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.

```
library(DBI)
library(RSQLite)

# Specify the path to your SQLite database
db_path <- "/Users/garrettpinkston/Desktop/Michigan/STAT506/Data/lahman_1871-2022.sqlite"

# Connect to the database
lahman <- dbConnect(RSQLite::SQLite(), dbname = db_path)
```

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

```
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
```

```
# connect to database
db_path <- "/Users/garrettpinkston/Desktop/Michigan/STAT506/Data/lahman_1871-2022.sqlite"

lahman <- dbConnect(RSQLite::SQLite(), dbname = db_path)

fielding <- dbReadTable(lahman, "Fielding")

fielding <- fielding %>%
  filter(!is.na(PO), !is.na(A), !is.na(InnOuts), InnOuts > 0) %>%
  mutate(RF = 3 * (PO + A) / InnOuts)

team_avg_rf <- fielding %>%
  group_by(teamID) %>%
  summarise(avg_RF = mean(RF, na.rm = TRUE))
```

```r
stratified_bootstrap <- function(data, strata_col, n_boot) {
  results <- vector("list", n_boot)

  for (i in seq_len(n_boot)) {
    boot_sample <- data %>%
      group_by_at(strata_col) %>%
      group_modify(~ .x[sample(nrow(.x), replace = TRUE), ]) %>%
      ungroup()

    boot_avg_rf <- boot_sample %>%
      group_by(teamID) %>%
      summarise(avg_RF = mean(RF, na.rm = TRUE))

    results[[i]] <- boot_avg_rf
  }

  bind_rows(results, .id = "bootstrap_iteration")
}

n_boot <- 1000
boot_results <- stratified_bootstrap(fielding, "teamID", n_boot)

boot_sd <- boot_results %>%
  group_by(teamID) %>%
  summarise(sd_RF = sd(avg_RF, na.rm = TRUE))

final_results_normal <- team_avg_rf %>%
  left_join(boot_sd, by = "teamID")

print(final_results_normal)
```

```
# A tibble: 140 x 3
   teamID avg_RF   sd_RF
   <chr>   <dbl>   <dbl>
 1 ALT     0.387 0.0506
 2 ANA     0.415 0.0161
 3 ARI     0.366 0.00873
 4 ATL     0.388 0.00600
 5 BAL     0.393 0.00544
 6 BFN     0.451 0.0208
 7 BFP     0.464 0.0526
```

```
 8 BL1     0.442 0.0296
 9 BL2     0.401 0.0154
10 BL3     0.444 0.0389
# i 130 more rows
```

```r
dbDisconnect(lahman)
```

2. Using parallel processing with the parallel package.

```r
library(dplyr)
library(DBI)
library(RSQLite)
library(parallel)

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

fielding <- dbReadTable(lahman, "Fielding") %>%
  filter(!is.na(PO), !is.na(A), !is.na(InnOuts), InnOuts > 0) %>%
  mutate(RF = 3 * (PO + A) / InnOuts)

team_avg_rf <- fielding %>%
  group_by(teamID) %>%
  summarise(avg_RF = mean(RF, na.rm = TRUE))

bootstrap_iter <- function(data) {
  library(dplyr)
  data %>%
    group_by(teamID) %>%
    group_modify(~ .x[sample(nrow(.x), replace = TRUE), ]) %>%
    summarise(avg_RF = mean(RF, na.rm = TRUE))
}

parallel_bootstrap <- function(data, n_boot) {
  n_cores <- detectCores() - 1  # Use all but one core
  cl <- makeCluster(n_cores)

  clusterExport(cl, c("data", "bootstrap_iter"), envir = environment())

  clusterEvalQ(cl, library(dplyr))
```

```
  results <- parLapply(cl, 1:n_boot, function(i) bootstrap_iter(data))
  stopCluster(cl)

  bind_rows(results, .id = "bootstrap_iteration")
}

n_boot <- 1000
boot_results <- parallel_bootstrap(fielding, n_boot)

boot_sd <- boot_results %>%
  group_by(teamID) %>%
  summarise(sd_RF = sd(avg_RF, na.rm = TRUE))

final_results_parallel <- team_avg_rf %>%
  left_join(boot_sd, by = "teamID")

print(final_results_parallel)
```

```
# A tibble: 140 x 3
   teamID avg_RF   sd_RF
   <chr>   <dbl>   <dbl>
 1 ALT     0.387 0.0513
 2 ANA     0.415 0.0154
 3 ARI     0.366 0.00892
 4 ATL     0.388 0.00615
 5 BAL     0.393 0.00568
 6 BFN     0.451 0.0201
 7 BFP     0.464 0.0531
 8 BL1     0.442 0.0291
 9 BL2     0.401 0.0156
10 BL3     0.444 0.0404
# i 130 more rows
```

```
dbDisconnect(lahman)
```

3. Using futures with the future package.

```
library(dplyr)
library(DBI)
library(RSQLite)
```

```r
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)

fielding <- dbReadTable(lahman, "Fielding") %>%
  filter(!is.na(PO), !is.na(A), !is.na(InnOuts), InnOuts > 0) %>%
  mutate(RF = 3 * (PO + A) / InnOuts)

team_avg_rf <- fielding %>%
  group_by(teamID) %>%
  summarise(avg_RF = mean(RF, na.rm = TRUE))

bootstrap_iter <- function(data) {
  data %>%
    group_by(teamID) %>%
    group_modify(~ .x[sample(nrow(.x), replace = TRUE), ]) %>%
    summarise(avg_RF = mean(RF, na.rm = TRUE))
}

plan(multisession)

n_boot <- 1000
boot_results <- future_map_dfr(1:n_boot, ~ bootstrap_iter(fielding), .options = furrr_options

boot_sd <- boot_results %>%
  group_by(teamID) %>%
  summarise(sd_RF = sd(avg_RF, na.rm = TRUE))

final_result_future <- team_avg_rf %>%
  left_join(boot_sd, by = "teamID")

print(final_result_future)
```

```
# A tibble: 140 x 3
   teamID avg_RF   sd_RF
   <chr>   <dbl>   <dbl>
 1 ALT     0.387 0.0502
 2 ANA     0.415 0.0154
 3 ARI     0.366 0.00876
```

```
 4 ATL     0.388 0.00590
 5 BAL     0.393 0.00554
 6 BFN     0.451 0.0207
 7 BFP     0.464 0.0536
 8 BL1     0.442 0.0285
 9 BL2     0.401 0.0165
10 BL3     0.444 0.0406
# i 130 more rows
```

```
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.

```
#normalize names for consistency
results_non_parallel <- final_results_normal
results_parallel <- final_results_parallel
results_future <- final_result_future

results_non_parallel <- results_non_parallel %>% mutate(method = "Non-Parallel")
results_parallel <- results_parallel %>% mutate(method = "Parallel")
results_future <- results_future %>% mutate(method = "Future")
all_results <- bind_rows(results_non_parallel, results_parallel, results_future)

top_teams <- all_results %>%
  arrange(desc(avg_RF)) %>%
  group_by(method) %>%
  slice_head(n = 10) %>%
  ungroup()

library(knitr)
library(kableExtra)
```

```
Attaching package: 'kableExtra'

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

    group_rows
```

```r
top_teams_table <- top_teams %>%
  select(teamID, avg_RF, sd_RF, method) %>%
  arrange(method, desc(avg_RF)) %>%
  kable(
    format = "latex",
    booktabs = TRUE,
    col.names = c("Team ID", "Average RF", "Standard Error", "Method"),
    caption = "Top 10 Teams with Highest RF Across Methods"
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))

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. On the other hand, the non-parallel method shows slightly higher standard errors in some cases compared to future, potentially due to computational limitations or slight inconsistencies in handling random sampling over a long single-threaded process. 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.

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.

Table 1: Top 10 Teams with Highest RF Across Methods

Team ID	Average RF	Standard Error	Method
RC1	0.5740314	0.0831038	Future
LS1	0.5301629	0.0565511	Future
ELI	0.5265842	0.0650235	Future
MLU	0.5133325	0.1315513	Future
KEO	0.5121290	0.1150830	Future
RIC	0.5089137	0.0686714	Future
BLA	0.4948384	0.0321775	Future
LS3	0.4891679	0.0154814	Future
TRN	0.4808805	0.0290972	Future
PHU	0.4805772	0.0476081	Future
RC1	0.5740314	0.0832428	Non-Parallel
LS1	0.5301629	0.0558963	Non-Parallel
ELI	0.5265842	0.0655006	Non-Parallel
MLU	0.5133325	0.1293780	Non-Parallel
KEO	0.5121290	0.1137438	Non-Parallel
RIC	0.5089137	0.0683903	Non-Parallel
BLA	0.4948384	0.0309958	Non-Parallel
LS3	0.4891679	0.0156701	Non-Parallel
TRN	0.4808805	0.0310905	Non-Parallel
PHU	0.4805772	0.0475361	Non-Parallel
RC1	0.5740314	0.0817341	Parallel
LS1	0.5301629	0.0568970	Parallel
ELI	0.5265842	0.0615765	Parallel
MLU	0.5133325	0.1198037	Parallel
KEO	0.5121290	0.1144725	Parallel
RIC	0.5089137	0.0690510	Parallel
BLA	0.4948384	0.0317191	Parallel
LS3	0.4891679	0.0156552	Parallel
TRN	0.4808805	0.0299672	Parallel
PHU	0.4805772	0.0460330	Parallel