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:

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

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)))
```

I create a function which takes a dataset as an argument. It first calculates the test statistic using the code provided. Then it calculates the p-value using the previously calculated test statistic based on the formula above. Finally, the function returns the p-value and the t-stat. The p-value produced by the function is very close to the value produced by the built-in function, as the Kolmogorov-Smirnoff CDF requires approximation, as was mentioned in the instructions.

```
ks_tst <- function(data) {
  ECDF <- ecdf(data)
  empiricalCDF <- ECDF(data)
  # generate test statistic
  D <- max(abs(empiricalCDF - pnorm(data)))
  #####
  #
  n <- length(data)
  sum1 <- 0
  for (i in 1:n) {   # loop through each value
  # get the summation
  sum1 <- sum1 + exp(-(2*i-1)^2 * pi^2 / (8*D^2))
  }
  # solve the first part of the equation
  pval <- (sqrt(2*pi)/D) * sum1
  # Return the t-stat and p value
  answer <- list(D = D, P_Value = pval)
  return(answer)
}

# Test my function
cauchyData <- rcauchy(1000, location = 0, scale = 1)
ks_tst(cauchyData)
```

Here is what my function returns.

```
$D[1] 0.1439423
$P_Value[1] 2.407469e-25
```

This is the built in function and what it returns.

```
ks.test(cauchyData, "pnorm")
```

```
D = 0.14394, p-value < 2.2e-16
```

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.

```
  ECDF <- ecdf(data)
  empiricalCDF <- ECDF(data)
  # generate test statistic
```

First I'll run a regular old liner regression model, lets see what I get based on the data provided.

```
summary(mod1)
```

Coefficients estimates are (Intercept) 0.13919, X 0.25276

The R code below is the log-likelihood function. First, it calculates beta (predictor), sigma2 (variance of error), and e (residuals). I will use these values in the log-likelihood or log(L) equation, which is displayed in the lecture slides. We use the log-likelihood function because it is easier to compute. We can simply add together the log-likelihoods rather than multiplying each individual likleyhood.

```
lin_link <- function(theta, y, x){
  n <- nrow(x)
  k <- ncol(x)
  beta <- theta[1:k] # our predictors
  sigma2 <- theta[k+1]^2 # calculate the variance of error
  e <- y-x%*%beta # compute the residuals
  # log likelihood equation
  logl <- -.5*n*log(2*pi)-.5*n*log(sigma2) - ((t(e) %*%
      e)/ (2*sigma2))
  return(-logl)
}
```

Now, to use the log-likelihood function I created as one of the arguments in the optim() function to find the best-fitting line. The best-fitting line finds the parameter values which, for lack of a better term, maximize the likelihood that a point will fall at a given y value.

```r
linear_MLE <- optim(
  fn = lin_link,
  par = c(1, 1, 1), # initial parameters
  hessian = TRUE,
  y = data$y,
  x = cbind(1, data$x),
  method = "BFGS"
)

linear_MLE$par
```

The optim() function returns these coefficients Intercept: 0.1398324 X: 2.7265559 the lm() funciton returned the same values Intercept: 0.13919 X: 2.7265559. As can be seen, in this case the ML model returned the same coefficients as the lm model.

While it is possible to use maximum likelihood for linear regression, it is better to use ordinary least squares (OLS) because OLS is computationally simpler and unbiased regardless of sample size. However, maximum likelihood should be used when the OLS assumptions (non-linear, non-continuous variables) are not met.