STA242 HW5 Report

SSH URL: git@bitbucket.org:sxli2015/sta242-assignment-5.git
Shuxin Li 912525987

1. Exploration of the data

The taxi data has 12 trip_data_.csv files and 12 trip_fare_.csv files.

Firstly I check if the number of rows of each pairs are the same. The results shows that all the pairs are the same.

Secondly I check if all the observations are matched in each pairs. Here I randomly select 10000 observations from each pairs and test if all these observations are matched by medallion and hack license. I also use parallel computation to test all 12 pairs simultaneously. From the results, it indicates that all the randomly selected observation matched.

From the test of data above, we can consider the data is basically correct.

2. The first approach to analyze the data

The first approach is using R and shell to analyze the data.

The first task is to compute the deciles of the total amount less the tolls. Firstly, I found the total amount is in the eleventh column and the tolls is in the tenth column, then I use the shell command to read the tenth and eleventh columns of each file, compute the difference of total amount and tolls and save them as rda file. To save the memory, I change the large vector to a frequency table to compute the deciles.

The deciles are below (Table 2.1).

Percentile	10%	20%	30%	40%	50%	60%	70%	80%	90%
Quantile	6	7.5	8.5	9.77	11	13	15	18.5	26.25

Table 2.1

The second task is to fit a linear regression predicting total amount less the tolls using trip time as the predictor. As before, I used shell to read trip time data and save them as rda file. Then I use these rda files to estimate the beta0 and beta1 by simple linear regression formula.

The computation time which the first approach takes is huge. It takes about 17280 seconds to finish the two tasks, mainly on the input of all data. While the programming time takes a little. It is very simple idea and easily understanding.

The estimate of beta0 is 14.52334 and the estimate of beta1 is 2.06023e-05

The fitted equation is $\hat{y} = 14.52334 + (2.06023e - 05)x_1$

3. The second approach to analyze the data

The second approach is using parallel computation in R and shell to analyze the data.

The second approach is an improvement to the first approach by parallel computation. In the second approach, I read the tenth and eleventh columns of different files simultaneously, compute the difference of total amount and tolls and save them as rda file. To save the memory, I also change the large vector to a frequency table to compute the deciles. The deciles are below (Table 3.1).

Percentile	10%	20%	30%	40%	50%	60%	70%	80%	90%
Quantile	6	7.5	8.5	9.77	11	13	15	18.5	26.25

Table 3.1

In the second task. I used shell to read trip time data of different files simultaneously and save them as rda files as well. Then using these rda files to estimate the beta0 and beta1 by simple linear regression formula.

The computation time of the second approach is less than the first approach without using parallel computation. Considering the situation of my computer, I set 3 clusters each time, which means that it can deal three files simultaneously. It takes about 8702 seconds to finish the two tasks, mainly on the input of all data as well. The programming time takes more than the first approach.

The estimate of beta0 is 14.52334 and the estimate of beta1 is 2.06023e-05.

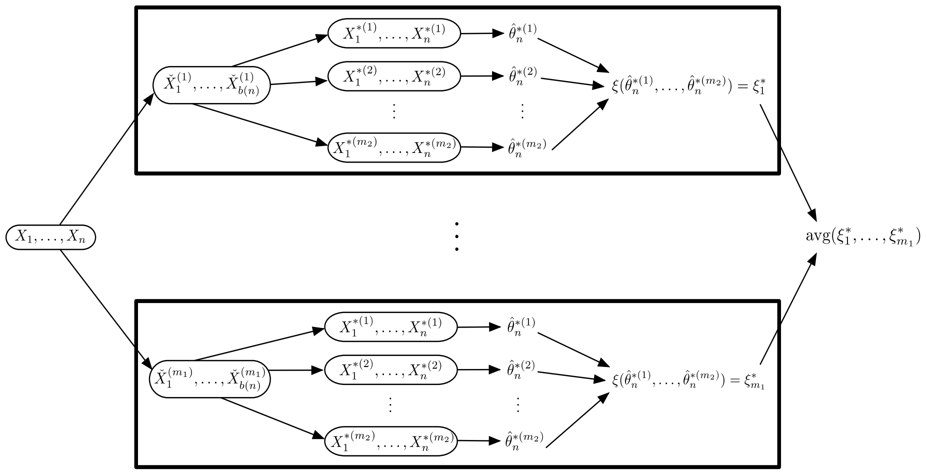
The fitted equation is $\hat{y} = 14.52334 + (2.06023e - 05)x_1$

4. Compute standard errors of the estimates

Here, I use the Bag of Little Bootstraps (Plot 4.1) to compute the variance of response variable Y and compute standard errors of the estimates beta0 and beta1 by formula below.

$$var\left(\hat{\beta}_1\right) = \frac{\sigma^2}{s_X^2},\ var\left(\hat{\beta}_0\right) = \sigma^2\left(\frac{1}{n} + \frac{\overline{x}^2}{s_x^2}\right)$$

Suppose the population has about N = 170 million observations. Do the two sampling steps. First is to sample s =10 subsets of size m = $N^{0.65}$ from the population. Second is to resample N observations from each subsets r = 10 times.



$X_1, \ldots, X_n$
$\check{X}_1^{(1)}, \ldots, \check{X}_{b(n)}^{(1)}$
$X_1^{*(1)}, \ldots, X_n^{*(1)}$
$X_1^{*(2)}, \ldots, X_n^{*(2)}$
$X_1^{*(m_2)}, \ldots, X_n^{*(m_2)}$
$\hat{\theta}_n^{*(1)}$
$\hat{\theta}_n^{*(2)}$
$\hat{\theta}_n^{*(m_2)}$
$\xi(\hat{\theta}_n^{*(1)}, \ldots, \hat{\theta}_n^{*(m_2)}) = \xi_1^*$
$\check{X}_1^{(m_1)}, \ldots, \check{X}_{b(n)}^{(m_1)}$
$\xi(\hat{\theta}_n^{*(1)}, \ldots, \hat{\theta}_n^{*(m_2)}) = \xi_{m_1}^*$
$\text{avg}(\xi_1^*, \ldots, \xi_{m_1}^*)$

Plot 4.1

There is a problem I encounter which is that the computer can’t allocate enough memory to a 170 million sampling observations. It is impossible to random select 170 million numbers directly. Thus, I random select 170 million numbers in several times and combined them as frequency table, then compute the standard errors

The results show that the standard error of beta0 is 0.000882736 and the standard error of beta1 is 5.574154e-08.

5. Comparison of different approaches

The first approach is using R and shell without parallel computation. It takes 17280 seconds to finish the two task. The deciles can be seen in Table 2.1. The estimate beta0 is 14.52334 and the estimate beta1 is 2.06023e-05.

The second approach is using R and shell with parallel computation. It takes 8702 seconds to finish the two tasks. The deciles can be seen in Table 3.1. The estimate beta0 is 14.52334 and the estimate beta1 is 2.06023e-05 as well.

The two approaches’ results are the same. Because, two approaches both save the same rda files of trip time and difference of total amount less tolls.

Though the results are the same, the second approach takes less time than the first approach. This shows that parallel processing work well.

However there are three main drawbacks for parallel processing I encountered.

The first is that parallel processing easily goes wrong. For example, I need to select the number of connection carefully.

The second is that when the parallel processing goes wrong, the computer is easily crashed. It takes time to restart the computer.

The third is that if one node goes wrong, then all other nodes will be stopped. For example, I want to extract a column of each files by applying parallel processing and save them, but if the last file has something wrong, the program will stop and get nothing. But in the first approach, if the last file goes wrong, the program save all files except the last one.

In addition, it is hard to debug the parallel processing program. Therefore, it takes more programming time than non-parallel approach.

In conclusion, these two approaches both have advantages and disadvantages. For the non-parallel approach, the advantage is that the programming is simple and easily to debug which save programming time. While the disadvantage is to take more computation time. For parallel approach, the advantage is to save the computation time but takes more programming time, especially for some data which is hard to deal with. Therefore, choosing approach should not only consider the computation time but also consider the programming time. In my opinion, for small dataset, it may be not necessary to apply the parallel computation.

6. Estimate multiple regression estimators

From each file, I use “pipe()” to randomly sample one observation from each block which has 1000 rows. I sample 10000 times, then I get 10000 observations for each file. Therefore, I have 120000 observations totally. I use these observations to fit a multiple regression. The fitted equation is below.

$$\hat{y} = 2.4019 + 0.0152x_1 + 3.5821x_2$$

Appendix

```
################################################
### Check if all pairs of files are matched ###
################################################

unique(unlist(compareParallel(12,4)))

compare = function(fileindex, block = 1000, iter = 10000){
  # fileindex is the index of file, block is the number of data read
  ## each time, iter is the number of blocks read
  # the function is to return a number 0 or 1, which represent no error
  ## or error exists
  # the function is to check if trip_fare_.csv file and trip_data_.csv
  ## file matched

  pattern1 = paste("cut -f 1,2 -d, trip_fare_", fileindex,
                                                        ".csv",  sep="")
  con20 = pipe(pattern1, "r")
  pattern2 = paste("cut -f 1,2 -d, trip_data_", fileindex,
                                                         ".csv", sep="")
  con21 = pipe(pattern2, "r")
  # initial the error
  error = 0
  for(i in 1:iter){
    # random sample an index from each block
    j = sample(block,1)
    a = readLines(con20, block)[j]
    b = readLines(con21, block)[j]
    if(a!=b){
      print(paste("file:", fileindex, ",", i*block+j))
      error = 1
      break
    }
  }
  return(error)
}


compareParallel = function(filenum, con){
  # filenum is the files’ total number, con is the number of cluster
  ## created
  # the function is to use parallel processing to check if all pairs of
  ## files are matched
  cl = makeCluster(con)
  clusterExport(cl, "compare")
  output = vector("list", filenum/con)
  for(i in 1:(filenum/con)){
    index1 = 1 + con*(i-1)
    index2 = con*i
    output[[i]] = parLapply(cl, c(index1:index2), compare)
  }
  stopCluster(cl)
  output
}


##############################
Version 1: R and shell
##############################

### compute the quantiles ###
# get a big frequency table of Y including all files
ytable_all = bigtableY(12)
cumfreq = cumsum(ytable_all$Freq)
size = sum(ytable_all$Freq)
quantiles = seq(0.1, 0.9, by = 0.1)
# compute the quantiles
sapply(quantiles, function(i) getQuantile(i, ytable_all,
                                                  size, cumfreq))

### fit a regression and estimate beta0 and beta1 ###
# save all files' triptime data
sapply(1:12, function(i) getTime(i))
# estimate beta0 and beta1
parameters = predictbeta(12)

##################### Version 1 functions #################
### Compute the deciles (version 1: shell and R version) ###

getDiff = function(fileindex){
  # fileindex is a trip_fare_.csv file's index
  # the function returns a list which contains a vector of difference of
  ## total_amount less total_tolls and a frequency table of the
  ### difference
  # the function is to save the frequency table and the vector of
  ## total_amount less total_tolls
```

```r
  # use shell command to read total_amount and total_tolls
  pattern11 = paste("cut -f 10,11 -d, trip_fare_", fileindex,
                                                 ".csv", sep="")
  y = system(pattern11, intern = TRUE)[-1]
  # split characters and transform character to numeric
  y = strsplit(y, ",")
  y = lapply(y, as.numeric)
  diff = unlist(lapply(y, function(y) y[2]-y[1]))
  # get the frequency table
  fretable = data.frame(table(diff))
  Ys = list(table = fretable, diff = diff)
  save(Ys, file = paste("diff_", fileindex, ".rda", sep=""))
  Ys
}


bigtableY = function(j){
  # j is an integer which represents the number of files
  # the function returns a big frequency table of Y including all files
  # initial a frequency table
  tabley = data.frame()
  for(i in 1:j){
    tabley = rbind(tabley, getDiff(i)$table)
  }
  # sort the Y
  tabley = tabley[order(tabley$diff),]
  # transform Y to integer
  tabley$diff = as.character(tabley$diff)
  tabley$diff = as.numeric(tabley$diff)
  tabley
}


getQuantile = function(singleQ, dftable, size, cumfreq){
  # singleQ is a percent, dftable is a frequency table, size is the
  ## total number of the frequency table, cunfreq is cumulated sum of
  ### the frequency
  # the function returns a quantile

  if(size*singleQ == ceiling(size*singleQ)){
    pos1 = size*singleQ
    pos2 = size*singleQ + 1
    Q1 = dftable[which(cumfreq>=pos1)[1],]$diff
    Q2 = dftable[which(cumfreq>=pos2)[1],]$diff
    Q = (Q1+Q2)/2
  }else{
    pos = ceiling(size*singleQ)
    Q = dftable[which(cumfreq>=pos)[1],]$diff
  }
  return(Q)
}


### fit the linear regression ###
getTime = function(fileindex){
  # fileindex is a trip_data_.csv file’s index
  # the function returns a list which contains a vector of triptime and
  ## a frequency table of the triptime
  # the function is to save the frequency table and vector of triptime

  # use shell command to read triptime
  pattern9 = paste("cut -f 9 -d, trip_data_", fileindex,
                                                    ".csv", sep="")
  triptime = system(pattern9, intern = TRUE)[-1]
  triptime = as.numeric(triptime)
  # save the column as vector and frequency table
  fretable = data.frame(table(triptime))
  Xs = list(table = fretable, time = triptime)
  save(Xs, file = paste("time_", fileindex, ".rda", sep=""))
}


predictbeta = function(filenum){
  # filenum is the files’ total number to estimate the beta
  # the function is to estimate the beta0 and beta1 of simple regression

  # initial the parameter
  df = data.frame()
  sumx = 0
  sumy = 0
  sumxy = 0
  sumx2 = 0
  n= 0
  # calculate parameter of each file and sum them
  for(i in 1:filenum){
    pattern1 = paste("diff_", i, ".rda", sep="")
    load(pattern1)
    pattern2 = paste("time_", i, ".rda", sep="")
    load(pattern2)
    diff = Ys$diff
    triptime = Xs$time
    xy = diff*triptime
    sumxy = sumxy + sum(xy)
    sumx = sumx + sum(triptime)
    sumy = sumy + sum(diff)
    sumx2 = sumx2 + sum(triptime^2)
    n = n + length(diff)
  }
  meanx = sumx/n
  meany = sumy/n
  Sxx = sumx2 - n*(meanx^2)
  beta1.hat = (sumxy - n*meanx*meany)/Sxx
  beta0.hat = meany - beta1.hat * meanx
  list(beta0.hat = beta0.hat, beta1.hat = beta1.hat,
                            Sxx = Sxx, meanx = meanx, n = n)
}


###########################################
Version 2: R and shell with Parallel
###########################################

# compute the quantiles
# get a big frequency table of Y including all files by parallel
ytable_all = getDifftableParallel(filenum = 12, con = 4)
cumfreq = cumsum(ytable_all$Freq)
size = sum(ytable_all$Freq)
quantiles = seq(0.1, 0.9, by = 0.1)
sapply(quantiles, function(i) getQuantile(i, ytable_all,
                                                     size, cumfreq))

### fit a regression and estimate beta0 and beta1 ###
# save all files' triptime data by parallel processing
getTimeParallel(12, 4)
# estimate beta0 and beta1
parameters = predictbeta(12)


##################### Version 2 functions ##########################
# Compute the deciles (version 2: shell and R version using parallel)
library(snow)
```

```r
getDiff_table = function(i) getDiff(i)$table

getDifftableParallel = function(filenum, con){
  # filenum is the files’ total number, con is the number of cluster
  ## created
  # the function is to use parallel computation to get all response
  ## variable and change them to a big frequency table

  # initial the first table
  tabley = data.frame()
  cl = makeCluster(con)
  clusterExport(cl, "getDiff_table")
  for(i in 1:(filenum/con)){
    index1 = 1 + con*(i-1)
    index2 = con*i
    result = parLapply(cl, c(index1:index2), getDiff_table)
    # combine the frequency table of response variable
    tabley = rbind(tabley, do.call(rbind, result))
  }
  stopCluster(cl)
  tabley
}

### fit the linear regression ###
getTimeParallel = function(filenum, con){
  # filenum is the files’ total number, con is the number of cluster
  ## created
  # the function is to use parallel computation to get all triptime data

  cl = makeCluster(con)
  clusterExport(cl, "getTime")
  for(i in 1:(filenum/con)){
    index1 = 1 + con*(i-1)
    index2 = con*i
    parLapply(cl, c(index1:index2), getTime)
  }
  stopCluster(cl)
}

# Other functions are the same as version 1
```

```
###########################################
### estimate the standard errors by BLB ###
###########################################

# use BLB to estimate the standard error of estimates
parameters = predictbeta(12)
n = parameters$n
meanx = parameters$meanx
Sxx = parameters$Sxx

Ysigma2.hat = BLBsigma(ytable_all, bag = 10, times = 10)
var_beta0.hat = sqrt(Ysigma2.hat*(1/n + meanx^2/Sxx))
var_beta1.hat = sqrt(Ysigma2.hat/Sxx)

BLBsigma = function(population, bag, times){
  # population is represented by a table, bag is the number of subsets
  ## of size m = N^0.65 to sample, times is the times to resample N
   ### observations from each subsets
   # the function is to estimate the sigma square by BLB

  n0 = sum(population$Freq)
  vsigma = integer(bag)
  for(j in 1:bag){
    # From Y1,...,Yn to sample Y1,...,Yb
    ymtable = table(sample(population$diff, n0^(0.6), replace = TRUE,
                                            prob = population$Freq/n0))
    ymtable = df_table(ymtable)
    n1 = sum(ymtable$Freq)
    sigma2 = 0
    # From Y1,...,Yb to resample Y1*,…,Yn* and estimate sigma square
    for(i in 1:times) {
      sigma2 = sigma2 + getSigma2(ymtable, limit = 10, n0)
    }
    vsigma[j] = sigma2/times
  }
  mean(vsigma)
}


getSigma2 = function(ymtable, limit, n0){
  # ymtable is a frequency table of y. limit is to control the sampling
  ## size each time, n0 is total number of population N
  # the function is to resample Y1*,...,Yn* from Y1,...,Yb and estimate
  ## the sigma square each time
  n1 = sum(ymtable$Freq)
  temp = data.frame()
  for(i in 1:limit){
    yntable = table(sample(ymtable$diff, n0/limit, replace = TRUE,
                                                  prob = ymtable$Freq/n1))
    yntable = df_table(yntable)
    yntable = rbind(temp, yntable)
    temp = yntable
  }

  sumy2 = sum(yntable$diff^2*yntable$Freq)
  meany = sum(yntable$diff*yntable$Freq)/n0
  sigma2 = (sumy2 - n0*meany^2)/(n0-1)
  sigma2
}


df_table = function(table){
  # the function is to transform a frequency table to a dataframe
  table = data.frame(table)
  names(table) = c("diff", "Freq")
  table$diff = as.character(table$diff)
  table$diff = as.numeric(table$diff)
  table
}


#######################################################
### Task 3: estimate multiple regression estimators ###
#######################################################

filenum = 12
con = 4
data = getDataParallel(filenum, con)
df =data.frame()
for(i in 1:(filenum/con)){
  df = rbind(df, do.call(rbind, data[[i]]))
}

model2 = lm(diff~surcharge + triptime, data = df)
summary(model2)
```

```
getData = function(fileindex, block = 1000, iter = 10000){
  # fileindex is the index of file, block is the number of data read
  ## each time, iter is the number of blocks read
  # the function is to randomly sample observation and return its
  ## x1,x2,y

  pattern1 = paste("cut -f 7,10,11 -d, trip_fare_", fileindex,
                                                    ".csv", sep="")
  con20 = pipe(pattern1, "r")
  pattern2 = paste("cut -f 9 -d, trip_data_", fileindex,".csv", sep="")
  con21 = pipe(pattern2, "r")
  v = integer(iter)
  for(i in 1:iter){
    # random sample a index from each block
    j = sample(block,1)
    a = readLines(con20, block)[j]
    b = readLines(con21, block)[j]
    v[i] = paste(a, ",", b, sep ="")
  }
  # transform character to numeric
  y = strsplit(v,",")
  y = lapply(y, as.numeric)
  # output a dataframe with y,x1,x2
  diff = unlist(lapply(y, function(y) y[3]-y[2]))
  surcharge = unlist(lapply(y, function(y) y[1]))
  triptime = unlist(lapply(y, function(y) y[4]))
  df =data.frame(diff,surcharge,triptime)
}

getDataParallel = function(filenum, con){
  # filenum is the files’ total number, con is the number of cluster
  ## created
  # the function is to use parallel processing to do function "getData"

  cl = makeCluster(con)
  clusterExport(cl, "getData")
  output = vector("list", filenum/con)
  for(i in 1:(filenum/con)){
    index1 = 1 + con*(i-1)
    index2 = con*i
    output[[i]] = parLapply(cl, c(index1:index2), getData)
  }
  stopCluster(cl)
  output}
```