ORF350 (Spring 2017) Analysis of Big Data **Assignment: 1**

Central Limit Theory, Law of Large Numbers and R Programming

Lecturer: Han Liu *Email: hanliu@princeton.edu*

The due date is Feb 24, Friday 5:00pm. Please submit your hard copy at the ORF 350 dropbox in the Sherrerd Hall student lounge. Electronic submissions will not be accepted unless there is an extreme and compelling case.

Q1. Law of Large Numbers and Central Limit Theorem (20 points)

This question helps you to better understand the concepts of convergence in probability and convergence in distribution. In particular, you will visualize the Law of Large Numbers and Central Limit Theorem for a Discrete Distribution:

$$\mathbb{P}(X = 1) = \mathbb{P}(X = -1) = 1/2.$$

Generate $N = 10,000$ datasets, each of which has n data points.
(Hint: Write a function in R that samples from the uniform distribution between 0 and 1 using the built-in `runif` function. If the result is less than 0.5, set it to -1. Otherwise, set it to 1.)
Let $\overline{X}_n^{(i)}$ be the average of i^{th} dataset, $\mu = EX$ and $\sigma^2 = \mathrm{Var}(X)$. We consider $n = \{10, 100, 1000, 10000\}$ for this simulation.
(Hint: You will not need the individual data points from each dataset. Therefore, to save memory, you need only store the $\overline{X}_n^{(i)}$ rather than all the data points. It is highly recommended that you do this to avoid freezing or crashing your computer.)

Plot and intepret the following:

(a) $\log_{10}(n)$ v.s. $\overline{X}_n^{(1)} - \mu$;
(Hint: This plot illustrates how the deviation $\overline{X}_n^{(1)} - \mu$ converges to 0 as n goes to infinity).

(b) Draw $\log_{10}(n)$ v.s. $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{|\overline{X}_n^{(i)} - \mu| > \epsilon\}$ for $\epsilon = 0.5, \epsilon = 0.1, \epsilon = 0.05$;
(Hint 1: This plot illustrates the law of large numbers. Please explain why.)
(Hint 2: For some statement S, the indicator function $\mathbb{1}\{S\}$ is defined as $\mathbb{1}\{S\} = 1$ if S is true and $\mathbb{1}\{S\} = 0$ otherwise.)

(c) Draw histograms and Q-Q plots of $\sqrt{n}(\overline{X}_n^{(i)} - \mu)/\sigma$ for N datsets for $n = 10, n = 1,000, n = 10,000$. You may choose your histogram bins or you may let R choose automatically–any meaningful plot will do.
(Hint: This plot illustrates the Central Limit Theorem. Please explain why.)

(d) Generate i.i.d. standard normal $Y_1, \ldots, Y_N$ that are independent to previous random variables. Plot

$$\log_{10}(n) \text{ v.s. } \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{|\sqrt{n}(\overline{X}_n^{(i)} - \mu)/\sigma - Y_i| > \epsilon\} \text{ for } \epsilon = 0.001.$$

Random sampling so even though pdf and cdf converge, you won't get the same values each time

(Hint: This plots illustrates the difference between convergence in probability and convergence in distribution. Explain why this plot shows that $\sqrt{n}(\overline{X}_n^{(i)} - \mu)/\sigma$ does not converge to Y in probability.)

Instructions: This problem tries to familiarize you with `R` programming. You should produce 11 graphics in total – one plot for (a), three plots for (b), six plots for (c), and one plot for (d). You may format and combine the plots together for each part in whichever way you like, but we must be able to clearly read and interpret your results.

Q2. Basic R Programming for Big Data (20 points)

This problem will help you practice coding in `R` under a big data setting. The dataset comes from a dating website "LibimSeti." It consists of two files, `ratings.dat` and `users.Rdata`. The `ratings.dat` contains 3,000,000 ratings of profiles made by LibimSeTi users. It is organized as a comma-separated matrix with 3,000,000 rows and 3 columns. The `users.Rdata` contains the basic information of the users. It has $135,358$ rows and 7 columns. `Readme.txt` contains a detailed description of the data.

Question 2.1 Load `ratings.dat` into `R` using the package `bigmemory`, and name the columns by UserID, ProfileID, Rating. To evaluate every profile's score, we use the weighted rank (also used by IMDB):

$$\text{Weighted Rank (WR)} = (v \div (v+m)) \times R + (m \div (v+m)) \times C, \text{ where}$$

$$\begin{array}{rcl} R & = & \text{average rate for the profile} \\ v & = & \text{number of votes for the profile} \\ m & = & 4182 \text{ (the 250th largest number of ratings for a single profile)} \\ C & = & \text{the mean rate across the whole data.} \end{array}$$

Write an `R` function `weighted.rank(ProfileID)` returning the weighted rank of the profile with its ID as input. Use the functions `which` and `apply` to compute the weighted ranks of all the profiles who were rated by UserID 100. Report your result by plotting a histogram of those scores you obtained. You can also use the `mwhich` and `lapply` functions. [This teaches you how to use bigmemory to load big data and how to write a R function.]

Question 2.2 Load `users.Rdata` into `R`. Then you will have a matrix `User` in your working environment. Based on `users.Rdata` and `ratings.dat`, plot the boxplots of ratings from 1) male users coming from New York State and 2) female users coming from California. (Hint: you can use the `unique` or `grep` functions to find out the different ways each state is recorded in the dataset.)

Question 2.3 In order to predict a user's rating for a profile, fit a linear regression using the average rating given by the user and the average rating of the profile from all users as predictors. Report your results. Specifically, report the R-squared value and the three coefficients (intercept, coefficient for average rating given by a user and coefficient for average ratings given to a profile). Informally, the model is

$$\text{(User } i\text{'s rating on Profile } j) = \theta_1 + \theta_2(\text{Average Rating given by User } i) + \theta_3(\text{Average Rating given to Profile } j) + \epsilon_{ij}$$

where $\{\theta_1, \theta_2, \theta_3\} \in \mathbb{R}^3$ and ϵ_{ij} is a Gaussian noise term. You are required to 1) run the regression over the entire dataset using the the `R` function `biglm` in the package of `biganalytics`; 2) apply the subsampling technique, which means you do regression over multiple small subsamples respectively and take the mean of coefficients fitted by all subsamples as the aggregated estimand of coefficients. You can refer to Hint 3 below for further instructions on the subsampling technique. You are encouraged to compare the difference between the two methods in terms of the coefficient values, the R-squared value and the CPU time. You might need to recall how to calculate the R-squared value from previous courses.

Hint 1: Simple things become painful under the big data setting. The most time-consuming part is to calculate average ratings involved in the linear model. The most natural way of doing this is to use `mwhich` for each user (profile) to locate the ratings associated with the given user (profile) and then do the average, but once you run the code you will feel the pain: the progress is very slow. Here we suggest another way of doing this, which we hope can make your life easier. Try to understand the following code for calculation of average ratings, and if you like it, you can copy the following code from the given `R` script file `AveRating.r`. :)

```
N=3000000   # number of rating records
Nu=135359   # maximum of UserID
Np=220970   # maximum of ProfileID
user.rat=rep(0,Nu)  # user.rat[i] denotes the sum of ratings given by user i
user.num=rep(0,Nu)  # user.num[i] denotes the number of ratings given by user i
profile.rat=rep(0,Np)  # profile.rat[i] denotes the sum of ratings given to profile i
profile.num=rep(0,Np)  # user.rat[i] denotes the number of ratings given to profile i
for (i in 1:N){ # In each iteration, we update the four arrays, i.e. user.rat,
    user.num, profile.rat and profile.num, using one rating record.
  user.rat[X[i,'UserID']]=user.rat[X[i,'UserID']]+X[i,'Rating']   # The matrix X here
      comes from the file 'ratings.dat'
  user.num[X[i,'UserID']]=user.num[X[i,'UserID']]+1
  profile.rat[X[i,'ProfileID']]=profile.rat[X[i,'ProfileID']]+X[i,'Rating']
  profile.num[X[i,'ProfileID']]=profile.num[X[i,'ProfileID']]+1
  if (i %% 10000==0) print(i/10000)
}
user.ave=user.rat/user.num
profile.ave=profile.rat/profile.num
X1=X
colnames(X1)=c('UsrAveRat','PrfAveRat','Rat')
X1[,'UsrAveRat']=user.ave[X[,'UserID']]
X1[,'PrfAveRat']=profile.ave[X[,'ProfileID']]  # X1 is the new data matrix we will
    work with in regression.
```

With the code given above, we can finish calculating all the average ratings by only one for-loop over the entire dataset.

Hint 2: Using `biglm` on the entire dataset can take around one hour on a descent laptop. Be patient. Here are several tips for running codes on big data: a) Test you code first on a small dataset; b) In order to know that your codes are still working, you should have your code display the current progress of your program (the percentage of work finished, the number of loops it is working on, etc.) For example, if there is a huge for-loop in your code, print an asterisk('*') after every 10,000 iterations; c) Have your code automatically save your results as an rda file every so often. This allows you to check your results in a different R console while the program is running and retain your results if your R program prematurely terminates; d) You can run R from the command line to avoid using the R Studio interface. This will typically make your code faster.

Hint 3: We first need to figure out what the data matrix $\mathbf{X}_1$ is for our regression task. It is worth noting that $\mathbf{X}_1$ is now different from the matrix $\mathbf{X}$ that we load from `ratings.dat`. $\mathbf{X}_1$ is of the same dimensions as $\mathbf{X}$, but to get $\mathbf{X}_1$, we need to replace the ProfileID's and UserID's in $\mathbf{X}$ with the associated average ratings as explained in the model. (See the code given in Hint 1 for reference.) The subsampling technique essentially means that we randomly choose a small number rows of $\mathbf{X}_1$, whose row indices form a set I, and do regression only over X_{1_I}, which is a sub-matrix of $\mathbf{X}_1$ that consists of only rows I. Since we use only a small number of samples, the regression coefficients we get will not be accurate. A natural way to solve this problem is to apply this subsampling trick for multiple times and average the coefficients we get. In this way, we expect the result to be more stable and accurate.