

ps6

Kehsin Su Esther 3033114294

October 30, 2017

```
knitr::opts_chunk$set(tidy = TRUE, cache = TRUE)
library(pryr)

## Warning: package 'pryr' was built under R version 3.3.3
```

1 Q1

(Brief summary)

The goal of the simulation is to investigate the finite sample properties of the test. How fast does the test statistics(likelihood ratio statistics) converge in distribution to asymptotic distribution and the power. They use power under different samples, proportion, alpha levels to assess their method.

The author has chosen EM algorithm to use maximum likelihood method to obtain 100 sets of parameters. The sample were generated from standard normal distribution. Those values and sample sizes that used to generate null sample may impact the result. Maybe the author should consider more about difference variance of the two components.

The table reveals enough information for the test results. For table 1, personally, I would recommend transfer the row into columns and combine the result of inadjusted test and adjusted test to compare the result more intuitive. I would even calculate the difference of powers. Also, plot some graphs might help reader understand the result better.

When the sample size increase, the it will coverage to alpha. Overall, he results makes more sense when sample size increase and D value decrease.

In my view, more simulation would be more powerful for the hypotethis. Size of 10 might not strong enough. Most of the case, the size should at least be 100 to be representative, and not exceed 1000 for the efficiency. (general 10% of the population)

2 Q2

```
## @knitr database-access

library(RSQLite)

## Warning: package 'RSQLite' was built under R version 3.3.3

drv <- dbDriver("SQLite")
# relative or absolute path to where the .db file is
dir <- "C:/Users/Esther/Desktop/stat243-fall-2017-master/section/s07"
dbFilename <- "stackoverflow-2016.db"
```

```

db <- dbConnect(drv, dbname = file.path(dir, dbFilename))

## @knitr database-tables
dbListTables(db)

## [1] "answers"          "maxRepByQuestion" "questions"
## [4] "questionsAugment" "questions_tags"   "users"

dbListFields(db, "questions")

## [1] "questionid"      "creationdate" "score"          "viewcount"
## [5] "title"           "ownerid"

dbListFields(db, "users")

## [1] "userid"           "creationdate"   "lastaccessdate" "location"
## [5] "reputation"       "displayname"    "upvotes"         "downvotes"
## [9] "age"              "accountid"

dbListFields(db, "questions_tags")

## [1] "questionid" "tag"

# simple query to get 5 rows from a table
dbGetQuery(db, "select * from questions limit 5")

##   questionid      creationdate score viewcount
## 1   34552550 2016-01-01 00:00:03     0      108
## 2   34552551 2016-01-01 00:00:07     1      151
## 3   34552552 2016-01-01 00:00:39     2     1942
## 4   34552554 2016-01-01 00:00:50     0      153
## 5   34552555 2016-01-01 00:00:51    -1       54
##                                     title
## 1                                     Scope between methods
## 2      Rails - Unknown Attribute - Unable to add a new field to a form on create/update
## 3 Selenium Firefox webdriver won't load a blank page after changing Firefox preferences
## 4                                     Android Studio styles.xml Error
## 5                                     Java: reference to non-final local variables inside a thread
##   ownerid
## 1 5684416
## 2 2457617
## 3 5732525
## 4 5735112
## 5 4646288

dbGetQuery(db, "select * from users limit 5")

##   userid      creationdate      lastaccessdate
## 1    13 2008-08-01 04:18:04 2017-03-13 21:03:28
## 2    24 2008-08-01 12:12:53 2017-03-10 22:11:55
## 3    36 2008-08-01 12:43:55 2017-03-13 20:33:22
## 4    37 2008-08-01 12:44:00 2017-03-13 18:08:20
## 5    39 2008-08-01 12:44:55 2017-03-11 13:17:47
##                                     location reputation      displayname upvotes
## 1 Raleigh, NC, United States    157267 Chris Jester-Young    5198
## 2 London, United Kingdom        2321      sanmiguel        535
## 3 San Diego, CA                 4038          Pat          423
## 4 Georgetown, Canada            4269      Wally Lawless      257
## 5 Istanbul, Turkey              10172     huseyint          523

```

```
##      downvotes age accountid
## 1         210  37          9
## 2           3  34         17
## 3          10  36         27
## 4          18  36         28
## 5          21  32         30

dbGetQuery(db, "select * from questions_tags limit 5")

##      questionid      tag
## 1    34552711      c#
## 2    34552711    razor
## 3    34552711    flags
## 4    34552829 javascript
## 5    34552829    rxjs

rresult <- dbGetQuery(db, "select distinct userid from
      questions Q
      join questions_tags T on Q.questionid = T.questionid
      join users U on Q.ownerid = U.userid
      where tag = 'r' or tag = 'python'
      except
      select distinct userid from
      questions Q
      join questions_tags T on Q.questionid = T.questionid
      join users U on Q.ownerid = U.userid
      where tag = 'python'
      ")
nrow(rresult)

## [1] 18611
```

3 Q3

The following codes are refer from chapter 8.

```
#from pyspark import SparkFiles
dir = '/global/scratch/paciorek/wikistats_full'
#dir=$HOME
#lines = sc.textFile(dir+'/' + 'newdir')
lines = sc.textFile(dir + '/' + 'dated')

lines.getNumPartitions() # 16590 (480 input files) for full dataset

# note delayed evaluation
lines.count()
testLines = lines.take(10)
testLines[0]
testLines[9]

### filter to sites of interest ###

import re
from operator import add

def find(line, regex = "Disney", 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)

lines.filter(find).take(100) # pretty quick

# not clear if should repartition; will likely have small partitions if not
obama = lines.filter(find).repartition(480) # ~ 18 minutes for full dataset (but remember lazy evaluation)
obama.count() # 433k observations for full dataset
mydir = '/global/scratch/esther730'
outputDir = mydir+'/' + 'Dinsney-counts'
obama.saveAsTextFile(outputDir)
## @knitr map-reduce

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

def stratify(line):
    # create key-value pairs where:
    #   key = date-time-language
    #   value = number of website hits
    vals = line.split(' ')
    return(vals[0] + '-' + vals[1] + '-' + vals[2], int(vals[4]))

# sum number of hits for each date-time-language value
counts = obama.map(stratify).reduceByKey(add) # 5 minutes
# 128889 for full dataset

### map step to prepare output ###

def transform(vals):
    # split key info back into separate fields
    key = vals[0].split('-')
    return(",".join((key[0], key[1], key[2], str(vals[1]))))

### output to file ###

# have one partition because one file per partition is written out
mydir = '/global/scratch/esther730'
outputDir = mydir+'/' + 'Dinsney-counts'
counts.map(transform).repartition(1).saveAsTextFile(outputDir)

```

After above action, I got Disney data from the wikipages.

```

#copy the data from savio and rename it as disney_data
scp esther730@dtb.brc.berkeley.edu://global/scratch/esther730/Dinsney-counts2/part-00000 /mnt/c/Users

```

As showing in the plot, the highest hits appears three days before the Christmas.

```

# Note: refer from chapter 8 obama_plot.R
library(dplyr)

## Warning: package 'dplyr' was built under R version 3.3.3

```

```
##
## 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

library(chron)

## Warning: package 'chron' was built under R version 3.3.3

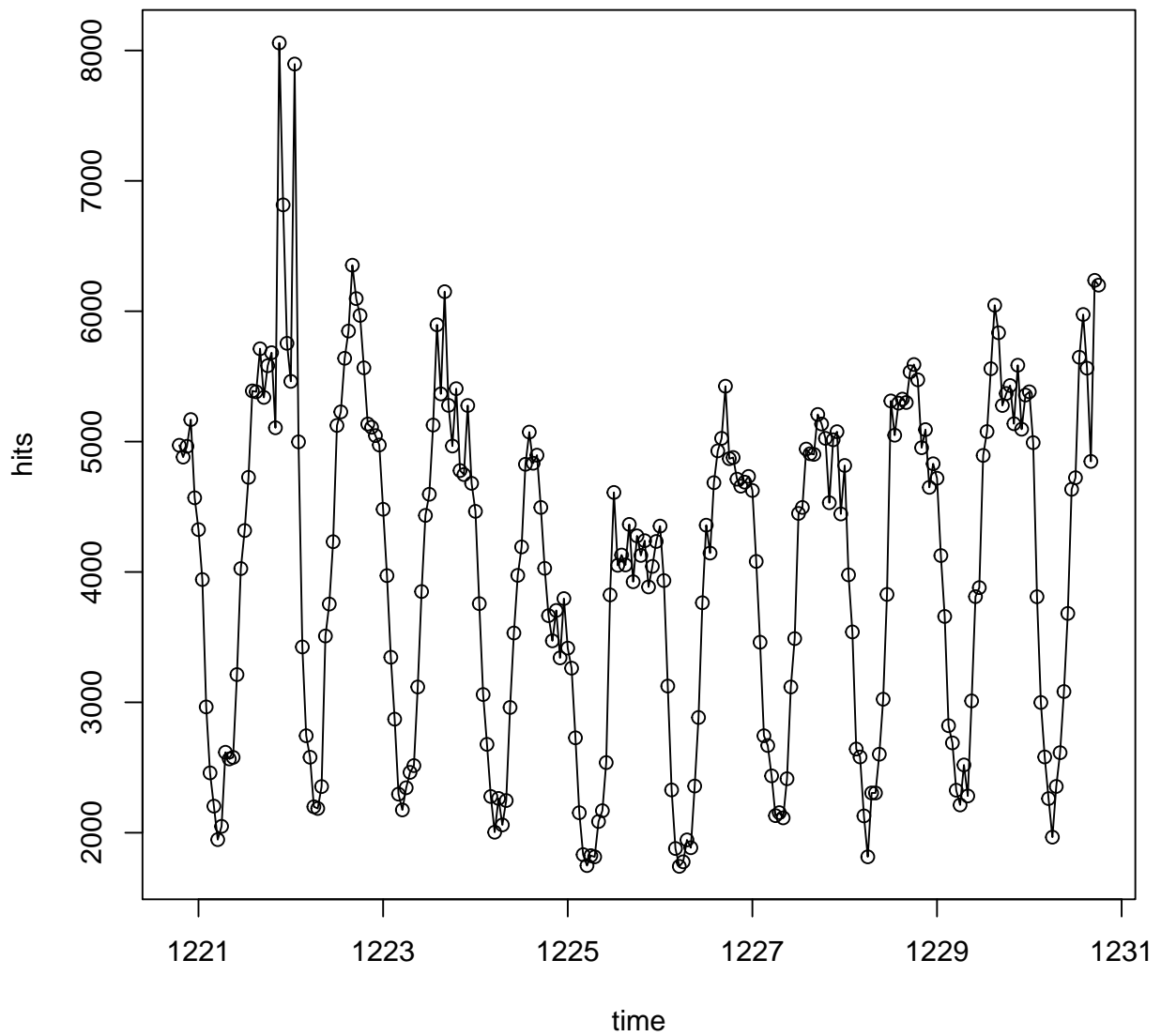
setwd("C:/Users/Esther/Desktop/stat243-fall-2017-master/section/s07")
dat <- read.table("disney_data", sep = ",")

names(dat) <- c("date", "time", "lang", "hits")
dat$date <- as.character(dat$date)
dat$time <- as.character(dat$time)
dat$time[dat$time %in% c("0", "1")] <- "000000"
wh <- which(nchar(dat$time) == 5)
dat$time[wh] <- paste0("0", dat$time[wh])
dat$chron <- chron(dat$date, dat$time, format = c(dates = "ymd", times = "hms"))
dat$chron <- dat$chron - 5/24 # GMT -> EST

dat <- dat %>% filter(dat$lang == "en")

## Warning: package 'bindrcpp' was built under R version 3.3.3

# collect data between Chismas period in 2008. Bedtime Stories release in
# December 25, 2008.
sub <- dat %>% filter(dat$date < 20081231 & dat$date > 20081220)
# pdf('disney-traffic.pdf', width = 5, height = 5) par(mfrow = c(2, 1), mgp
# = c(1.8, 0.7, 0), mai = c(0.6, 0.6, 0.1, 0.1)) plot(dat$chron, dat$hits,
# type = 'l', xlab = 'time', ylab = 'hits')
plot(sub$chron, sub$hits, type = "l", xlab = "time", ylab = "hits")
points(sub$chron, sub$hits)
```



```
# dev.off()
```

4 Q4

4.1 (a)

```
library(foreach)
require(parallel)
require(doParallel)
nCores <- as.integer(Sys.getenv("SLURM_CPUS_ON_NODE"))
registerDoParallel(nCores)
library(stringr)
library(readr)
```

```
#Note: The single function and usage of str_pad are refer from my classmate, Ming Qiu.
single <- function(filename){
  data <- readLines(filename)
  library(stringr)
  out <- data[str_detect(data,'Barack_Obama')]
  return(out) }

n <- 960
#nsub <- 10
result <- foreach(
  i = 1:n,
  .packages = c("stringr"),
  .combine = c,
  .verbose = TRUE) %dopar% {
  filename <- paste("/global/scratch/paciorek/wikistats_full/dated_for_R/part-", str_pad(i-1, width=5,
    #filename <- paste("C:/Users/Esther/Desktop/stat243-fall-2017-master/part-",str_pad(i-1, width=5,
    outputs <- single(filename)
    #outputs
  }

results <- data.frame (matrix(sapply(list(result),function(x) unlist(strsplit(x,split=" ") ) ), ncol=6,
#results <- data.frame ( t(sapply(result,function(x) unlist(strsplit(x,split=" ") ) ) )
colnames(results) <- c("date", "time", "language", "webpage", "number of hits", "page size")
head(results)
class(results)
print(dim(results))
proc.time()
```

Part of output file is following:

```
dic <- "C:/Users/Esther/Desktop/stat243-fall-2017-master/section/s07/ps6q4no.out"
output <- readLines(dic)
tail(output, 25)

## [1] "      date   time  language                               webpage"
## [2] "1 20081231 100000 commons.m                               File:Barack_Obama_signature.svg"
## [3] "2 20081202 150000      en Image:Michelle,_Oprah_Winfrey_and_Barack_Obama.jpg"
## [4] "3 20081229 200001      en      Europe_and_Ireland_not_MYOB_or_Barack_Obama"
## [5] "4 20081123 100000      it                               Discussion:Barack_Obama"
## [6] "5 20081101 010000      pt                               Imagem:Barack_Obama.jpg"
## [7] "6 20081231 120000      en      Washington_Or_Barack_Obama"
## [8] "      number of hits page size"
## [9] "1              4    26796"
## [10] "2              2    19158"
## [11] "3              1     6047"
## [12] "4              1    40260"
## [13] "5              3    24840"
## [14] "6              1     6007"
## [15] "> class(results)"
## [16] "[1] \"data.frame\""
## [17] "> print(dim(results))"
## [18] "[1] 325263      6"
## [19] "> proc.time()"
## [20] "      user      system    elapsed "
```

```
## [21] "34664.559 2247.375 2715.079 "
```

```
## [22] "> "
```

```
## [23] "> proc.time()"
```

```
## [24] "      user      system    elapsed "
```

```
## [25] "34664.560 2247.375 2715.093 "
```

4.2 (b)

As the time is 2715 seconds(46 minutes) for one node, which vert close to 12 minutes if divide it my 4. Hence, using R for each has almost similar times or even faster.

4.3 (c)

```
#statistic allicate
library(foreach)
require(parallel)
require(doParallel)
nCores <- as.integer(Sys.getenv("SLURM_CPUS_ON_NODE"))
registerDoParallel(nCores)
library(stringr)

#Note: The single function and usuage of str_pad are refer from my classmate, Ming Qiu.
single <- function(filename){
  data <- readLines(filename)
  library(stringr)
  out <- data[str_detect(data,'Barack_Obama')]
  return(out) }

n <- 960
#n <- 2
path <- "/global/scratch/paciorek/wikistats_full/dated_for_R/part-"
#path <- "C:/Users/Esther/Desktop/stat243-fall-2017-master/part-"
i=1:n
filename <- mclapply(n, mc.preschedule=TRUE, function(x) paste(path,str_pad(i-1, width=5, side="left",
output <- mclapply(filename[[1]], single, mc.preschedule=TRUE)
results <- data.frame( matrix(sapply( unlist( output ), function(x) unlist(strsplit(x,split=" ") )), nc

colnames(results) <- c("date", "time", "language", "webpage", "number of hits", "page size")
head(results)
class(results)
print(dim(results))
proc.time()
write.table(results, file='/global/scratch/esther730/results_ob.csv',sep = ",",row.names = FALSE)
```

The following output reveal the processing time of static allocate, which is 17198 that nearly 289 minutes, when running at 4 cores, it will cost nearly 71 minutes.

```
dic <- "C:/Users/Esther/Desktop/stat243-fall-2017-master/section/s07/ps6q4st.out"
output <- readLines(dic)
tail(output, 25)

## [1] "1
## [2] "2 Special:AllPages/I_ran_Project_Vote_voter_registration_drive_in_Illinois,_ACORN_was_smack_da
## [3] "3
```



```
## [4] "4"
## [5] "5"
## [6] "6"
## [7] " number of hits page size"
## [8] "1      86    2032215"
## [9] "2       2    25520"
## [10] "3       1     7825"
## [11] "4      16   760462"
## [12] "5       4    55875"
## [13] "6       1    20922"
## [14] "> class(results)"
## [15] "[1] \"data.frame\""
## [16] "> print(dim(results))"
## [17] "[1] 433895      6"
## [18] "> proc.time()"
## [19] "   user   system elapsed "
## [20] "16643.04  343.28 17197.31 "
## [21] "> write.table(results, file='/global/scratch/esther730/results_ob.csv',sep = '\\','\\',row.names = "
## [22] "> "
## [23] "> proc.time()"
## [24] "   user   system elapsed "
## [25] "16643.653  343.314 17197.955 "
```

```
#dynamic allocate
library(foreach)
require(parallel)
require(doParallel)
nCores <- as.integer(Sys.getenv("SLURM_CPUS_ON_NODE"))
registerDoParallel(nCores)
library(stringr)

single <- function(filename){
  data <- readLines(filename)
  library(stringr)
  out <- data[str_detect(data,'Barack_Obama')]
  return(out) }

n <- 960
#nsub <- 10
result <- foreach(
  i = 1:n,
  .packages = c("stringr"),
  .combine = c,
  .verbose = TRUE) %dopar% {
  filename <- paste("/global/scratch/paciorek/wikistats_full/dated_for_R/part-", str_pad(i-1, width=5)
  outputs <- mclapply(filename,single, mc.preschedule=FALSE)
  #outputs
}
results <- data.frame (matrix(sapply(list(result),function(x) unlist(strsplit(x,split=" ") ) ), ncol=6,
#results <- data.frame ( t(sapply(result,function(x) unlist(strsplit(x,split=" ") ) ) )
colnames(results) <- c("date", "time", "language", "webpage", "number of hits", "page size")
head(results)
class(results)
print(dim(results))
```

```
proc.time()
```

The dynamic allocate used 22585 seconds(377minutes) to grep the strings among 960 files. When running on four nodes, it will cost about 94 minutes to obtain the results.

```
dic <- "C:/Users/Esther/Desktop/stat243-fall-2017-master/section/s07/ps6q4dy.out"
output <- readLines(dic)
tail(output, 25)

## [1] "1
## [2] "2 Special:AllPages/I_ran_Project_Vote_voter_registration_drive_in_Illinois,_ACORN_was_smack_da
## [3] "3
## [4] "4
## [5] "5
## [6] "6
## [7] " number of hits page size"
## [8] "1          86    2032215"
## [9] "2           2     25520"
## [10] "3           1      7825"
## [11] "4          16    760462"
## [12] "5           4     55875"
## [13] "6           1     20922"
## [14] "> class(results)"
## [15] "[1] \"data.frame\""
## [16] "> print(dim(results))"
## [17] "[1] 433895      6"
## [18] "> proc.time()"
## [19] " user system elapsed "
## [20] "43138.56 1849.43 22584.29 "
## [21] "> #write.table(results, file='/global/scratch/esther730/results_ob.csv',sep = '\\',\\",row.names
## [22] "> "
## [23] "> proc.time()"
## [24] " user system elapsed "
## [25] "43138.56 1849.43 22584.30 "
```

Dynamic allocation is better when iterations may take very different amounts of time. However, in this case, the data size for each file is quite similar, so it would be better to use static allocation which evenly divides loop iteration space into n chunks. Hence, generally, static allocation is a better method than dynamic way in this case.

5 Q5

5.1 (a)

(calculation attached as hand written version)

The cholesky will do $\frac{n^3}{6} + \frac{n^2}{2} - \frac{2n}{3}$ times operations.

5.2 (b)

Yes, we can overwrite the original matrix.

By the formula of cholesky decomposition, first, the a_{11} , which means the first element in the matrix, would only be used after the square root of it. Hence, it's fine to overwrite it.

Second, the first row will be calculated by dividing each element by u_{11} , which is the first element of the cholesky matrix we obtained by the square root of a_{11} . Hence, according to the step 3 of the formula that begins to calculate the diagonal by row, and we found that only the elements after calculation will be used to obtain u_{ii} and u_{ij} (where

i is smaller than j). Therefore, it's safe to overwrite the first row.

From the fomula, we can found that when calcuate u_{ii} and u_{ij} , what we need is the corresponding elements from original matrix and the transfer elements from the former rows. Hence, we won't used any corresponding elements of the former rows from the original matrix. Hence, we can overwritten the original matrix by row to each elements.