Problem Set 1

Applied Stats II

Due: February 11, 2024

Instructions

- Please show your work! You may lose points by simply writing in the answer. If the problem requires you to execute commands in `R`, please include the code you used to get your answers. Please also include the `.R` file that contains your code. If you are not sure if work needs to be shown for a particular problem, please ask.
- Your homework should be submitted electronically on GitHub in `.pdf` form.
- This problem set is due before 23:59 on Sunday February 11, 2024. No late assignments will be accepted.

Question 1

The Kolmogorov-Smirnov test uses cumulative distribution statistics test the similarity of the empirical distribution of some observed data and a specified PDF, and serves as a goodness of fit test. The test statistic is created by:

$$D = \max_{i=1:n} \left\{ \frac{i}{n} - F_{(i)}, F_{(i)} - \frac{i-1}{n} \right\}$$

where F is the theoretical cumulative distribution of the distribution being tested and $F_{(i)}$ is the ith ordered value. Intuitively, the statistic takes the largest absolute difference between the two distribution functions across all x values. Large values indicate dissimilarity and the rejection of the hypothesis that the empirical distribution matches the queried theoretical distribution. The p-value is calculated from the Kolmogorov- Smirnoff CDF:

$$p(D \leq d) = \frac{\sqrt{2\pi}}{d} \sum_{k=1}^{\infty} e^{-(2k-1)^2\pi^2/(8d^2)}$$

which generally requires approximation methods (see Marsaglia, Tsang, and Wang 2003). This so-called non-parametric test (this label comes from the fact that the distribution of the test statistic does not depend on the distribution of the data being tested) performs

poorly in small samples, but works well in a simulation environment. Write an `R` function that implements this test where the reference distribution is normal. Using `R` generate 1,000 Cauchy random variables (`rcauchy(1000, location = 0, scale = 1)`) and perform the test (remember, use the same seed, something like `set.seed(123)`, whenever you're generating your own data).

As a hint, you can create the empirical distribution and theoretical CDF using this code:

```
  # create empirical distribution of observed data
  ECDF <- ecdf(data)
  empiricalCDF <- ECDF(data)
  # generate test statistic
  D <- max(abs(empiricalCDF - pnorm(data)))
```

Answer to Question 1:

First I set my seed, create my random variables and create my empirical distribution of observed data, following the code that is provided:

```
set.seed(123) # Setting my seed
data1 <- rcauchy(1000) # Creating my random variables

# Creating empirical distribution of observed data
ECDF <- ecdf(data1) # Creating my function
empiricalCDF <- ECDF(data1) # Using function to calculate cumulative probs for
    data
```

Then I can generate my test statistic with the code provided:

```
# Generating test statistic
D <- max(abs(empiricalCDF - pnorm(data1)))
print(D)
```

The resulting test statistic is **0.1347281**.
And I can create a function to calculate the corresponding p-value, following the provided formula:

```
# Creating function to calculate p-value:
# I follow the given formula.
# I use sum from k = 1 to 1000, instead of infinity.

kol_smirn_p <- function(D) {
  # Creating constants:
  sqrtpi = sqrt(2 * pi)

  # Initializing empty variables:
  sum_serie = 0

  # Summing from k = 1 to 1000, using a for loop:
```

```
  for (k in 1:1000) {
    term = exp(-(2*k - 1)^2 * pi^2 / (8 * D^2))
    sum_serie = sum_serie + term
  }

  p_val = (sqrtpi / D) * sum_serie
  return(p_val)
}
```

Finally, substituting our test statistic in our function:

```
# Substituting our D test statistic value:
pvalue <- kol_smirn_p(D)
print(pvalue)
```

This yields a p-value of **5.652523e-29**.
This is very close to the values produced by the ks.test() function, which yields **0.13573** for the test statistic and **2.22e-16** for the p-value.
Code to generate results from ks.test():

```
# Comparing to ks.test results:
ks_comp <- ks.test(data1, pnorm)
print(ks_comp)
```

We have used the Kolmogorov-Smirnov test to determine if a sample (the sample we created using rcauchy) comes from a population with a specific distribution (in this case, a normal distribution). From the very small p-value that we obtain (smaller than the standard 0.05, 0.01, or 0.001), we can reject the null hypothesis that the sample is drawn from a normal distribution, as it was expected.

Question 2

Estimate an OLS regression in `R` that uses the Newton-Raphson algorithm (specifically `BFGS`, which is a quasi-Newton method), and show that you get the equivalent results to using `lm`. Use the code below to create your data.

```
set.seed (123)
data <- data.frame(x = runif(200, 1, 10))
data$y <- 0 + 2.75*data$x + rnorm(200, 0, 1.5)
```

Answer to Question 2:

First I set my seed, create new data and a relationship, following the code provided:

```
set.seed (123) # Setting my seed
data2 <- data.frame(x = runif(200, 1, 10)) # Creating new data
data2$y <- 0 + 2.75*data2$x + rnorm(200, 0, 1.5) # Modeling relationship
```

Then I define my log-likelihood function for optimization. This is based on slides from class:

```
# Defining my log-likelihood function that I will maximize (or minimze its
    negative).
# My parameters for OLS are the intercept, the slope and the variance of the
    error term (sigma squared).
# So I take the formula of the log likelihood for a normal distribution.
# This is based on the slides for class:

linear.lik <- function(theta, y, X) {
  n <- nrow(X) # No. obs
  k <- ncol(X) # No. param
  beta <- theta[1:k] # Regression coefs
  sigma2 <- theta[k + 1]^2 # Squared variance

  e <- y - X %*% beta # Calculating residuals
  logl <- -0.5 * n * log(2 * pi) - 0.5 * n * log(sigma2) - ((t(e) %*% e) / (2
    * sigma2))
  return(-logl) # Returning negative log-likelihood for minimization
}
```

Then I can actually maximize my log-likelihood function (or minimize its negative, which is equivalent) with respect to my parameters. This code I also borrowed from slides:

```
# Maximizing the log-likelihood (or minimizing negative log likelihood) using
    BFGS method.
# This code is also taken from slides:
linear.MLE <- optim(fn = linear.lik, par = c(1, 1, 1), hessian = TRUE, y =
    data2$y, X = cbind(1, data2$x), method = "BFGS")
```

Looking at my results:

```
# Seeing results:
linear.MLE$par
```

This yields an estimated intercept of **0.1429829**, an estimated slope of **2.7263116**, and a n estimated sigma of **-1.4423360**.
And finally, comparing with lm:

```
# Comparing with lm:
lm_mod <- lm(y ~ x, data = data2)
summary(lm_mod)
```

Lm yields an estimated intercept of **0.13919**, and an estimated slope of **2.72670**, both of which are quite close to our maximum likelihood estimates.
The results of ordinary least squares and maximum likelihood estimation should be approximately the same, especially as the size of the data gets larger, however they will not be exactly the same, as MLE follows a different estimation method which involves optimization.