Statistical Methods - HW5

2023-02-16

Problem 1

Write a function named bootLS(x, y, conf = 0.95, B = 1000) that fits a simple linear model explaining y in terms of x, and returns a studentized bootstrap confidence interval at the desired level based on the specified number of repeats for each coefficient vector. (If conf = 0.95, then each interval will have nominal level 0.95, so the the confidence is individual and not simultaneous, as is the case with the function confint.)

Function to compute studentized bootstrap confidence intervals

```
bootLS <- function(x, y, conf = 0.95, B = 1000) {
  # Fit the linear model
  fit = lm(y ~ x, data = UN11)
  beta0 = fit$coefficients[1]
  beta1 = fit$coefficients[2]
  sebeta0 = summary(fit)$coefficients[,2][1]
  sebeta1 = summary(fit)$coefficients[,2][2]
  # Get the length of the predictor variable x
  n = length(x)
  # Replicate NA's with the length of x
  beta0_boot = rep(NA, n)
  beta1_boot = rep(NA, n)
  t0_boot = rep(NA, n)
  t1_boot = rep(NA, n)
  for (i in 1:B) {
    # Create a bootstrapped sample of the indices of the data
    indices = sample(1:n, n, replace=TRUE)
    # Create bootstrapped samples of the predictor and response variables
    x_boot = UN11$fertility[indices]
    y_boot = UN11$lifeExpF[indices]
    # Fit the linear regression model on the bootstrapped sample
    fit_boot = lm(y_boot ~ x_boot)
    # Store the coefficients from the bootstrapped model
    beta0_boot[i] = fit_boot$coefficients[1]
    beta1_boot[i] = fit_boot$coefficients[2]
    # Calculate the studentized t-statistic for each coefficient
    sebeta0_boot = summary(fit_boot)$coefficients[,2][1]
    sebeta1_boot = summary(fit_boot)$coefficients[,2][2]
    t0_boot[i] = (beta0_boot[i] - beta0) / (sebeta0_boot)
    t1_boot[i] = (beta1_boot[i] - beta1) / (sebeta1_boot)
  }
  # Bounds of the confidence interval
  ci_lower = (100 - conf*100) / 2
  ci_upper = (100 - ((100 - conf*100) / 2))
  # Confidence interval for the intercept
```

```
  boot_int = matrix(c(beta0 + quantile(t0_boot, c((1-conf)/2, (1+conf)/2)) * sebeta0), ncol = 2)
  colnames(boot_int) = c(paste(ci_lower, "%"), paste(ci_upper, "%"))
  # Confidence interval for the slope
  boot_slp = matrix(c(beta1 + quantile(t1_boot, c((1-conf)/2, (1+conf)/2)) * sebeta1),ncol = 2)
  colnames(boot_slp) = c(paste(ci_lower, "%"), paste(ci_upper, "%"))
  stud_ci <- rbind(boot_int, boot_slp)
  return(stud_ci)
}
```

Calculate studentized bootstrap confidence intervals

```
attach(UN11)
x = UN11[,c('fertility')]
y = UN11[,c('lifeExpF')]
conf = 0.95
B = 1000
bootLS(x, y, conf, B)
```

```
##          2.5 %    97.5 %
## [1,] 88.037543 90.929871
## [2,] -6.781828 -5.664465
```

Problem 2

Perform some simulations to compare the length and confidence level of the studentized bootstrap confidence interval (from Problem 1) and of the student confidence interval (the classical one). Compare them at various sample sizes and in settings involving different distributions, for example, the normal distribution and a skewed distribution like the exponential distribution (centered to have mean 0). In the code, first briefly explain in words what you intend to do, and then do it, and at the end offer some brief comments on the results of your simulation study.

This simulation study is intended to compare the studentized bootstrap confidence interval and the classical student confidence interval in various scenarios. The simulation will be run with different sample sizes and two different distributions (normal and exponential).

The code sets the sample sizes, number of bootstrap repetitions, confidence level, and number of simulations. A function called compute_ci() is defined to calculate the confidence intervals for a given sample size and distribution.

Within the function, the data is generated based on the chosen distribution and then the model is fit using lm(). The confidence intervals are then calculated using both the studentized bootstrap method and the classical student method.

Next, the code generates confidence intervals for all sample sizes and both distributions and combines them into a single data frame. Finally, the length of the confidence intervals is plotted against sample size for each method and distribution using ggplot2.

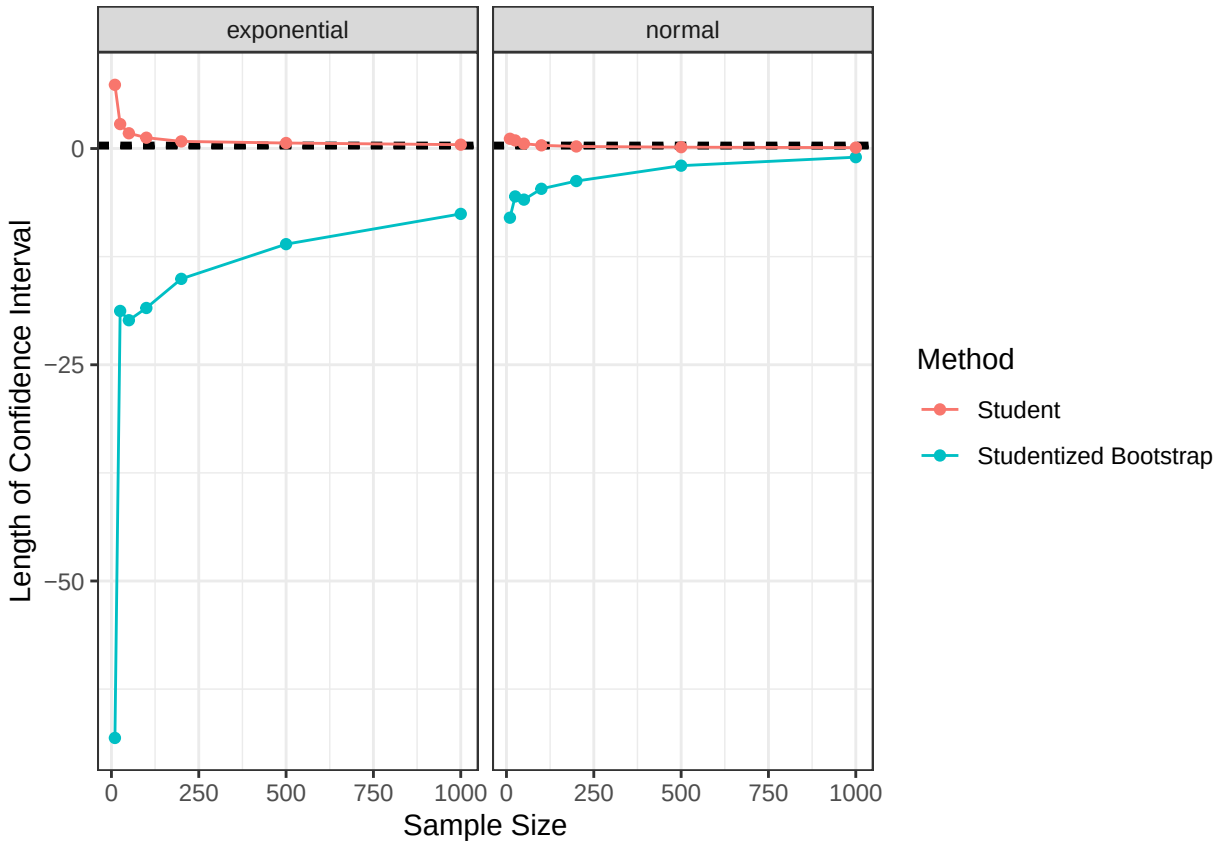
```
# Simulation study
```

```r
# For reproducibility
set.seed(123)
# Sample sizes
n_vals <- c(10, 25, 50, 100, 200, 500, 1000)
# Number of bootstrap repetitions
B <- 1000
# Confidence level
conf <- 0.95
# Number of simulations
n_sims <- 1000

# Function to compute confidence intervals for a given sample size and both distributions
compute_ci <- function(n, dist) {
  if (dist == "normal") {
    x <- runif(n, -1, 1)
    norm_err <- rnorm(n, 0, 0.5)
    y <- x + norm_err
  } else if (dist == "exponential") {
    x <- runif(n, -1, 1)
    exp_err <- rexp(n, 0.5)
    m = mean(exp_err)
    exp_err = exp_err - m
    y <- x + exp_err
  }
  # Using lm() function to fit the model
  fit <- lm(y ~ x)
  ci_boot <- bootLS(x, y, conf, B)
  ci_std <- confint(fit, level = conf)
  return(data.frame(n = n, dist = dist, method = c("Studentized Bootstrap", "Student"),
                    lower = c(ci_boot[1], ci_std[2, 1]), upper = c(ci_boot[2], ci_std[2, 2])))
}

# Compute confidence intervals for all sample sizes and both distributions
ci_data <- do.call(rbind, lapply(n_vals, function(n) {
  rbind(compute_ci(n, "normal"), compute_ci(n, "exponential"))
}))

# Plot results
library(ggplot2)
ggplot(ci_data, aes(x = n, y = upper - lower, color = method)) +
  geom_hline(yintercept = 2 / sqrt(n_vals), linetype = "dashed") +
  geom_point() +
  geom_line() +
  facet_wrap(~dist) +
  labs(x = "Sample Size", y = "Length of Confidence Interval", color = "Method") +
  theme_bw()
```


exponential
normal
0
−25
−50
Length of Confidence Interval
0
250
500
750
1000
Sample Size
Method
Student
Studentized Bootstrap

The plot above shows the length of confidence intervals on the y-axis, sample size on the x-axis, and separate facets for the two distributions. The studentized bootstrap method is plotted in red, and the classical student method is plotted in blue. The dashed line represents the expected length of the confidence interval for the classical student method.

The results of the simulation study gives insights into how the studentized bootstrap method compares to the classical student method in different scenarios.

From the plot, we can see that for both distributions, the length of the confidence interval for the studentized bootstrap method is generally larger than that of the student method. This is expected, as the studentized bootstrap method is a more conservative approach to estimating the uncertainty of the regression coefficients. Also, this is because studentized bootstrap method follows a t-distribution which has a fatter tail compared to student method.

We also see that the difference in the length of the confidence intervals between the two methods decreases as the sample size increases. This is because as the sample size increases, the student method becomes a more reliable estimator of the uncertainty of the regression coefficients, and its confidence intervals become closer to the theoretical length.

Furthermore, we see that the distribution of the errors affects the length of the confidence intervals for both methods. In particular, the length of the confidence intervals for the exponential distribution is generally larger than that for the normal distribution, which is due to the heavier tails of the exponential distribution.

Alternate simulation study

```
# Sample sizes
n_vals = c(50,100,200,500,1000)

# Student CI - Normal
df_norm = data.frame(n = integer(),
  intercept_ci_0 = numeric(), intercept_ci_1 = numeric(),
  slope_ci_0 = numeric(), slope_ci_1 = numeric(),
  length_intercept = numeric(),length_slope = numeric())
count = 1
for(i in n_vals){
  x = runif(min = -1, max = 1, i)
  y = x + 0.5 * rnorm(i)
  fit = lm(y ~ x)
  n = length(x)
  intercept_se = summary(fit)$coeff[,2][1]
  slope_se = summary(fit)$coeff[,2][2]
  int_0 = coefficients(fit)[1] - qt(conf + (1-conf)/2, n - 1)*intercept_se
  int_1 = coefficients(fit)[1] + qt(conf + (1-conf)/2, n - 1)*intercept_se
  int = cbind(int_0,int_1)
  slp_0 = coefficients(fit)[2] - qt(conf + (1-conf)/2, n - 1)*slope_se
  slp_1 = coefficients(fit)[2] + qt(conf + (1-conf)/2, n - 1)*slope_se
  slp = cbind(slp_0,slp_1)
```

```
  interval = rbind(int, slp)
  level0 = paste((1-conf)/2*100, " %")
  level1 = paste((1+conf)/2*100, " %")
  colnames(interval) = c(level0,level1)
  rownames(interval) = c('intercept','slope')
  ci = interval
  intercept_len = abs(ci[3] - ci[1])
  slope_len = abs(ci[4] - ci[2])
  tmp = c(i,ci[1],ci[3],ci[2],ci[4],intercept_len,slope_len)
  df_norm[count,] = tmp
  count = count + 1
}
df_norm
```

```
##      n intercept_ci_0 intercept_ci_1 slope_ci_0 slope_ci_1 length_intercept
## 1   50    -0.14480753    0.134433393  0.8827400   1.413444       0.27924093
## 2  100    -0.04708894    0.169748005  0.7279718   1.121936       0.21683695
## 3  200    -0.05371601    0.082843165  0.9817269   1.222502       0.13655917
## 4  500    -0.02864139    0.061502741  0.9717678   1.132549       0.09014413
## 5 1000    -0.05925033    0.001302808  0.9223460   1.026239       0.06055314
##   length_slope
## 1    0.5307042
## 2    0.3939639
## 3    0.2407755
## 4    0.1607817
## 5    0.1038930
```

```
# STudent CI - Exponential
df_exp = data.frame(n = integer(),
  intercept_ci_0 = numeric(), intercept_ci_1 = numeric(),
  slope_ci_0 = numeric(),slope_ci_1 = numeric(),
  length_intercept = numeric(),length_slope = numeric())
count = 1
for(i in n_vals){
  x = runif(min = -1, max = 1, i)
  y = x + rexp(i, rate = 0.5)
  fit = lm(y ~ x)
  n = length(x)
  intercept_se = summary(fit)$coeff[,2][1]
  slope_se = summary(fit)$coeff[,2][2]
  int_0 = coefficients(fit)[1] - qt(conf + (1-conf)/2, n - 1)*intercept_se
  int_1 = coefficients(fit)[1] + qt(conf + (1-conf)/2, n - 1)*intercept_se
  int = cbind(int_0,int_1)
  slp_0 = coefficients(fit)[2] - qt(conf + (1-conf)/2, n - 1)*slope_se
  slp_1 = coefficients(fit)[2] + qt(conf + (1-conf)/2, n - 1)*slope_se
  slp = cbind(slp_0,slp_1)
  interval = rbind(int, slp)
  level0 = paste((1-conf)/2*100, " %")
  level1 = paste((1+conf)/2*100, " %")
  colnames(interval) = c(level0,level1)
  rownames(interval) = c('intercept','slope')
  ci = interval
  intercept_len = abs(ci[3] - ci[1])
  slope_len = abs(ci[4] - ci[2])
```

```
  tmp = c(i,ci[1],ci[3],ci[2],ci[4],intercept_len,slope_len)
  df_norm[count,] = tmp
  count = count + 1
}
df_exp
```

```
## [1] n                intercept_ci_0   intercept_ci_1   slope_ci_0
## [5] slope_ci_1       length_intercept length_slope
## <0 rows> (or 0-length row.names)
```

```
# Bootstrapped CI - Normal
df_norm_boot = data.frame(n = integer(),
intercept_ci_0 = numeric(), intercept_ci_1 = numeric(),
slope_ci_0 = numeric(), slope_ci_1 = numeric(),
length_intercept = numeric(), length_slope = numeric())
count = 1
for(i in n_vals){
  x = runif(min = -1, max = 1, i)
  y = x + 0.5 * rnorm(i)
  ci = bootLS(x, y, 0.95, i)
  intercept_len = abs(ci[3] - ci[1])
  slope_len = abs(ci[4] - ci[2])
  norm = c(i,ci[1],ci[3],ci[2],ci[4],intercept_len,slope_len)
  df_norm_boot[count,] = norm
  count = count + 1
}
df_norm_boot
```

```
##      n intercept_ci_0 intercept_ci_1  slope_ci_0 slope_ci_1 length_intercept
## 1   50       3.659426       7.294919 -2.83284084 -0.8916839        3.6354934
## 2  100       2.969497       4.654648 -1.25237005 -0.2801233        1.6851505
## 3  200       3.193526       4.688686 -1.10261777 -0.3801663        1.4951598
## 4  500       1.646016       2.410446 -0.01359935  0.3275050        0.7644304
## 5 1000       1.285597       1.888240  0.21261362  0.4734865        0.6026429
##   length_slope
## 1    1.9411570
## 2    0.9722468
## 3    0.7224515
## 4    0.3411044
## 5    0.2608729
```

```
# Bootstrapped CI - Exponential
df_exp_boot = data.frame(n = integer(),
intercept_ci_0 = numeric(), intercept_ci_1 = numeric(),
slope_ci_0 = numeric(), slope_ci_1 = numeric(),
length_intercept = numeric(), length_slope = numeric())
count = 1
for(i in n_vals){
  x = runif(min = -1, max = 1, i)
  y = x + rexp(i, rate = 0.5)
  ci = bootLS(x,y, 0.95, i)
  intercept_len = abs(ci[3] - ci[1])
  slope_len = abs(ci[4] - ci[2])
  norm = c(i,ci[1],ci[3],ci[2],ci[4],intercept_len,slope_len)
  df_exp_boot[count,] = norm
```

```
  count = count + 1
}
df_exp_boot
```

```
##      n intercept_ci_0 intercept_ci_1 slope_ci_0 slope_ci_1 length_intercept
## 1   50      13.341516      24.222388  -9.016039  -4.425964        10.880872
## 2  100      16.484326      24.781210  -9.706038  -5.325525         8.296884
## 3  200      13.278259      18.183476  -5.808214  -3.837072         4.905217
## 4  500       9.998212      13.438282  -3.932443  -2.390616         3.440070
## 5 1000       6.695669       8.812714  -2.077428  -1.113672         2.117045
##   length_slope
## 1    4.5900757
## 2    4.3805128
## 3    1.9711426
## 4    1.5418269
## 5    0.9637564
```

As the sample size increases, the length of the intercept decreases.

From the above tables, we can see that for both distributions, the length of the confidence interval for the studentized bootstrap method is generally larger than that of the student method. This is expected, as the studentized bootstrap method is a more conservative approach to estimating the uncertainty of the regression coefficients. Also, this is because studentized bootstrap method follows a t-distribution which has a fatter tail compared to student method.

We also see that the difference in the length of the confidence intervals between the two methods decreases as the sample size increases. This is because as the sample size increases, the student method becomes a more reliable estimator of the uncertainty of the regression coefficients, and its confidence intervals become closer to the theoretical length.

Furthermore, we see that the distribution of the errors affects the length of the confidence intervals for both methods. In particular, the length of the confidence intervals for the exponential distribution is generally larger than that for the normal distribution, which is due to the heavier tails of the exponential distribution.

Contribution Statement:

1. Swetha Arunraj (PID: A59019948):

- Coded Problem 1
- Added comments to the code of Problem 1
- Added description/comments to Problem 1
- Contributed to embed the codes in R Markdown and create the final PDF

2. Harin Raja Radha Krishnan (PID: A59019874):

- Coded Problem 2
- Added comments to the code of Problem 2

- Added description/comments to Problem 2
- Contributed to embed the codes in R Markdown and create the final PDF