# Stat 243 PS 6


## Riv Jenkins


### 1

The goal of the simulation study presented in the article was to see the performance of a null hypothesis test in a controlled setting. They do this by comparing their hypothesis test with a non-adjusted hypothesis test using the same data. They measure the effectiveness of their adjusted hypothesis test by looking at the difference in significance levels and power of the two tests in a variety of settings. The authors chose to test certain mixing parameters to generate the samples from and test their hypotheses. They also do not mention how these samples were generated which probably would be worth mentioning for the sake of thoroughness. The tables seem to make sense and do a decent job of showing how the tests are more powerful as more samples are drawn or as the distributions being mixed get farther apart (and thus easier to distinguish). The authors probably used 1000 replications based off of an attempt to balance rigor with computational efficiency. 10 replications would probably not be enough. We could determine the appropriate number of replications by looking at the distribution of the results. One would expect them to approach a normal distribution with enough replications.

### 2

```
library(RSQLite)

#set up SQL connection
drv <- dbDriver("SQLite")
dir = '../../'
dbFilename = 'stackoverflow-2016.db'
db = dbConnect(drv, dbname = file.path(dir, dbFilename))

#create a view that has all distinct python users
#dbSendQuery(db, "drop view Pyusers")
dbSendQuery(db, "create view Pyusers as select distinct userid from users
            U join questions Q on U.userid = Q.ownerid
            join questions_tags T on Q.questionid = T.questionid
            where tag = 'python' ")
```

```
## <SQLiteResult>
##   SQL  create view Pyusers as select distinct userid from users
##             U join questions Q on U.userid = Q.ownerid
##             join questions_tags T on Q.questionid = T.questionid
##             where tag = 'python'
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
#select only the R users that are not also python users
Rusers = dbGetQuery(db, "select distinct userid from users U join questions Q on
                    U.userid = Q.ownerid join questions_tags T on Q.questionid =
                    T.questionid where tag = 'r' and userid not in Pyusers")
```

```
## Warning: Closing open result set, pending rows
```

```
dbDisconnect(db)
head(Rusers)
```

```
##    userid
## 1  575952
## 2 5738949
## 3 4802680
## 4 3507767
## 5 2670641
## 6 4148256
```

## 3

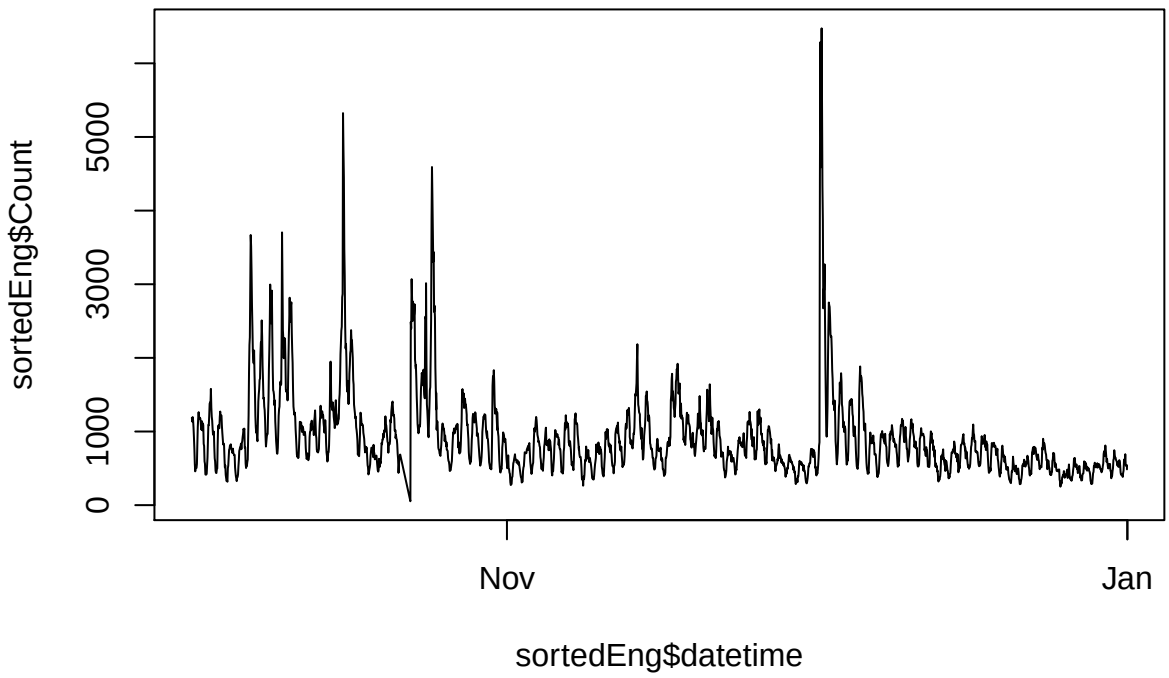
For my analysis I decided to look at the hits for pages with "Recession" in the title since the end of 2008 was around the time we were officially entering the recession.

```
dir = '/global/scratch/paciorek/wikistats_full'

### read data and do some checks ###

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

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

# note delayed evaluation

lines.count()  # 9467817626 for full dataset

# watch the UI and watch wwall as computation progresses

testLines = lines.take(10)

testLines[0]

testLines[9]

import re
from operator import add
def find(line, regex = "[Rr]ecession", 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)

recess = lines.filter(find).repartition(480)

recess.count()
```

```
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]))

counts = recess.map(stratify).reduceByKey(add)

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
outputDir = 'rec'
out = counts.map(transform).repartition(1)
out.saveAsTextFile(outputDir)
```

After running the map reduce steps to scrape information from the raw data, I transferred it to my laptop and did some basic analysis in R. I looked at the hit count for 'recession' on english wikipedia pages over time. The results are plotted below.

```
#read in the data
data = read.csv('part-00000.csv', header = FALSE)
#give the columns appropriate names
names(data) = c('Date', 'Time', 'Language', 'Count')
#filter out english results
engl = data[(data$Language == 'en'),]

#convert the date and time to format R can understand
datetime = str_c(as.character(engl$Date), ' ', as.character(engl$Time/10000))
datetime = str_replace(datetime,"(2008)(1[0-9])([0-9]{2}) ([0-9]{1,2})", "\\2/\\3/\\1 \\4:00:00")
engl$datetime = as.POSIXct(datetime, format = '%m/%d/%Y %H:%M:%S')

#sort by date/time and plot graph of page hits over time
sortedEng = engl[order(engl$datetime),]
plot(sortedEng$datetime, sortedEng$Count, type = 'l')
```




As one might expect, there are clear daily trends and some spikes probably corresponding to news events related to the recession. I was curious about the giant spike in December. I found that this spike corresponded to December 1, 2008. After googling that date I found this news article: http://money.cnn.com/2008/12/01/markets/markets_newyork/index.htm. That day the DOW fell 680 points or 7.7%. The National Bureau of Economic Research also officially confirmed that the US was in a recession.

## 4

The parallel R code took about 5,255 seconds to run which is about 87.6 minutes on 24 cores. With perfect scalability this would mean the code would take around 22 minutes on 96 cores. This is not horribly worse than the code with pyspark.

```
library(foreach)
library(doParallel)
library(stringr)
library(readr)

#setup parallel
parread <- function(){
  nCores <- 24
  registerDoParallel(nCores)

  #find files to read in
  files = list.files(path = "/global/scratch/paciorek/wikistats_full/dated_for_R/", pattern = "part-",
                     full.names=TRUE)
```

```
  result <- foreach(i = files) %dopar% {
    #read in file to dataframe
    df = read_delim(i, delim = " ", quote = "")
    #get only the entries with Barrack
    output = df[grep("Barrack_Obama", df$V4),] # this will become part of the out object

  }
  return(result)
}
system.time(result <- parread())
#combine the results into single dataframe
df = do.call("rbind", result)
```

## 5 a)

To compute $U_{ii}$ there are $i-1$ operations. To compute $U_{ij}$ for $j = i+1, ..., n$ there are $i-1+1$ operations, but these occur $n-i$ times. This means that the total number of operations per row $i$ is $i-1+i(n-i)$. Thus, the total number of operations for the Cholesky decomposition is $\sum_{i=1}^{n} -i^2 + i(n+1) - 1$.

This can be rewritten as $-\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)(n+1)}{2} - n$.

This simplifies to $\frac{n^3}{6} + \frac{n^2}{2} - \frac{2n}{3}$ operations.

## b)

Since the cholesky decomposition only depends on other values from the decomposed matrix (and the value that is to be changed), storing the new matrix in the old one could save some memory, and it shouldn't create any problems in the decomposition algorithm. The only problem might be that you now don't have your original matrix if you need it for any other computations down the road.