`tureen625`: R Package Tutorial

Tahmeed Tureen

BSTAT 625 : Big Data Computing

Relevant Libraries

```r
library(tureen625)
```

Requirements for Submission:

- Comparison(s) against the original R functions on simulated or real datasets
- Correction (e.g. via all.equal())
- Efficiency (e.g. bench::mark())

Table of Contents

Fit the Linear Regression Model

For this tutorial, we will fit a multi-variable linear regression model with some continous predictor variables and one categorical predictor variable. The data we use for demonstration is the `mtcars` dataset which is provided by the `datasets` library (More documentation on this dataset can be viewed HERE).

The regression model we will fit is shown below:

$$mpg_i = \beta_0 + \beta_1 cyl_i + \beta_2 hp_i + \beta_3 wt_i + \beta_4 vs_i$$

where

- `mpg` : Miles/gallon
- `cyl` : Number of cylinders
- `hp` : Horsepower
- `wt` : Weight (1000 lbs)
- `vs` : Engine (0 = V-shape, 1 = manual)

Setting Up Data

```r
# Load up mtcars dataset
data("mtcars")
dim(mtcars)
```

```
## [1] 32 11
```

For our Ordinary Least Squares (OLS) function, we need the inputs of the outcome and independent variables to be separated and in matrix format. We demonstrate this process in the following `R` chunk:

```
# Create design matrix
# We need to add a vector of 1's for the intercept
X_matrix <- cbind(1, mtcars$cyl, mtcars$hp, mtcars$wt, mtcars$vs)

Y_vec <- as.vector(mtcars$mpg)
```

NOTE: You may want to fit a regression model that incorporates a categorical variable that has more than 2 classes. In that case, we recommend using the `model.matrix()` function to easily create a design matrix. This function will convert a matrix with factor variables into a dummy encoded (reference cell encoding) matrix (see `?model.matrix()`) for additional help.

For example:

```
# Make sure to use as.factor() if your variable isn't in that type/class format
X_matrix2 <- stats::model.matrix(~cyl + hp + wt + as.factor(vs), data = mtcars)
X_matrix2 <- as.matrix(X_matrix2)

# Check if all 160 entries are equal in the two design matrices
all.equal(sum(X_matrix ==  X_matrix2), 160)
```

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

Fitting the Model

We input our data variables into our function `fit_OLS()` to fit our OLS model

```
OLS.fit <- tureen625::fit_OLS(design_X = X_matrix, Y = Y_vec)

names(OLS.fit)
```

```
## [1] "param_estimates" "residuals"       "sigma_Sq"        "var_Mat"
## [5] "param_StdErrors" "conf_int"        "results"
```

Analyze & Visualize the Results

In this section, we show how we can use the returned object from our function to do some analyses of our results. First, we will observe our $\hat{\beta}$ estimates and the standard errors associated with the estimates $SE(\hat{\beta})$

```
OLS.fit$param_estimates
```

```
## [1] 38.48404581 -0.90501219 -0.01785427 -3.18355380  0.15457896
```

```
OLS.fit$param_StdErrors
```

```
## [1] 3.33740414 0.67894306 0.01224378 0.77368907 1.61535124
```

Note that the order of the values in this vector/matric objects corresponds to the same order in which the variables are arranged in the design matrix X. Thus, the first value from the two outputs above corresponds to the Intercept in the regression model and the 2nd value above corresponds to the `cyl` variable.

```
# View confidence intervals
OLS.fit$conf_int
```

```
##             [,1]         [,2]
## [1,] 31.94273370 45.025357921
```

```
## [2,] -2.23574059  0.425716221
## [3,] -0.04185208  0.006143545
## [4,] -4.69998438 -1.667123225
## [5,] -3.01150946  3.320667387
```

We can see that only the first and fourth $\hat{\beta}$ estimates have confidence interval intervals that do not include a zero. Therefore, we can make the claim that the weight (`wt`) of a car has a significant negative effect on the outcome of interest (`mpg`) by a magnitude of approximately -3.18 miles per gallon.

You can also view all of these results by simplying running the following code

```r
round(OLS.fit$results, 3)
```

```
##   Term Estimate Std. Error 95% CI Lower Bound 95% CI Upper Bound
## 1    1   38.484      3.337             31.943             45.025
## 2    2   -0.905      0.679             -2.236              0.426
## 3    3   -0.018      0.012             -0.042              0.006
## 4    4   -3.184      0.774             -4.700             -1.667
## 5    5    0.155      1.615             -3.012              3.321
```

Correction Analysis

We compare our results with the commonly used function in `R` for fitting linear regression models called `lm()`. First, we compare the parameter estimates then the standard errors. Finally, we compare if we can draw the same inferences from both functions.

Compare $\hat{\beta}$ estimates

```r
# Fit the model using lm()
lm.fit <- lm(formula = "mpg ~ cyl + hp + wt + vs", data = mtcars)
```

```r
all.equal(as.vector(lm.fit$coefficients), as.vector(OLS.fit$param_estimates))
```

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

- Our implemented `R` function has the same β esimates as the `lm()` function

Compare $SE(\hat{\beta})$

```r
stdErr.lm <- summary(lm.fit)$coefficients[,"Std. Error"]

all.equal(as.vector(stdErr.lm), OLS.fit$param_StdErrors)
```

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

- Our implemented `R` function has the same $SE(\hat{\beta})$ values as the `lm()` function

Compare residuals

```r
all.equal(as.vector(OLS.fit$residuals), as.vector(lm.fit$residuals))
```

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

- Our implemented `R` function has the same residual values as the `lm()` function

Efficiency Analysis

Now, we compare the speed of the two functions by using the `microbenchmark` package.

```
library(microbenchmark)
```

```
microbenchmark(
    lm(formula = "mpg ~ cyl + hp + wt + vs", data = mtcars),
    tureen625::fit_OLS(design_X = X_matrix, Y = Y_vec)
)
```

```
## Unit: microseconds
##                                                     expr   min     lq     mean
##  lm(formula = "mpg ~ cyl + hp + wt + vs", data = mtcars) 912.9 1430.4 1836.751
##       tureen625::fit_OLS(design_X = X_matrix, Y = Y_vec) 256.7  409.5  529.054
##  median      uq    max neval
##  1903.3 2179.75 5680.7   100
##   527.8  622.85 1525.3   100
```

- We see that our implemented function runs faster than the `lm()`, on average. We believe this is because our function provides estimates for the $\hat{\beta}$, the standard errors, residuals, and the confidence intervals. See below for a couple more features.

```
names(OLS.fit)
```

```
## [1] "param_estimates" "residuals"       "sigma_Sq"        "var_Mat"
## [5] "param_StdErrors" "conf_int"        "results"
```

The `lm()` function provides a lot of other features such as the t-test statistic, p-values, R^2 value, Adjusted R^2. So, there must be a lot of other computations that are ongoing.

Analysis on Larger Dataset

We simulate a larger dataset and compare the speed to see if we get different results

```
set.seed(12)

sim.B <- rnorm(n = 5, mean = 0, sd = 10) # simulated betas

c1 <- rnorm(10000, mean = 15, sd = 2) # continuous var
c2 <- rnorm(10000, mean = 2, sd = 3) # continuous var
c3 <- rbinom(10000, size = c(0,1), prob = 0.75) # binary indicator
c4 <- rpois(n = 10000, lambda = 2) # count var

sim.X <- cbind(1, c1, c2, c3, c4)

sim.Y <- as.vector(sim.X %*% sim.B + rnorm(10000, 0,3)) # simulate Y's

summary(sim.Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -64.37  125.54  159.27  158.73  193.51  366.61
```

```
sim.data <- data.frame(cbind(Y = sim.Y, sim.X[,-1])) # save a dataframe version for lm()
colnames(sim.data)
```

```
## [1] "Y"  "c1" "c2" "c3" "c4"
```

Correctness

```r
lm.fit <- lm(formula = "Y ~ c1 + c2 + c3 + c4", data = sim.data)
OLS.fit <- tureen625::fit_OLS(design_X = sim.X, Y = sim.Y)

stdErr.lm <- summary(lm.fit)$coefficients[,"Std. Error"]

all.equal(as.vector(lm.fit$coefficients), as.vector(OLS.fit$param_estimates))
```

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

```r
all.equal(as.vector(stdErr.lm), as.vector(OLS.fit$param_StdErrors))
```

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

```r
all.equal(as.vector(OLS.fit$residuals), as.vector(lm.fit$residuals))
```

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

- Results are the same when it comes to parameter estimation, std. error calculation, and residuals

Efficiency

```r
microbenchmark(
    lm(formula = "Y ~ c1 + c2 + c3 + c4", data = sim.data),
    tureen625::fit_OLS(design_X = sim.X, Y = sim.Y)
)
```

```
## Unit: milliseconds
##                                                    expr    min      lq     mean
##  lm(formula = "Y ~ c1 + c2 + c3 + c4", data = sim.data) 2.9350 3.19740 4.391922
##         tureen625::fit_OLS(design_X = sim.X, Y = sim.Y) 2.1909 2.46565 3.408717
##   median     uq     max neval
##  3.99575 4.9624 14.8569   100
##  2.74705 4.0223 20.5614   100
```

- With a dataset that has 10000 observations, we note that both of the functions are very similar in terms of computational speed. Our function is a bit faster, but the difference is ignorable
- If you want to try out larger datasets, simply simulate data using the above code as a skeleton and just increase the size from 10000 to something larger
- You can also use other datasets from the `datasets` library to make comparisons if you really want to