STA 242 Assignment 5
Hong Fan 912524085
git@bitbucket.org:rainbowf/assignment-5.git

1. Introduction

In this assignment, we work with the New York taxi data which contain 24 files from 2010 to 2013. Among all the 24 files, 12 are trip_data files and the other 12 are corresponding trip_fare files. Trip data files contain 14 columns in which the 9th column is trip_time_in_secs. Fare data files contain 11 columns in which surcharge, tolls_amount, total_amount are the 7th, 10th and 11th column respectively.

Our tasks are to compute the deciles of the total amount less the tolls, fit a linear regression of total amount less the tolls using trip time as the predictor and then fit a multiple regression after adding surcharge as the second predictor.

Two approaches are used to analyze the taxi data. The first one is parallel computation in R and the second one is random sampling with Bag of Little Bootstraps being used to compute standard errors of the estimates. In both methods, we use shell commands to do file manipulation and extract our target columns.

2. Estimation

2.1 File matching

Before extracting the data, we need to double-check “fare” and “data” files to make sure they match “line by line”. We use 3 common columns--“medallion”, “hack_license” and “pickup_datetime” to do this task (shell commands are in the Appendix). The result shows that they match each other in pairs.

2.2 Method 1

Using package “parallel”, we run computations in parallel by setting cores = 12 (Running computations in 12 different files simultaneously). In order to calculate deciles of total less tolls, we need to combine all their values across 12 different files. By using parallel computing, we calculate frequency table of total less tolls in each file and use tbl.merge function (in the Appendix) to update our table sequentially. We then use wtd.quantile() in “Hmisc” package to calculate their deciles.

Theoretically, the coefficients of the linear model can be computed as

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

Let n denote the total number of observations that we use to do estimations. For simple linear regression model, X is an n by 2 matrix in which the first column is an all-1 vector of length N and the second column contains values of total less tolls from all the observations. For our multilinear regression model, X is an n by 3 matrix in

which the first 2 columns are the same as the X matrix in our first model. Its third column contains “surcharge” values from all the observations. y here is our response variable in the form of n by 1 matrix. It is clear to see from above that betas can be estimated by $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$. The benefit of using these two is that they not only have very low dimensions but also can be updated easily with parallel processing. In our simple linear model, their dimensions are 2 by 2 and 2 by1 respectively and in our multilinear model their dimensions are 3 by 3 and 3 by 1 respectively.

2.3 Method 2

Compared to Method 1, Method 2 also uses parallel computing but applies it to randomly sampled data and use the Bag of Little Bootstrap (BLB) to calculate standard errors of the estimates.

Applying the Bag of Little Bootstrap to our data:
In the data, the total population is N = 173179759.

1) Sample s = 12 subsets of size n $\approx$ N^0.65 from the population without replacement.
2) Resample N = 173179759 observations from each subset r = 13 times with replacement.
3) Calculate standard errors of the estimates in different r groups for every subset.
4) Average standard errors over 12 subsets.

The Bag of Little Bootstrap is used here to avoid the need to fit the “full” data. Within each bootstrapped dataset, there are at most n unique data points. Running the computation in parallel by setting cores = 12, we use command table() to get frequency table of the values for each bootstrapped dataset and then apply wtd.quantile() in “Hmisc” package to get deciles. Follow the BLB algorithm above, we can easily use “apply” functions to calculate standard errors of the deciles (R codes are shown in the Appendix).

2.4 Comparison and Discussion

Unlike the First approach in which the whole population data is used to calculate deciles and betas, approach 2 randomly select a large sample to conduct estimation. Results in Table 1 shows that deciles returned from approach 2 are very close to its true value in the population. As you can see in the table below, estimated values at 40%, 60% and 90% are slightly different, but the other 6 estimations are the same as its true values. Like deciles estimation, the estimated betas are very close to the results from Approach 1. These indicate that randomly sampling is a good approach to conduct estimations when the sample size is large enough.

	10%	20%	30%	40%	50%	60%	70%	80%	90%
1	6.00	7.50	8.50	9.75	11.00	13.00	15.00	18.50	26.12
2	6.000	7.500	8.500	9.774	11.000	12.600	15.000	18.500	26.300

Table 1: Deciles of the total fare less the tolls based on Approach 1 and 2
(Approach 1 is parallel computing in R; Approach 2 is parallel computing and random sampling in R plus standard deviation estimated by BLB)

	Simple Linear Regression	
	beta0	**beta1(trip time)**
Approach 1	1.452334e+01	2.060230e-05
Approach 2	1.450068e+01	3.062508e-05

Table 2: Simple linear regression comparisons

In Approach 2, BLB is used to avoid using the full data and measure the uncertainty. It is clear to see from Table 3, standard errors for the estimates are very small. For the estimated 20% and 30%, their standard errors are 0 or almost 0. The small standard errors indicate our second approach is good to do estimation.

Deciles	**10%**	**20%**	**30%**	**40%**	**50%**
Standard Error	0.008527	0.000000	0.000000	0.011893	0.007665
Deciles	**60%**	**70%**	**80%**	**90%**	
Standard Error	0.021882	0.005189	0.012298	0.040983	
beta		**beta0**		**beta1**	
Standard Error		2.320938*10^(-5)		1.982434*10^(-2)	

Table 3: Standard errors of deciles and betas by BLB (Bag of Little BootStrap)

The whole process of applying Method 1 to our data (including cutting columns in shell and reading data in R) takes 427.112 seconds elapsed time in total.

The whole process of applying Method 2 to our data (including cutting columns in shell and reading data in R) takes 370.187 seconds elapsed time in total. Since elapsed time is highly affected by other activities on the server, I opened two terminals, connecting to the same server and ran the two approaches almost at the same time and got the above result. The difference between elapsed time of running these two approaches is not that long (In approach 2, we only sampled 226476 observations from 173179759). This may result from unrefined R codes and much better algorithm design of sampling, but in general, when population is very large, Approach 2 is a good way to conduct estimation.

Next section shows the estimated coefficients using Approach 2.

2.5 Multilinear Regression Result by Approach 2

	Multilinear Regression		
	beta0	**beta1(trip time)**	**beta2(surcharge)**
Approach 2	1.483120e+01	3.054064e-05	0.032137

Table 4: Use Approach 2 to estimate coefficients of estimates after adding "surcharge"

Shell Command:

```
for i in {1..12}
do
  data = "trip_data_${i}.csv"
  fare = "trip_fare_${i}.csv"
     if diff -w <($data | cut -d , -f 1,2,6) <($fare | cut -d , -f 1,2,4) > /dev/null; then
  echo "$i:Files matched"
  else
  echo "$i:Files do not match"
  fi
  done
```

Approach1

```
library(data.table)
library(parallel)
library(Hmisc)

#set.clusters
cl = makeCluster(12, "FORK")

f = list.files("/home/data/NYCTaxis", pattern = "trip_fare.*\\.csv$", full.names = TRUE)
g = list.files("/home/data/NYCTaxis", pattern = "trip_data.*\\.csv$", full.names = TRUE)

cmd1 = paste("cut -f 7,10,11 -d ,",f) #7--surcharge, 10--tolls amount, 11--tot amount
els1 = clusterSplit(cl,cmd1)

cmd2 = paste("cut -f 9 -d ,",g)
els2 = clusterSplit(cl, cmd2)

clusterExport(cl,"els1")
tt.y = parLapply(cl, els1, function(x) table(fread(x)[, " total_amount", with = F]-fread(x)[, "
                                                    tolls_amount", with = F]))

tbl.merge = function(x, y)
{
   x.name = names(x)
   y.name = names(y)
   cmn = intersect(x.name, y.name)

   tsum = c(x[setdiff(x.name, cmn)], y[setdiff(y.name, cmn)], x[cmn] + y[cmn])
   return(tsum)
```

```
}

totlesstoll = tt.y[[1]]

for (i in 2:12){
   totlesstoll = tbl.merge(totlesstoll, tt.y[[i]])
}

Probs = seq(0.1,0.9,0.1)

wtd.quantile(as.numeric(names(totlesstoll)), weights = totlesstoll, probs = Probs)
```

###Simple Linear Regression

```
Ni = as.numeric(system("wc -l trip_fare* | cut -d ' ' -f4", intern = TRUE)[-13])-1 #extract counts of cases in each trip_fare file #space as "delimiter"
ones = sapply(1:12, function(x) rep(1, Ni[x]))

clusterExport(cl,c("els1","els2","ones"))
mtx.xy = parLapply(cl, 1:12, function(x){mtx = data.matrix(fread(els2[[x]])[,1,with =F]);
                                         mtx.x = cbind(ones[[x]], mtx);
                                         mtx.y = data.matrix(fread(els1[[x]])[,"
                                                 total_amount", with = F] -
                                                 fread(els1[[x]])[, "
                                                 tolls_amount", with = F]);
                                         c(t(mtx.x)%*%mtx.x, t(mtx.x)%*%mtx.y) })

mtx.xy = Reduce("+", mtx.xy)
mtx.1 = matrix(mtx.xy[1:4], nrow = 2) #XtX matrix
mtx.2 = matrix(mtx.xy[5:6], nrow = 2) #xtY matrix

beta = solve(mtx.1)%*%(mtx.2)
```

Approach2

```
install.packages("data.table", lib = "~/Rpackages")
library(data.table)
library(parallel)
library(Hmisc)
Ni = as.numeric(system("wc -l trip_fare* | cut -d ' ' -f4", intern = TRUE)[-13])-1 #extract counts of cases in each trip_fare file #space as "delimiter"
N = sum(Ni) #total population--173179759
n = round(N^0.65) #sample size--226476
ni = round(n*(Ni/N))

cl = makeCluster(12, "FORK")
```

```
Index = parLapply(cl, 1:12, function(i) (sample(1:Ni[i], ni[i], replace = FALSE)))

f = list.files("/home/data/NYCTaxis", pattern = "trip_fare.*\\.csv$", full.names = TRUE)
g = list.files("/home/data/NYCTaxis", pattern = "trip_data.*\\.csv$", full.names = TRUE)

cmd1 = paste("cut -f 10,11 -d ,",f) #10--tolls amount, 11--tot amount
els1 = clusterSplit(cl,cmd1)

cmd2 = paste("cut -f 9 -d ,",g)
els2 = clusterSplit(cl, cmd2)

clusterExport(cl,c("els1","Index"))
tt.y=parLapply(cl,1:12,function(x){
                                    table(fread(els1[[x]])[Index[[x]],“ total_amount",with = F]
                                     -fread(els1[[x]])[Index[[x]], “    tolls_amount", with = F])})

tbl.merge = function(x, y)
{
   x.name = names(x)
   y.name = names(y)
   cmn = intersect(x.name, y.name)

   tsum = c(x[setdiff(x.name, cmn)], y[setdiff(y.name, cmn)], x[cmn] + y[cmn])
   return(tsum)
}

totlesstoll = tt.y[[1]]

for (i in 2:12){
   totlesstoll = tbl.merge(totlesstoll, tt.y[[i]])
}

Probs = seq(0.1,0.9,0.1)

wtd.quantile(as.numeric(names(totlesstoll)), weights = totlesstoll, probs = Probs)

###Simple Linear Regression
ones = sapply(1:12, function(x) rep(1, ni[x]))

clusterExport(cl,c("els1","els2","ones"))
mtx.xy=parLapply(cl,1:12,function(x){
                                mtx= data.matrix(fread(els2[[x]])[Index[[x]],1,with =F]);
                                mtx.x = cbind(ones[[x]], mtx);
```

```
                        mtx.y= data.matrix(fread(els1[[x]])[Index[[x]]," total_amount",
                        with = F] - fread(els1[[x]])[Index[[x]], " tolls_amount", with =
                        F]);
                        c(t(mtx.x)%*%mtx.x, t(mtx.x)%*%mtx.y) })

mtx.xy = Reduce("+", mtx.xy)
mtx.1 = matrix(mtx.xy[1:4], nrow = 2)
mtx.2 = matrix(mtx.xy[5:6], nrow = 2)

beta = solve(mtx.1)%*%(mtx.2)
```

```
###The Bag of Little Bootstraps###############
library(data.table)
library(parallel)
library(Hmisc)

Ni = as.numeric(system("wc -l trip_fare* | cut -d ' ' -f4", intern = TRUE)[-13])-1
N = sum(Ni) #total population--173179759
n = round(N^0.65) #sample size--226476
s = 12 #The number of subsets
r = 13 #the number of bootstrap replicates per subset

f = list.files("/home/data/NYCTaxis", pattern = "trip_fare.*\\.csv$", full.names =
TRUE)
g = list.files("/home/data/NYCTaxis", pattern = "trip_data.*\\.csv$", full.names =
TRUE)

cl = makeCluster(12, "FORK")

cmd1 = paste("cut -f 10,11 -d ,",f) #10--tolls amount, 11--total amount
els1 = clusterSplit(cl,cmd1)

cmd2 = paste("cut -f 9 -d ,",g) #9--trip time in secs
els2 = clusterSplit(cl, cmd2)

Probs = seq(0.1,0.9,0.1)

clusterExport(cl, c("els1","els2"))
Amtbysubset = parLapply(cl, 1:12, function(x) {fread(els1[[x]])[, " total_amount",
                                                with = F] - fread(els1[[x]])[, "
                                                tolls_amount", with = F]})
Amt = unlist(Amtbysubset)

Timbysubset = parLapply(cl, 1:12, function(x) {fread(els2[[x]])[, 1, with = F]})
```

```
Tim = unlist(Timbysubset)

sm = lapply(1:12, function(x) {sample(1:N, n, replace = FALSE)})

clusterExport(cl, c("Amt","Tim"))

SubsetVal_Amt = parLapply(cl, sm, function(x)Amt[x])
SubsetVal_Tim = parLapply(cl, sm, function(x)Tim[x])

clusterCall(cl, library, "Hmisc", character.only = TRUE)

##calculate se of deciles
clusterExport(cl,"SubsetVal_Amt")

Bootstr_Amt=parLapply(cl, SubsetVal_Amt, function(x){
                      sapply(1:13, function(y){tbl = table(sample(x, N,
                      replace = TRUE));
                      wtd.quantile(as.numeric(names(tbl)), weights = tbl,
                      probs = Probs)})})

se_dec = parLapply(cl, Bootstr_Amt, function(x) {apply(x, 1, function(y) sd(y))})

mtx_se.dec = matrix(unlist(se_dec), byrow = TRUE, nrow = 12)

se_d = apply(mtx_se.dec, 2, function(x) mean(x))

##calculate se of betas
Bootstr_AnT = parLapply(cl, 1:12, function(x){sapply(1:r, function(y){
                        ind = sample(sm[[x]], N, replace = TRUE);
                        beta1 = cov(Amt[[ind]],Tim[[ind]])/var(Tim[[ind]]);
                        beta0 = mean(Amt[[ind]]) - beta1*mean(Tim[[ind]]);
                        c(beta1, beta0) })})

se_beta = perLapply(cl, Bootstr_AnT, function(x) {apply(x, 1, function(y) sd(y))})

mtx_se.beta = matrix(unlist(se_beta), byrow = TRUE, nrow = 12)

se_b = apply(mtx_se.beta, 2, function(x) mean(x))
```