STAT243: Problem Set 6

Zicheng Huang

11/01/2017

1.

(a) The goal of the simuation study is to investigate the finite sample properties of the likelihood ratio test in determining the mixture distribution, more specifically, choosing between a k0-component normal mixture versus a k1-component normal mixture. The metrics being considered in assessing the method are the significance level and the statistical power of the likelihood ratio test statistic.

(b) In designing their simuation study, the authors first need to choose the value of k0 and k1 for the two competing distributions, together with the values of the mixing proportion and the distance (or degree of seperation) between the components of the competing distribution. They also need to determine the different sample sizes under investigation and how many samples should be generated to make the analysis. The size of the samples and the level of separation of the components of the competing distribution are some key aspects of the data generating mechanism that likely affect the statistical power. The data generating mechanism considered in this paper does not consider the scenario where the components of the competing distributions have difference variance. This might be something useful to consider.

(c) Their tables in general do a good job of presenting the simuation results by showing the comparison of significance level and statistical power under different sample sizes, mixing proportions, and magnitudes of separation between components. In some cases, a static graphical representation of the results may also be a good choice. However, it might be too complicated to construct a good static graphic when the number of changing variables gets larger. In this case, we might need to implement a dynamic graphical representation of the simuation results.

(d) The values in tables 2 and 4 represents the simulated power, which is the percentage of the 1000 repetitions of the simulated samples from the alternative hypothesis distribution where the null hypothesis is rejected based on the corresponding test statistics of the simulated samples. In general, the more separated the components of the mixture distribution where the simulated samples are generated from, the easier to reject the null hypothesis where the number of components in the mixture distribution is always smaller. This is in general consistent with the results shown in the tables 2 and 4. Table 2 suggests that the statistical power is low for sample size less than 200 where the components of the mixture distribution is not well separated. Under the condition of components being well separated, at least a sample size of 100 is needed to be able to observe a reasonable statistical power. According to the table, we cannot see a relationship between power and the mixing proportion. By observing that the unadjusted test tends to reject the null hypothesis more often does the adjusted test, it is reasonable in that the convergence of the unadjusted test is not as accurate. Table 4 presents similar results, with two sets of mixing proportions, that

larger separation between components and larger sample sizes are correlated with a larger statistical power.

(e) The authors decide to use 1000 simulations because the more simulation the better the approximation of the distribution and better accuracy in terms of the rejection region. The number of repetition will affect the simulated significance level summarized in Table 1 and 3. Thus, authors may decide to use 1000 simulations in order to get a better rate of convergence of the test statistic toward the asymptotic distribution so as to have a better representation of the underline population. The simulation size of 10 is too small to give enough information needed for the distribution and not representative of the population. If more than 1000 simulations can give better results and convergence, then it might be useful to take more than 1000 simulations. However, if the implementation of more simulations does not lead to a better result, instead, only similar level of results, then 1000 simulations may be enough. Moreover, it is often the time being too costly to run too large number of repetitions.

2.

```
library(RSQLite)
drv <- dbDriver("SQLite")
dir <- 'C:/HZC/Berkeley/STAT243/ps6' # absolute path to where the .db file is
dbFilename <- 'stackoverflow-2016.db'
db <- dbConnect(drv, dbname = file.path(dir, dbFilename))

result <- dbGetQuery(db, "select distinct userid, displayname from
                      users U, questions Q, questions_tags T where
                      U.userid = Q.ownerid and
                      Q.questionid = T.questionid and
                      tag = 'r' and U.userid not in
                      (select distinct userid from
                      users U, questions Q, questions_tags T where
                      U.userid = Q.ownerid and
                      Q.questionid = T.questionid and
                      tag = 'python')")

# number of users satisfying the required condition
length(result$userid)

## [1] 18611
```

We first select the distinct userid from whom has asked R-related questions by choosing the userid from users who has ownerid in a question taged as related to R. Then select the distinct userid from whom has asked Python-related questions in a similar fashion. Then we need to filter out those users who has asked both R-related and Python-related questions by filter out those userid having ownerid in questions related both to R and Python.

3.

With the Wikipedia traffic data, I am trying to investigate the difference between the number of hits per day on webpages related to Barack Obama and that of John McCain, given that they were competing for the president of the United States. Below are the Spark code:

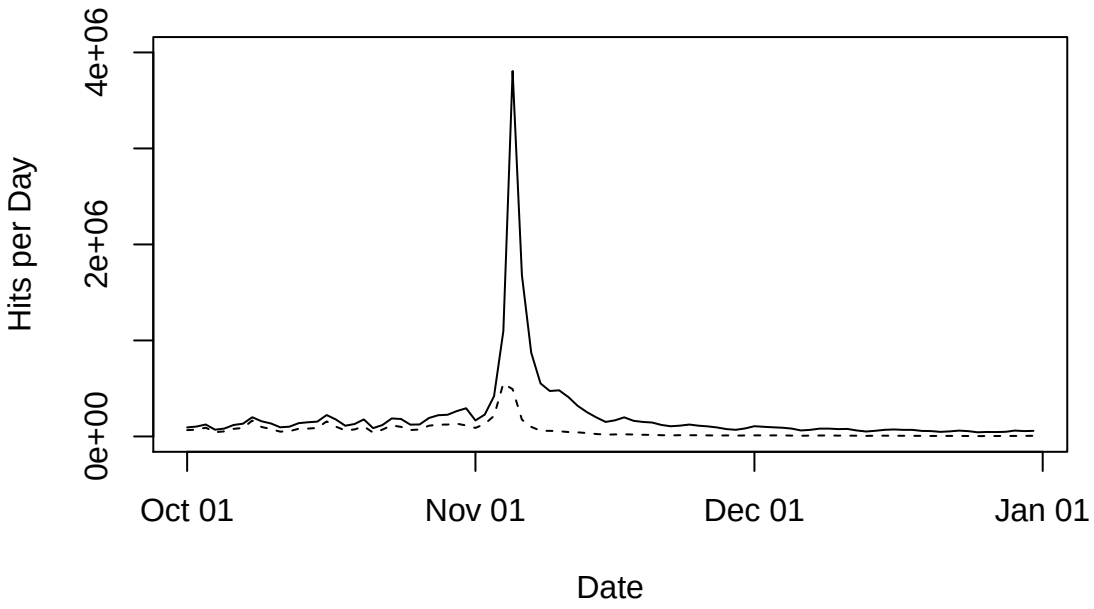
```
# define input directory
dir = '/global/scratch/paciorek/wikistats_full'

### read data  ###

lines = sc.textFile(dir + '/' + 'dated')

### filter to sites of interest ###

import re
from operator import add

# function for finding traffic data related to Barack Obama
def findBO(line, regex = "Barack_Obama", language = None):
    vals = line.split(' ')
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[3])
    if tmp is None or (language != None and vals[2] != language):
        return(False)
    else:
        return(True)

# function for finding traffic data related to John McCain
def findJM(line, regex = "John_McCain", language = None):
    vals = line.split(' ')
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[3])
    if tmp is None or (language != None and vals[2] != language):
        return(False)
    else:
        return(True)

# find all traffic data related to Barack Obama
obama = lines.filter(findBO).repartition(480)

# find traffic data related to John McCain
```

```
mccain = lines.filter(findJM).repartition(480)

### map-reduce step to sum hits across date-time-language triplets ###

def stratify(line):
    vals = line.split(' ')
    return(vals[0] + '-' + vals[1] + '-' + vals[2], int(vals[4]))

# sum number of hits related to Barack Obama for each date-time-language value
countsBO = obama.map(stratify).reduceByKey(add)

# sum number of hits related to John McCain for each date-time-language value
countsJM = mccain.map(stratify).reduceByKey(add)

### map step to prepare output ###

def transform(vals):
    key = vals[0].split('-')
    return(",".join((key[0], key[1], key[2], str(vals[1]))))

### output to file ###

# define output directory
outdir = '/global/scratch/zicheng_huang'
outputDirBO = outdir + '/' + 'BO'
outputDirJM = outdir + '/' + 'JM'

# output to file
countsBO.map(transform).repartition(1).saveAsTextFile(outputDirBO)
countsJM.map(transform).repartition(1).saveAsTextFile(outputDirJM)
```

Once we have the output from the above Spark codes, we make analysis by ploting out the number of hits per day for each person and see how the difference evolves over time.

```
library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##    filter, lag
## The following objects are masked from 'package:base':
##
##    intersect, setdiff, setequal, union
```

```
# read in the result processed in Spark
BO <- read.csv("Obama", header = F, na.strings = "", stringsAsFactors = F)
# name each columns
names(BO) <- c("Date", "Time", "Language", "NumOfHits")
# change the Date column to date format
BO$Date <- as.Date(as.character(BO$Date), "%Y%m%d")
# obtain a table containing date and hits per day
Obama <- BO %>% group_by(Date) %>% summarize(HitsPerDay = sum(NumOfHits))
# read in the result processed in Spark
JM <- read.csv("McCain", header = F, na.strings = "", stringsAsFactors = F)
# name each columns
names(JM) <- c("Date", "Time", "Language", "NumOfHits")
# change the Date column to date format
JM$Date <- as.Date(as.character(JM$Date), "%Y%m%d")
# obtain a table containing date and hits per day
McCain <- JM %>% group_by(Date) %>% summarize(HitsPerDay = sum(NumOfHits))
# make a plot
plot(Obama$Date, Obama$HitsPerDay,
     type = "l", xlab = "Date", ylab = "Hits per Day",
     ylim = c(0, 4000000))
lines(McCain$Date, McCain$HitsPerDay, lty = 2)
```



In the plot, the solid line represents the number of hits per day in the Wikipedia traffic data related to Barack Obama, and the dashed line represents the number of hits per day in the Wikipedia traffic data

related to John McCain. As we can see, before and after the days around Nov 4th, the difference in hits per day is relatively stable. During the several days around Nov 4th, when Barack Obama won the presidential campaign, the difference in hits really spikes, revealing that only the winner really got the public's attention.

4.

(a)

We used "sbatch" to submit a job with the file "ps6Q4.sh" which contains running the R script "ps6Q4.R" and the output is specified below in the R code, with the dimension of the resulting object saved in "dimension" to serve as a check.

```
# required packages
library(parallel)
library(doParallel)
library(foreach)
library(stringr)
# set number of cores being used
ncores <- as.integer(Sys.getenv("SLURM_CPUS_ON_NODE"))
registerDoParallel(ncores)
# set number of files being processed
nSub <- 959
# find and filters to only the rows that refer to pages where "Barack_Obama" appears
obama <- foreach(i=0:nSub,
                 .packages = c("stringr"),  # libraries to load onto each worker
                 .combine= c,               # how to combine results
                 .verbose=TRUE) %dopar%     # print statuses of each job
                 {
                   # file path
                   file <- paste("/global/scratch/paciorek/wikistats_full/dated_for_R/part-",
                                 str_pad(i, width=5, side="left", pad="0"), sep="")
                   # read in each file
                   table <- readLines(file)
                   # filter to only rows refer to pages where "Barack_Obama" appears
                   output <- grep("Barack_Obama", table, value = TRUE)
                   # return as part of the result object
                   output
                 }
# convert the result object to a data frame
df <- data.frame(do.call(rbind, strsplit(obama, " ", fixed = TRUE)))
# count the number of pages where "Barack_Obama" appears
dimension <- dim(df)[1]
# some sample pages
sample <- head(df)
# output to file the number of pages that is relevant  with Barack Obama
```

```
write.table(dimension, file='/global/home/users/zicheng_huang/dimension.txt')
# output some sample entries in the result to file
write.table(sample, file='/global/home/users/zicheng_huang/sample.txt')
```

Importing the resulting output file, we can see the number of pages where "Barack_Obama" appears is 433895.

```
read.csv("dimension.txt")

##          x
## 1 1 433895
```

(b)

Upon finish the "sbatch" job, we also obtain a file called "ps6Q4.out" which contains the running time of the job. Below I will show the information contained in the proc.time() in the "ps6Q4.out" file.

```
## proc.time()
##     user    system   elapsed
## 42639.366  4634.308  2617.266
```

By using 1 node with 24 cores, the running time is 2617.266 seconds. Assuming that we are running with 96 cores, with the assumption of perfect scalability, the running time should be $2617.226/4 = 654.3065$ seconds, which is less than the 15 minutes run by Spark code to do the filtering.

5(b)

You can store the Cholesky upper triangular matrix, U, in the block of memory that is used for the original matrix, A. During the Cholesky decomposition algorithm, calculation of row i of matrix U does not require any information from row 1 to i-1 of matrix A, only requiring information from Aii entry and Aij entry where j larger than i. In other words, we work through row by row of the original matrix to obtain matrix U, and never use entries in previous rows to calculate entries in matrix U. For example, when obtaining the third row of entries in matrix U, we do not need information from the first two rows of the original matrix A. Thus, we can store the resulting rows of matrix U that we obtained in the corresponding rows of the original matrix without overwrite anything required in the future calculation of the Cholesky.