstrap results with the bootstrap iteration and calculated s
#' @export
#'
#' @examples
#' # Example usage:
#' boot_results <- stratified_bootstrap(data = my_data, strata_col = "teamID", n_boot = 1000)
bootstrap_iter <- function(data) {
  data %>%
    group_by(teamID) %>%                        # groups data by team ID
    group_modify(~ .x[sample(nrow(.x), replace = TRUE), ]) %>% # resample data
    summarize(avg_RF = mean(RF, na.rm = TRUE)) # find mean rf for each group
}

# setup parallel processing and define number of bootstraps
plan(multisession)
n_boot <- 1000

# perform bootstap and combine results
boot_results <- future_map_dfr(
  1:n_boot,                                # iterate to n_boot
  ~ bootstrap_iter(fielding),              # apply bootstrap function
  .options = furrr_options(seed = TRUE) #set random seed
)

# find sd of the bootstrap estimates for each team
boot_sd <- boot_results %>%
  group_by(teamID) %>%                     # group by team ID
  summarise(sd_RF = sd(avg_RF, na.rm = TRUE)) # find sd of RF

# merge both datasets
final_result_future <- team_avg_rf %>%
  left_join(boot_sd, by = "teamID")        # joining on team id

# print and disconnect from database
print(final_result_future)
```

```
# A tibble: 140 x 3
   teamID avg_RF   sd_RF
   <chr>   <dbl>   <dbl>
 1 ALT     0.387 0.0503
 2 ANA     0.415 0.0154
 3 ARI     0.366 0.00845
 4 ATL     0.388 0.00590
 5 BAL     0.393 0.00535
 6 BFN     0.451 0.0211
 7 BFP     0.464 0.0515
 8 BL1     0.442 0.0307
 9 BL2     0.401 0.0154
10 BL3     0.444 0.0404
# i 130 more rows
```

```r
dbDisconnect(lahman)
```

b) Generate a table showing the estimated RF and associated standard errors for the teams with the 10 highest RF from the three approaches.

```r
# normalize variable names
results_non_parallel <- final_results_normal
results_parallel <- final_results_parallel
results_future <- final_result_future

# add a column for each method used for each set of results
results_non_parallel <- results_non_parallel %>% mutate(method = "Non-Parallel")
results_parallel <- results_parallel %>% mutate(method = "Parallel")
results_future <- results_future %>% mutate(method = "Future")

# combine all results
all_results <- bind_rows(results_non_parallel, results_parallel, results_future)

# get the top 10 teams with the highest RF for each method
top_teams <- all_results %>%
  arrange(desc(avg_RF)) %>%       # sort by rf descending
  group_by(method) %>%            # Group by method
  slice_head(n = 10) %>%          # Take the top 10 teams from each group
  ungroup()                       # Remove grouping for further operations

# table libraries
library(knitr)
library(kableExtra)
```

```
Attaching package: 'kableExtra'
```

```
The following object is masked from 'package:dplyr':

    group_rows
```

```
# format a table of the top 10 teams
top_teams_table <- top_teams %>%
  select(teamID, avg_RF, sd_RF, method) %>%  # select relevant columns
  arrange(method, desc(avg_RF)) %>%          # sort by method and then by RF
  kable(
    format = "latex",                        # specify format for pdf output
    booktabs = TRUE,                         # adds table borders
    col.names = c("Team ID", "Average RF", "Standard Error", "Method"), # column titles
    caption = "Top 10 Teams with Highest RF Across Methods"              # table name
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position")) # add fixed position

# print table
top_teams_table
```

c) Report and discuss the performance difference between the versions:

The future method generally produces slightly lower or comparable standard errors for most teams, which shows better consistency in bootstrap sampling across iterations since it handles random seeds and parallelization better. It is the fastest because it optimally manages parallel tasks.

On the other hand, the non-parallel method shows slightly higher standard errors in some cases compared to future, which can be due to computational limitations or inconsistencies in handling random sampling. It is moderately fast but can be slower than the future method due to the cluster setup.

Last, the parallel method sometimes shows the highest standard errors (e.g., MLU, KEO), which could be attributed to variations in random sampling or differences in how workers handle bootstrap iterations in a distributed environment. It is the slowest, as it processes all iterations sequentially.

The three methods analyzed produce pretty consistent results in terms of average RF and team rankings, but their efficiency and reliability differ. The future approach stands out as the best overall method for handling stratified bootstrapping for this problem. This future method stands out as the most efficient, providing both speed and consistency. The non-parallel method is impractical, while the parallel method offers some speed improvements but lacks the scalability of the future approach.

Table 1: Top 10 Teams with Highest RF Across Methods

Team ID	Average RF	Standard Error	Method
RC1	0.5740314	0.0795596	Future
LS1	0.5301629	0.0581553	Future
ELI	0.5265842	0.0650527	Future
MLU	0.5133325	0.1291375	Future
KEO	0.5121290	0.1099859	Future
RIC	0.5089137	0.0682373	Future
BLA	0.4948384	0.0304272	Future
LS3	0.4891679	0.0157930	Future
TRN	0.4808805	0.0296185	Future
PHU	0.4805772	0.0471295	Future
RC1	0.5740314	0.0808567	Non-Parallel
LS1	0.5301629	0.0548937	Non-Parallel
ELI	0.5265842	0.0630483	Non-Parallel
MLU	0.5133325	0.1266411	Non-Parallel
KEO	0.5121290	0.1137470	Non-Parallel
RIC	0.5089137	0.0642643	Non-Parallel
BLA	0.4948384	0.0311740	Non-Parallel
LS3	0.4891679	0.0151789	Non-Parallel
TRN	0.4808805	0.0291545	Non-Parallel
PHU	0.4805772	0.0461898	Non-Parallel
RC1	0.5740314	0.0817094	Parallel
LS1	0.5301629	0.0556586	Parallel
ELI	0.5265842	0.0654689	Parallel
MLU	0.5133325	0.1247652	Parallel
KEO	0.5121290	0.1139143	Parallel
RIC	0.5089137	0.0652689	Parallel
BLA	0.4948384	0.0324283	Parallel
LS3	0.4891679	0.0155516	Parallel
TRN	0.4808805	0.0289410	Parallel
PHU	0.4805772	0.0483743	Parallel