Homework 4

Due October 14, 2020

2020-09-08

For each assignment, turn in by the due date/time. Late assignments must be arranged prior to submission. In every case, assignments are to be typed neatly using proper English in Markdown.

The last couple of weeks, we spoke about vector/matrix operations in R, discussed how the apply family of functions can assist in row/column operations, and how parallel computing in R is enabled. Combining this with previous topics, we can write functions using our Good Programming Practices style and adhere to Reproducible Research principles to create fully functional, redable and reproducible code based analysis in R. In this homework, we will put this all together and actually analyze some data. Remember to adhere to both Reproducible Research and Good Programming Practices, ie describe what you are doing and comment/indent code where necessary.

R Vector/matrix manipulations and math, speed considerations R's Apply family of functions Parallel computing in R, foreach and dopar

Problem 2: Using the dual nature to our advantage

Sometimes using a mixture of true matrix math plus component operations cleans up our code giving better readibility. Suppose we wanted to form the following computation:

- $while(abs(\Theta_0^i - \Theta_0^{i-1})$ AND $abs(\Theta_1^i - \Theta_1^{i-1}) > tolerance)$ {

$$\begin{aligned}\Theta_0^i &= \Theta_0^{i-1} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_0(x_i) - y_i)\\ \Theta_1^i &= \Theta_1^{i-1} - \alpha\frac{1}{m}\sum_{i=1}^{m}((h_0(x_i) - y_i)x_i)\end{aligned}$$

}

Where $h_0(x) = \Theta_0 + \Theta_1 x$.

Given $\mathbf{X}$ and $\vec{h}$ below, implement the above algorithm and compare the results with lm(h~0+$\mathbf{X}$). State the tolerance used and the step size, α.

```
set.seed(1256)
theta <- as.matrix(c(1,2),nrow=2)
X <- cbind(1,rep(1:10,10))
h <- X%*%theta+rnorm(100,0,0.2)
```

Problem 3

The above algorithm is called Gradient Descent. This algorithm, like Newton's method, has "hyperparameters" that are determined outside the algorithm and there are no set rules for determing what settings to use. For gradient descent, you need to set a start value, a step size and tolerance.

Part a. Using a step size of $1e^{-7}$ and tolerance of $1e^{-9}$, try 10000 different combinations of start values for β_0 and β_1 across the range of possible β's +/-1 from true determined in Problem 2, making sure to take advantages of parallel computing opportunities. In my try at this, I found starting close to true took 1.1M iterations, so set a stopping rule for 5M. Report the min and max number of iterations along with the starting values for those cases. Also report the average and stdev obtained across all 10000 β's.

Part b. What if you were to change the stopping rule to include our knowledge of the true value? Is this a good way to run this algorithm? What is a potential problem?

Part c. What are your thoughts on this algorithm?

Problem 4: Inverting matrices

Ok, so John Cook makes some good points, but if you want to do:

$$\hat{\beta} = (X'X)^{-1}X'\underline{y}$$

what are you to do?? Can you explain what is going on?

Problem 5: Need for speed challenge

In this problem, we are looking to compute the following:

$$y = p + AB^{-1}(q - r) \tag{1}$$

Where A, B, p, q and r are formed by:

```
set.seed(12456)

G <- matrix(sample(c(0,0.5,1),size=16000,replace=T),ncol=10)
R <- cor(G) # R: 10 * 10 correlation matrix of G
C <- kronecker(R, diag(1600)) # C is a 16000 * 16000 block diagonal matrix
id <- sample(1:16000,size=932,replace=F)
q <- sample(c(0,0.5,1),size=15068,replace=T) # vector of length 15068
A <- C[id, -id] # matrix of dimension 932 * 15068
B <- C[-id, -id] # matrix of dimension 15068 * 15068
p <- runif(932,0,1)
r <- runif(15068,0,1)
C<-NULL #save some memory space
```

Part a.

How large (bytes) are A and B? Without any optimization tricks, how long does the it take to calculate y?

Part b.

How would you break apart this compute, i.e., what order of operations would make sense? Are there any mathmatical simplifications you can make? Is there anything about the vectors or matrices we might take advantage of?

Part c.

Use ANY means (ANY package, ANY trick, etc) necessary to compute the above, fast. Wrap your code in "system.time({})", everything you do past assignment "C <- NULL".

Problem 3

a. Create a function that computes the proportion of successes in a vector. Use good programming practices.

b. Create a matrix to simulate 10 flips of a coin with varying degrees of "fairness" (columns = probability) as follows:

```
set.seed(12345)
P4b_data <- matrix(rbinom(10, 1, prob = (31:40)/100), nrow = 10, ncol = 10, byrow = FALSE)
```

c. Use your function in conjunction with apply to compute the proportion of success in P4b_data by column and then by row. What do you observe? What is going on?

d. You are to fix the above matrix by creating a function whose input is a probability and output is a vector whose elements are the outcomes of 10 flips of a coin. Now create a vector of the desired probabilities. Using the appropriate apply family function, create the matrix we really wanted above. Prove this has worked by using the function created in part a to compute and tabulate the appropriate marginal successes.

Problem 4

In Homework 4, we had a dataset we were to compute some summary statistics from. The description of the data was given as "a dataset which has multiple repeated measurements from two devices by thirteen Observers". Where the device measurements were in columns "dev1" and "dev2". Reimport that dataset, change the names of "dev1" and "dev2" to x and y and do the following:

1. create a function that accepts a dataframe of values, title, and x/y labels and creates a scatter plot
2. use this function to create:
   (a) a single scatter plot of the entire dataset
   (b) a seperate scatter plot for each observer (using the apply function)

Problem 5

Our ultimate goal in this problem is to create an annotated map of the US. I am giving you the code to create said map, you will need to customize it to include the annotations.

Part a. Get and import a database of US cities and states. Here is some R code to help:

```
#we are grabbing a SQL set from here
# http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip

#download the files, looks like it is a .zip
library(downloader)
download("http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip",dest="us_cities_sta
unzip("us_cities_states.zip", exdir="./")

#read in data, looks like sql dump, blah
library(data.table)
states <- fread(input = "./us_cities_and_states/states.sql",skip = 23,sep = "'", sep2 = ",", header
### YOU do the CITIES
### I suggest the cities_extended.sql may have everything you need
### can you figure out how to limit this to the 50?
```

Part b. Create a summary table of the number of cities included by state.

Part c. Create a function that counts the number of occurances of a letter in a string. The input to the function should be "letter" and "state_name". The output should be a scalar with the count for that letter.

Create a for loop to loop through the state names imported in part a. Inside the for loop, use an apply family function to iterate across a vector of letters and collect the occurance count as a vector.

```
##pseudo code
letter_count <- data.frame(matrix(NA,nrow=50, ncol=26))
getCount <- function(what args){
    temp <- strsplit(state_name)
    # how to count??
    return(count)
}
for(i in 1:50){
    letter_count[i,] <- xx-apply(args)
}
```

Part d.

Create 2 maps to finalize this. Map 1 should be colored by count of cities on our list within the state. Map 2 should highlight only those states that have more than 3 occurances of ANY letter in thier name.

Quick and not so dirty map:

```
#https://cran.r-project.org/web/packages/fiftystater/vignettes/fiftystater.html
library(ggplot2)
library(fiftystater)

data("fifty_states") # this line is optional due to lazy data loading
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)
# map_id creates the aesthetic mapping to the state name column in your data
p <- ggplot(crimes, aes(map_id = state)) +
  # map points to the fifty_states shape data
  geom_map(aes(fill = Assault), map = fifty_states) +
  expand_limits(x = fifty_states$long, y = fifty_states$lat) +
  coord_map() +
  scale_x_continuous(breaks = NULL) +
  scale_y_continuous(breaks = NULL) +
  labs(x = "", y = "") +
  theme(legend.position = "bottom",
        panel.background = element_blank())

p
#ggsave(plot = p, file = "HW6_Problem6_Plot_Settlage.pdf")
```

Problem 2

Bootstrapping

Recall the sensory data from five operators:
http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat

Sometimes, you really want more data to do the desired analysis, but going back to the "field" is often not an option. An often used method is bootstrapping. Check out the second answer here for a really nice and detailed description of bootstrapping: https://stats.stackexchange.com/questions/316483/manually-bootstrapping-linear-regression-in-r.

What we want to do is bootstrap the Sensory data to get non-parametric estimates of the parameters.

Assume that we can neglect item in the analysis such that we are really only interested in a linear model lm(y~operator).

Part a. First, the question asked in the stackexchange was why is the supplied code not working. This question was actually never answered. What is the problem with the code? If you want to duplicate the code to test it, use the quantreg package to get the data.

Part b. Bootstrap the analysis to get the parameter estimates using 100 bootstrapped samples. Make sure to use system.time to get total time for the analysis. You should probably make sure the samples are balanced across operators, ie each sample draws for each operator.

Part c. Redo the last problem but run the bootstraps in parallel (`cl <- makeCluster(8)`), don't forget to `stopCluster(cl)`). Why can you do this? Make sure to use system.time to get total time for the analysis.

Create a single table summarizing the results and timing from part a and b. What are your thoughts?

Problem 3

Newton's method gives an answer for a root. To find multiple roots, you need to try different starting values. There is no guarantee for what start will give a specific root, so you simply need to try multiple. From the plot of the function in HW4, problem 8, how many roots are there?

Create a vector (`length.out=1000`) as a "grid" covering all the roots and extending +/-1 to either end.

Part a. Using one of the apply functions, find the roots noting the time it takes to run the apply function.

Part b. Repeat the apply command using the equivelant parApply command. Use 8 workers. `cl <- makeCluster(8).`

Create a table summarizing the roots and timing from both parts a and b. What are your thoughts?

Problem 9

Finish this homework by pushing your changes to your repo.

Only submit the .Rmd and .pdf solution files. Names should be formatted HW4_pid.Rmd and HW4_pid.pdf