Stats_506_PS6

```r
suppressWarnings(library(nycflights13))
suppressWarnings(library(future))
```

```r
# Load our data
data <- flights[, c("origin", "dest", "air_time")]
```

```r
set.seed(8964)
```

First, we can use the following function to perform stratified bootstrap. This function is the only computationally costly function in this problem set.

```r
# Function to perform stratified bootstrap
stratified_bootstrap <- function(data) {
  unique_strata <- unique(data$dest)
  bootstrap_sample <- data.frame()

  for (stratum in unique_strata) {
    stratum_data <- subset(data, dest == stratum)
    n_stratum <- nrow(stratum_data)

    # Sample with replacement within each stratum
    resample_indices <- sample(1:n_stratum, n_stratum, replace = TRUE)
    stratum_bootstrap_sample <- stratum_data[resample_indices, ]

    # Combine resamples
    bootstrap_sample <- rbind(bootstrap_sample, stratum_bootstrap_sample)
  }

  # Aggregate here to save memory
  # bootstrap_sample is a 336776*3 data frame
```

```r
  # After aggregation, our return value is of size 3*1
  return(aggregate(air_time ~ origin, data = bootstrap_sample, FUN = mean))
}
```

After obtaining the means from bootstrap samples, we can store the means of different origins from different samples into a single data frame.

Notice that for a single bootstrap sample, we only record k mean values, where k is the number of different origins, which is always 3 in our case.

```r
# bootstrap_sample is a 3*1 vector
# bootstrap_samples is a length-1000 list of 3*1 vectors
# This function takes in bootstrap_samples as input
# And returns a 3*1000 data frame
get_summary_vectors <- function(bootstrapped_samples){
  summary_vectors <- bootstrapped_samples[[1]]
  for (sample in bootstrapped_samples[-1]){
    summary_vectors <- cbind(summary_vectors, sample$air_time)
  }
  origins <- summary_vectors[, 1]
  summary_vectors <- summary_vectors[, -1]
  rownames(summary_vectors) <- origins
  return(summary_vectors)
}
```

```r
# Input 'row' is a length-1000 vector
# Return value is a list of point estimate and confidence interval
calculate_stats <- function(row) {
  mean_value <- mean(row, na.rm = TRUE)
  lwb <- quantile(row, 0.025, na.rm = TRUE)
  upb <- quantile(row, 0.975, na.rm = TRUE)
  return(c(mean_value, lwb, upb))
}
```

```r
# Print out the resultt data frame
print_result <- function(summary_vectors){
  summary_stats <- t(apply(summary_vectors, 1, calculate_stats))
  summary_stats <- as.data.frame(summary_stats)
  colnames(summary_stats) <- c("Estimate",
                               "Lower Bound(2.5%)", "Upper Bound(97.5%)")
  return(summary_stats)
}
```

Functions `get_summary_vectors`, `calculate_stats`, `print_result`, are not time consuming. We can check it later.

Without parallel processing

```
system.time({
  num_bootstraps <- 1000
  bootstrapped_samples <- lapply(1:num_bootstraps,
                                 function(i){
                                   stratified_bootstrap(data)
                                   })
})
```

```
   user  system elapsed
 935.41  206.35 1376.45
```

The above is the time we used to perform the bootstrap without parallel processing.

And the following is the point estimate and the confidence interval we obtained.

```
print_result(get_summary_vectors(bootstrapped_samples))
```

```
    Estimate Lower Bound(2.5%) Upper Bound(97.5%)
EWR 153.2928          152.8799           153.7275
JFK 178.3546          177.9107           178.8186
LGA 117.8276          117.5965           118.0562
```

As mentioned, we can rerun `print_result` to check it is not computationally costly.

```
system.time({
  print_result(get_summary_vectors(bootstrapped_samples))
})
```

```
   user  system elapsed
   0.04    0.00    0.06
```

With parallel

```
system.time({
  plan(multisession)
  num_bootstraps <- 1000
  bootstrapped_samples_original <- lapply(1:num_bootstraps,
                                 function(i){
                                   future(stratified_bootstrap(data),
                                          seed = TRUE)
                                   })
  bootstrapped_samples <- lapply(bootstrapped_samples_original, value)
})
```

```
   user  system elapsed 
 188.95   34.91 1816.50 
```

The above is the time we used to perform the bootstrap with parallel processing using `future` package. And the obtained point estimates and confidence intervals are given below.

```
print_result(get_summary_vectors(bootstrapped_samples))
```

```
    Estimate Lower Bound(2.5%) Upper Bound(97.5%)
EWR  153.297          152.8747           153.7091
JFK  178.352          177.9037           178.8064
LGA  117.824          117.5868           118.0551
```

And we can also check `print_result` function is not time-consuming.

```
system.time({
  print_result(get_summary_vectors(bootstrapped_samples))
})
```

```
   user  system elapsed 
   0.02    0.00    0.06 
```