ST 558 Homework 7

John Hinic

2022-06-14

Contents

Improving R Programs

Writing a Custom t-test Function

We are going to write our own function to perform a t-test. The final function will require two helper functions:

1. The first one will take in a numeric vector and the mean under the null hypothesis, then calculate the t-statistic for us.
2. The second one will take in a t-statistic (from the first function), the sample size, the significance level α, and the form of the alternative hypothesis (left tailed, right tailed, or both).

t-statistic helper function

The first function we need will simply calculate the sample t-statistic for us. It will take in a numeric vector (i.e. our sample data), the scalar mean under the null hypothesis, and any additional arguments to pass to `sd()` or `mean()`. It will return a named list with 4 objects:

1. The sample t-statistic
2. The number of non-missing observations in the input data
3. The sample mean, rounded to 3 digits
4. The sample standard deviation, rounded to 4 digits

```
calcT <- function(data, null, ...){
  ybar <- mean(data, ...)
  s <- sd(data, ...)
  n <- sum(1 - is.na(data))
  return(
    list(
```

```
      t = sqrt(n) * (ybar - null) / s,
      n = n,
      mean = round(ybar, 3),
      std = round(s, 4)
    )
  )
}
```

Decision helper function

The decision function will take in the t-value that calculated from the previous function, the sample size, significance level, and direction of the test. From that, it will calculate the p-value and decide whether we should reject the null hypothesis. It will return a named list with 2 objects:

1. A boolean value stating whether we should reject H_0 or not (`TRUE` = reject)
2. The p-value of the test, rounded to 5 digits

```
reject <- function(t, n, alpha = 0.05, direction = "both") {
  pval <- switch(
    direction,
    both = 2 * pt(abs(t), df = n-1, lower.tail = FALSE),
    left = pt(t, df = n-1),
    right = pt(t, df = n-1, lower.tail = FALSE),
    stop('Invalid direction input. Please specify "left", "right", or "both"')
  )
  if(pval < alpha) {
    if(round(pval, 4) == 0) pval <- "<0.0001"
      else pval <- round(pval, 5)
    return(list(
      reject = TRUE,
      pval = pval,
      tail = direction,
      alpha = alpha
      )
    )
  }
    else return(list(
      reject = FALSE,
      pval = round(pval, 4)
      )
    )
}
```

t-test wrapper function

Finally, we will create a wrapper function that actually carries out the test. It will use `calcT()` to calculate the sample statistics, then use `reject()` to generate the test results. This function will take in the numeric vector of data, null mean, significance level α, and the direction of the test (as well as any additional arguments to pass in `mean()` or `sd()`). It will return a named list of 7 objects:

1. The test statistic t of the sample

2. The boolean value of whether we reject or fail to reject H_0
3. The p-value, rounded to 5 digits
4. The sample mean, rounded to 3 digits
5. The mean under the null hypothesis
6. The sample standard deviation, rounded to 4 digits
7. The significance level α

```
ttestManual <- function(data, null, alpha = 0.05, direction = "both", ...) {
  t <- calcT(data, null, ...)
  result <- reject(t$t, t$n, alpha, direction, ...)
  return(
    list(
      `Test statistic` = round(t$t, 3),
      `Reject` = result$reject,
      `P-value` = result$pval,
      `Sample mean` = t$mean,
      `Null mean` = null,
      `Sample standard deviation` = t$std,
      `Significance level` = alpha
    )
  )
}
```

Monte Carlo Study

Now that we have a function to perform a basic t-test, we want to evaluate how robust the t-test is to a violation of assumptions (namely, that the data comes from a normal distribution). The aspect of the t-test that we want to evaluate is the significance level. When assumptions are met, α represents the probability we make a type I error.

In order to do this, we are going to generate data from a gamma distribution with several different shape parameters and sample sizes. We will use the previously created t-test function to test each sample, then observe how well α (the theoretical type I error rate) is controlled.

This bit of code will take a random sample of size 10 from a gamma distribution with a shape parameter of 0.5 and rate of 1, perform a 2-sided t-test with the default significance level (0.05) and H_0 of the true mean, extract the boolean value of our rejection decision, and repeat that 10000 times. Once the 10000 tests are complete, we will take the mean of the values to see the proportion of type I errors we made.

```
set.seed(9001)
npSamps <- replicate(10000, ttestManual(rgamma(10, 0.5), 0.5)[["Reject"]])
mean(npSamps)
```

```
## [1] 0.1417
```

We can see that our type I error rate of 0.1417 is a good deal higher than the expected 0.05, but this was only one specific scenario we tested under. We want to see how the t-test performs with many different sample sizes and shape parameters.

Parallelization

To test many different scenarios, we will want to parallelize our code to speed up the computations. We will perform the tests with several different sample sizes and shape parameters:

- sample sizes: 10, 20, 30, 40, 50
- shape parameters: 0.5, 1, 2, 5, 10, 20

With 5 levels of sample size and 6 of the shape parameter, we will have 30 total different scenarios (all with a rate parameter of 1). First, we will create a list that contains all these scenarios, then create a function that handles the simulation within each scenario (essentially the same as the previous code chunk). Then we will need to specify our cluster and export the functions.

```
n <- seq(10, 50, by = 10)
shape <- c(0.5, 1, 2, 5, 10, 20)
rate <- 1

parms <- apply(expand.grid(list(n=n, shape=shape)), MARGIN = 1, FUN = as.list)
parRep <- function(parms) {
  set.seed(parms$n*10 - 1)
  samps <- replicate(
    10000,
    ttestManual(
      rgamma(parms$n, parms$shape, rate = 1),
      null = parms$shape
      )[["Reject"]]
    )
  prop <- mean(samps)
  return(list(prop = prop, n = parms$n, shape = parms$shape))
}

cluster <- makeCluster(detectCores() - 2)
clusterExport(cluster, list("calcT", "reject", "ttestManual", "parRep"))
```

With the cluster set-up complete, we now just need to pass everything to `parLapply` and view the results.

```
resultsList <- parLapply(cluster, X = parms, fun = "parRep")
resultsTall <- tibble(
  prop = map_dbl(resultsList, "prop"),
  n = map_dbl(resultsList, "n"),
  shape = map_dbl(resultsList, "shape")
)
pivot_wider(
  resultsTall,
  names_from = shape,
  names_prefix = "shape = ",
  values_from = prop
)
```

```
## # A tibble: 5 x 7
##       n `shape = 0.5` `shape = 1` `shape = 2` `shape = 5` `shape = 10`
##   <dbl>         <dbl>       <dbl>       <dbl>       <dbl>        <dbl>
## 1    10         0.142       0.102       0.0762      0.0597       0.0565
## 2    20         0.104       0.0781      0.0631      0.0594       0.0543
## 3    30         0.0875      0.0744      0.0597      0.055        0.0527
## 4    40         0.0833      0.0702      0.0643      0.0546       0.0492
## 5    50         0.0749      0.0683      0.058       0.0533       0.0562
## # ... with 1 more variable: `shape = 20` <dbl>
```

We can now see the table of our results; note that the values of each "cell" are the type I error rates of the test given each set of parameters. This is nice, but a bit overwhelming to look for patterns. We can also summarize the type I error rates by each level of n and shape parameter individually:

```
resultsTall %>%
  group_by(n) %>%
  summarise("Type I Error Rate by Sample Size" = mean(prop))
```

```
## # A tibble: 5 x 2
##       n `Type I Error Rate by Sample Size`
##   <dbl>                              <dbl>
## 1    10                             0.0820
## 2    20                             0.0689
## 3    30                             0.0630
## 4    40                             0.0621
## 5    50                             0.0606
```

```
resultsTall %>%
  group_by(shape) %>%
  summarise("Type I Error Rate by Shape Parameter" = mean(prop))
```

```
## # A tibble: 6 x 2
##   shape `Type I Error Rate by Shape Parameter`
##   <dbl>                                 <dbl>
## 1   0.5                                0.0984
## 2   1                                  0.0786
## 3   2                                  0.0643
## 4   5                                  0.0564
## 5  10                                  0.0538
## 6  20                                  0.0525
```

Now, we can clearly see that the type I error rate decreases as both n and the shape parameters increase (averaged across the other). As both n and our shape parameter get larger, we can see that the error rate gets closer and closer to the specified α of 0.5, which I think makes sense (especially given the central limit theorem).