ST 558 Homework 6

Eric Warren

Contents

1 Code to Create this Document **1**

2 Writing a Function for the T-Test **1**

3 Quick Monte Carlo Study **3**

4 Parallel Computing **4**

1 Code to Create this Document

```
rmarkdown::render("~/ST-558---Data-Science-in-R/Homeworks/Warren_ST 558 Homework 6.Rmd",
                  output_format = "pdf_document",
                  output_options = list(
                    toc = TRUE,
                    toc_depth = 2,
                    number_sections = TRUE,
                    df_print = "tibble"
                    )
                  )
```

2 Writing a Function for the T-Test

First, we are going to write a function to calculate the test statistic. Inputs will be a vector of numeric data and the mean value to compare against μ_0. The output should be the calculated t_{obs} value.

```
calculate_t_statistic <- function(x, compared_mean) {
    t_obs <- (mean(x) - compared_mean)/(sd(x)/sqrt(length(x)))
    return(t_obs)
}
```

Now we are going to write a function to determine whether you reject H_0 or fail to reject it. Inputs should be the test statistic value, the sample size, the significance level (α), and the direction of the alternative hypothesis (left, right, or two-sided). The output should be a `TRUE` or `FALSE` value depending on whether or not you reject the null hypothesis.

```r
hypothesis_test <- function(test_statistic, sample_size, alpha = 0.05,
    direction = "two sided") {
    # Get the critical value (value we test against)
    t_critical_value <- ifelse(direction == "two sided", qt(alpha/2,
        df = sample_size - 1, lower.tail = FALSE), ifelse(direction ==
        "left", qt(alpha, df = sample_size - 1, lower.tail = TRUE),
        ifelse(direction == "right", qt(alpha, df = sample_size -
            1, lower.tail = FALSE), warning("Error: Must input values of 'two sided', 'left', or 'right
    # The results of the tests that can be done
    result_of_test_two_sided <- ifelse((direction == "two sided") &
        (abs(test_statistic) > t_critical_value), TRUE, FALSE)  #two sided test

    result_of_test_left <- ifelse((direction == "left") & (test_statistic <
        t_critical_value), TRUE, FALSE)  # One sided lower test

    result_of_test_right <- ifelse((direction == "right") & (test_statistic >
        t_critical_value), TRUE, FALSE)  # One sided upper test

    # Now get result of the appropriate test
    result_used <- ifelse(direction == "two sided", result_of_test_two_sided,
        ifelse(direction == "left", result_of_test_left, ifelse(direction ==
            "right", result_of_test_right, warning("Error: Must input values of 'two sided', 'left', or

    # Return the appropriate value of the test
    return(result_used)
}
```

Now we are going to test how well our functions work. Use the built-in `iris` data set and run a few hypothesis tests. Using the data, determine if the true mean:

- sepal length differs from 5.5 (i.e. test $H_0 : \mu = 5.5 vs H_A : \mu \neq 5.5$ and report whether or not we reject H_0)
- sepal width is greater than 3.5 (i.e. test $H_0 : \mu = 3.5 vs H_A : \mu > 3.5$ and report whether or not we reject H_0)
- petal length is less than 4 (i.e. test $H_0 : \mu = 4.0 vs H_A : \mu < 4.0$ and report whether or not we reject H_0)

First we are going to show the test of sepal length differs from 5.5 (i.e. test $H_0 : \mu = 5.5 vs H_A : \mu \neq 5.5$ and report whether or not we reject H_0).

```r
# Get the test statistic and store it for our hypothesis
# test
t_statistic1 <- calculate_t_statistic(iris$Sepal.Length, 5.5)

# Perform the two-sided hypothesis test
result_test1 <- hypothesis_test(t_statistic1, length(iris$Sepal.Length),
    alpha = 0.05, direction = "two sided")
result_test1
```

```
## [1] TRUE
```

Since the result from our test is TRUE, we can say that we have statistically significant evidence to reject our H_0. Therefore, we can conclude that we have statistically significant evidence to accept the alternative hypothesis that the average sepal length is not equal to 5.5.

Now we are going to show the test of sepal width is greater than 3.5 (i.e. test $H_0 : \mu = 3.5 vs H_A : \mu > 3.5$ and report whether or not we reject H_0).

```
# Get the test statistic and store it for our hypothesis
# test
t_statistic2 <- calculate_t_statistic(iris$Sepal.Width, 3.5)

# Perform the two-sided hypothesis test
result_test2 <- hypothesis_test(t_statistic2, length(iris$Sepal.Width),
    alpha = 0.05, direction = "right")
result_test2
```

```
## [1] FALSE
```

Since the result from our test is FALSE, we can say that we do not have statistically significant evidence to reject our H_0. Therefore, we can conclude that we do not have statistically significant evidence to accept the alternative hypothesis that the average sepal width is greater than 3.5. Therefore, we fail to reject our null hypothesis saying that the average sepal width is 3.5, which means this is a possible value for the average sepal length.

Lastly, we are going to show the test of petal length is less than 4 (i.e. test $H_0 : \mu = 4.0 vs H_A : \mu < 4.0$ and report whether or not we reject H_0).

```
# Get the test statistic and store it for our hypothesis
# test
t_statistic3 <- calculate_t_statistic(iris$Petal.Length, 4)

# Perform the two-sided hypothesis test
result_test3 <- hypothesis_test(t_statistic3, length(iris$Petal.Length),
    alpha = 0.05, direction = "left")
result_test3
```

```
## [1] TRUE
```

Since the result from our test is TRUE, we can say that we have statistically significant evidence to reject our H_0. Therefore, we can conclude that we have statistically significant evidence to accept the alternative hypothesis that the average petal length is less than 4.0.

3 Quick Monte Carlo Study

A Monte Carlo Simulation is one where we generate (pseudo) random values using a random number generator and use those random values to judge properties of tests, intervals, algorithms, etc. We'll generate data from a `gamma` distribution using different *shape* parameters and *sample sizes*. We'll then apply the t-test functions from above and see how well the α level is controlled under the incorrect assumption about the distribution our data is generated from. (By controlled we mean how close the observed proportion of rejected null hypotheses - from data where the null hypothesis is true - is to the desired significance level of α.)

The following Pseudocode will be used:

-generate a random sample of size n from the gamma distribution with a given shape and rate parameters specified (see `rgamma()`) - find the test statistic using the sample data and the mean of the gamma distribution

for μ0 (shape*rate) - using the given alternative (left, right, or both) determine whether you reject or not - repeat many times - observe the proportion of rejected null hypothesis from your repetitions (these are all incorrectly rejected since the null hypothesis is true). This proportion is our Monte Carlo estimate of the α value under this data generating process.

Write code to do the above when sampling from a gamma distribution with *shape = 0.2* and *rate = 1* with a *sample size of n = 10* for a two-sided test. Use the `replicate()` function (a wrapper for the `sapply()` function) to do the data generation and determination of reject/fail to reject (with the number of replications being 10,000)! Then find the mean of the resulting TRUE/FALSE values as your estimated α level under these assumptions.

```
# Put in shape, rate, and n values in
n_sample <- 10
shape_sample <- 0.2
rate_sample <- 1

# Generate data; set seed
set.seed(9)
sample_data <- replicate(10000, rgamma(n = n_sample, shape = shape_sample,
    rate = rate_sample))

# Get mu_0 = shape * rate
mu_0_gamma <- shape_sample * rate_sample

# Do for loop to go through all 10000 simulations
list_of_ttests <- lapply(1:ncol(sample_data), function(i) {
    t_statistic <- calculate_t_statistic(sample_data[, i], mu_0_gamma)
    hypothesis_test(t_statistic, length(sample_data[, i]), alpha = 0.05,
        direction = "two sided")
})

# Proportion of rejections
mean(as.numeric(list_of_ttests))
```

```
## [1] 0.2277
```

Here we can see that the proportion of rejected hypothesis tests were 0.2277. That means of our random samples there we drawn 22.77% of them were rejected for a total of 2277 random samples being rejected.

4 Parallel Computing

Now, we may want to do the above Monte Carlo simulation for several settings of sample size, shape, and rate parameter. As it turns out, we really don't need to worry about the rate parameter, it just scales the distribution. We will want to define the following:

- create a vector of sample size values (10, 20, 30, 50, 100)
- create a vector of shape values (0.2, 0.5, 1, 2, 5, 10, 20)
- create a rate vector with the value 1

Then we want to be able to use `parLapply()` to execute the above Monte Carlo study for combinations of sample sizes and shape values (with rate always 1). To use `parLapply()`, we need to pass:

- our cluster set up via `makeCluster()`
  - We must also use `clusterExport()` function to pass a list of our functions that we created
  - If you use any packages, you'll need to use `clusterEvalQ()` to pass the package
- X a list we will parallelize our computations over (each list element would be a combination of sample size and shape parameter)
- fun a function that does the replication of gamma samples, finds the decisions for each sample, and calculates and returns the proportion
  - Create a function to do this that just uses the `replicate()` and `mean()` code above, any other arguments needed for our functions to run (for instance rate = 1)

We are going to create a list we want to parallelize over but you'll need to figure out a way to create it (use `expand.grid()` and/or `apply()`)

```
# Get data first and into correct form
n <- c(10, 20, 30, 50, 100)
shape <- c(0.2, 0.5, 1, 2, 5, 10, 20)

list_sample_data <- unname(lapply(split(expand.grid(n = n, shape = shape),
    seq_along(expand.grid(n = n, shape = shape)[, 1])), as.list))

# Use parallel computing by loading up cores
library(parallel)
cluster <- makeCluster(detectCores() - 1)

clusterExport(cluster, list("calculate_t_statistic", "hypothesis_test",
    "list_sample_data"))
clusterEvalQ(cluster, library(tidyverse))
```

```
## [[1]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[2]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[3]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[4]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[5]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
```

```
##
## [[6]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[7]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[8]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[9]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[10]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[11]]
##  [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
##  [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
## [13] "grDevices" "utils"     "datasets"  "methods"   "base"
```

```
# Get results
results_grid <- parLapply(cluster, 1:length(list_sample_data),
    function(i) {
        set.seed(9)  # Set seed here for reproducible results
        sample_data <- replicate(10000, rgamma(n = list_sample_data[[i]]$n,
            shape = list_sample_data[[i]]$shape, rate = 1))  # Sample data
        mu_0_gamma <- list_sample_data[[i]]$shape * 1  # Get what the mu value is for a particular shap
        list_of_ttests <- lapply(1:ncol(sample_data), function(i) {
            t_statistic <- calculate_t_statistic(sample_data[,
                i], mu_0_gamma)
            hypothesis_test(t_statistic, length(sample_data[,
                i]), alpha = 0.05, direction = "two sided")
        })  # Get the list of test results
        mean(as.numeric(list_of_ttests))  # Get the proportion of rejections
    })
```

```
# close cluster
stopCluster(cluster)
```

Lastly, we are going to make a table of the proportion of rejections for each value of `n` and `shape` we tested.

```r
library(tidyverse)
# Turn lists to data frames
sample_data_combos <- expand.grid(n = n, shape = shape)
results_df <- as.data.frame(do.call(rbind, results_grid))
colnames(results_df) <- "results"

# Combine into one data frame
final_results <- bind_cols(sample_data_combos, results_df)

# Pivot data to wide format to make it easy to read
final_table <- final_results %>%
    pivot_wider(names_from = shape, names_glue = "shape = {shape}",
        values_from = results)

knitr::kable(final_table)
```

n	shape = 0.2	shape = 0.5	shape = 1	shape = 2	shape = 5	shape = 10	shape = 20
10	0.2277	0.1414	0.0990	0.0733	0.0623	0.0529	0.0514
20	0.1692	0.1041	0.0810	0.0690	0.0569	0.0541	0.0527
30	0.1348	0.0902	0.0762	0.0630	0.0542	0.0498	0.0518
50	0.1080	0.0725	0.0668	0.0630	0.0545	0.0563	0.0528
100	0.0796	0.0601	0.0596	0.0567	0.0553	0.0495	0.0510

Our final note here is that we can see that as the `shape` and `n` increases the proportion of rejected tests decreases. There are some anomalies that do not go with this trend but that is as the values for both tend to get really large (and they are really close proportions).