STAT-S 610: Assignment 5

Luis Navarro

Github Repository

Here is the link for my repository: https://github.com/LuisNavarro07/Stat_Computing_A5

Commit History

I typed the following code into the terminal. These lines of code added the files to the main branch of my repository and commit to such changes. Hence, created a snapshot of the repository with the saved versions (until this point) of the files added.

```
## 2. Git init
git init
## 3. Git status
git status
## Create the files in R and then add them
git add benchmark_llr.R
git add llr_functions.R
git add test_llr.R
## check whether files were added
git status
## commit
git commit -m 'initial commit'
```

Speed Test 1

Typing **git checkout speed-test-1** changes you from the master branch to the speed-test-1 branch. Performs a similar action as **git switch speed-test-1**

```
git branch speed-test-1
git checkout speed-test-1
git add llr_functions_1.R
git commit -m 'speed-test-1'
```

These are the original functions that implemented the locally weighted regression.

```r
### Function LLR
llr = function(x, y, z, omega) {
    fits = sapply(z, compute_f_hat, x, y, omega)
    return(fits)
```

```
}

### Function F Hat
compute_f_hat = function(z, x, y, omega) {
    Wz = make_weight_matrix(x, z, omega)
    X = make_predictor_matrix(x)
    f_hat = c(1, z) %*% solve(t(X) %*% Wz %*% X) %*% t(X) %*% Wz %*% y
    return(f_hat)
}
```

The functions below are the updated versions. I changed the name of the functions according to each speed test. For this update what I did was write how the matrix $X^T W_z X$ looks like. This is a 2×2 matrix with the following structure. Denote W_d as a vector with the diagonal elements of W_z. So W_{di} refers to the ith element of the diagonal.

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

where

$$d_1 = \sum_{i \in N} W_{di}$$

$$d_2 = \sum_{i \in N} W_{di} x_i$$

$$d_3 = \sum_{i \in N} W_{di} x_i^2$$

Similarly for the matrix $X^T W_z Y$. This ends up being a 2×1 vector:

$$\begin{pmatrix} d_4 \\ d_5 \end{pmatrix}$$

where

$$d_4 = \sum_{i \in N} W_{di} y_i$$

$$d_5 = \sum_{i \in N} W_{di} x_i y_i$$

So basically what I did was to compute manually all the matrix multiplications. I used the apply command to do the sums. Since the inner matrix is a 2×2, the inverse can be computed easily with the determinant formula. The factor to multiply d_1, d_2, d_3 is given by:

$$\gamma = \frac{1}{d_1 d_3 - d_2^2}$$

Define $\hat{d}_j = \gamma d_j$. With these numbers the whole multiplication $(X^T W_z X)^{-1} X^T W_z Y$ is reduced to a vector

$$\begin{pmatrix} c_1 = d_4\hat{d}_3 - d_5\hat{d}_2 \\ c_2 = -d_4\hat{d}_2 + d_5\hat{d}_1 \end{pmatrix}$$

and $\hat{f}(z)$ is given by the following expression. Note that in this notation, z is a scalar.

$$\hat{f}(z) = c_1 + c_2 z$$

```
### Function LLR
llr_3 = function(x, y, z, omega) {
    fits = sapply(z, compute_f_hat_3, x, y, omega)
    return(fits)
}


### Function F Hat
compute_f_hat_3 = function(z, x, y, omega) {
    ### Change this line --> Wz = make_weight_matrix(x, z, omega) For this line
    ### --> Wz = diag(make_weight_matrix(x, z, omega))
    Wz = diag(make_weight_matrix(x, z, omega))
    X = make_predictor_matrix(x)
    c <- matrix_product(x, y, Wz)
    ### Change this line --> f_hat = c(1, z) %*% solve(inner_mat(x,Wz)) %*%
    ### t(X) %*% Wz %*% y For this lines --> f_hat = c[1] + c[2]*z
    f_hat = c[1] + c[2] * z
    return(f_hat)
}

### Change to the f_hat line of code using apply First thing to note is that
### t(X) %*% Wz %*% X is always a 2x2 matrix, where the entries of the matrix
### are easy to compute
matrix_product <- function(x, y, Wz) {
    ### Define all the multiplications in vectorized form. In this case all
    ### should be nX1 vectors
    objects <- list(Wz, Wz * x, Wz * x * x, Wz * y, Wz * x * y)
    sums <- as.double(lapply(objects, sum))
    inv_fac <- 1/((sums[1] * sums[3]) - (sums[2]^2))
    inv_val <- sums[1:3] * inv_fac
    #
    c1 <- sums[4] * inv_val[3] - sums[5] * inv_val[2]
    c2 <- -sums[4] * inv_val[2] + sums[5] * inv_val[1]

    c <- c(c1, c2)
    return(c)
}
```

Which version works faster?

Using the microbenchmark function I compare the speed of these two implementations. For simplicity, I'll stick to the following testing environment.

```r
n = 15
## a very simple regression model
x = rnorm(n)
y = rnorm(x + rnorm(n))
z = seq(-1, 1, length.out = 100)
omega = 1
```

It seems to be that my implementation improves slightly the computation speed. I think it makes sense since I am reducing computational burden by avoiding matrix multiplication.

```r
summary(speedtests)[1:2, ]
```

```
##                    expr     min       lq     mean  median       uq      max
## 1   llr(x, y, z, omega) 67.8943 72.42495 76.27262 74.2865 76.57585 196.8835
## 2 llr_3(x, y, z, omega) 63.3205 68.77655 71.93052 70.4063 72.42735 255.0169
##   neval
## 1  1000
## 2  1000
```

Speed Test 2

For this second implementation I used the sweep function to simplify some calculations by defining them as objects. Sweep basically applies the same arithmetic operation to the columns (or rows) of a matrix. As explained above, most of the matrix multiplications done lead to a vector where each entry is $w_i x_i$. So I used the sweep function to create the matrix $X^T W z$, and then just multiply it by the remaining terms. The good part is that these terms appears twice in the equation so I can recycle the object. These are the updated functions.

```r
### Function LLR
llr_2 = function(x, y, z, omega) {
    fits = sapply(z, compute_f_hat_2, x, y, omega)
    return(fits)
}

### Function F Hat
compute_f_hat_2 = function(z, x, y, omega) {
    ### Change this line --> Wz = make_weight_matrix(x, z, omega) For this line
    ### --> Wz = diag(make_weight_matrix(x, z, omega))
    Wz = diag(make_weight_matrix(x, z, omega))
    X = make_predictor_matrix(x)
    ### Change this line --> f_hat = c(1, z) %*% solve(t(X) %*% Wz %*% X) %*%
    ### t(X) %*% Wz %*% y Sweep Function
    xw <- sweep(t(X), MARGIN = 2, STATS = Wz, FUN = "*")
    ### Sweep Matrix
    sweep_matrix <- xw %*% X
    ### Vectorized Calculation of the Last Term
    xwy <- xw %*% y
    f_hat = c(1, z) %*% solve(sweep_matrix) %*% xwy
    return(f_hat)
}
```

Which one runs faster?

Again, I'll use the microbenchmark function to test the speed. It seems that my implementation through the sweep functions slows down the average time considerably.

```
summary(speedtests)
```

```
##                    expr     min       lq     mean   median       uq      max
## 1   llr(x, y, z, omega) 67.8943 72.42495 76.27262 74.28650 76.57585 196.8835
## 2 llr_3(x, y, z, omega) 63.3205 68.77655 71.93052 70.40630 72.42735 255.0169
## 3 llr_2(x, y, z, omega) 71.4403 77.81230 81.69842 79.79305 82.57780 229.9733
##   neval
## 1  1000
## 2  1000
## 3  1000
```

Git Log Graph

This is how it looks on my end. It shows the number of commits, and in which branch they were made.

```
S C:\Users\luise\OneDrive - Indiana University\Statistical Computing\Assignments\A5_Git> git log --grapl
* commit 357b75afb43d9f2f225af17dec7cc821c8b3fffa (HEAD -> speed-test-1, statln/speed-test-1)
| Author: LuisNavarro07 <lunavarr@iu.edu>
| Date:   Thu Oct 27 16:41:40 2022 -0400
|
|     update 3
|
| * commit 21a0f079195e291241576cc46241838ff913322c (statln/master, master)
| | Author: LuisNavarro07 <lunavarr@iu.edu>
| | Date:   Thu Oct 27 16:30:10 2022 -0400
| |
| |     update 2
| |
| | * commit 4d5f09ca22195e007a7389ccbc17a64c2d2103b1 (statln/speed-test-2, speed-test-2)
| |/  Author: LuisNavarro07 <lunavarr@iu.edu>
|/|   Date:   Thu Oct 27 02:01:43 2022 -0400
| |
| |       speed-test-2
| |
* | commit 3ffcccd8b4398afc8a633c7adccf47b828a69f16
|/  Author: LuisNavarro07 <lunavarr@iu.edu>
|   Date:   Thu Oct 27 01:58:32 2022 -0400
|
|       speed-test-1
|
* commit 193e4c1819ed2cf5ec6ae10a38f97108aca62d60
  Author: LuisNavarro07 <lunavarr@iu.edu>
  Date:   Thu Oct 27 01:51:12 2022 -0400

      initial commit
(END)
```