Introduction to Big data and regression with R

March 1, 2019

Big data with R

- Problem
  1. R keeps matrices, data.frames etc on the memory. The memory on a laptop is usually not enough for storing really big data sets.
- Possibilities
  1. There are packages efficiently keeping data on the memory: *data.table* and *bigmemory* and more.
  2. There are packages interacting with Database management systems (DBMSs) that keeps the data on the hard drive. For example with the packages *RODBC*, *DBI* and *RHadoop* you can communicate with DBMSs such as *Oracle*, *PostgreSQL*, *Access*, *Hadoop* and more.
  3. The package *ff* saves data directly on the hard-drive and you can access it from R without any DBMS. Or one can see it as a special DBMS for R.
  4. There are packages for handling parallel computing on clusters (multiple cores): *Rmpi*, *snow*, *snowfall* and more. This can reduce computation time dramatically.

Big data using *ff* and *biglm*

- We will work with the *ff* package and the *biglm* package
- The latter includes biglm() and bigglm(), that are using less space on the memory than lm() and glm()
- The package also includes the function update(), which enables sequential estimation on chunks of the data
  - One can import x number of rows (observations) from the hard drive, make computations, remove the chunk and import a new part of the data and continue until the whole data has been processed
  - It is no approximation, you get the same result as if you estimated directly on the complete data set
  - One con is though that the updates have to be made sequentially and not in parallel
    - This disable the use of clusters or using multiple cores on a computer. But I have made my own function which works in parallel.

US Airline data

We will work with a big data set you can find at
`http://stat-computing.org/dataexpo/2009/the-data.html`
Download the data for 2008!

Import data into R with ff

```
rm(list=ls())
#Load the ff packages
library(ff)
library(ffbase)
#Load csv-file as ff data frame
#(this takes some time; like 5 minutes on my laptop)
system.time(ffx08<-
read.csv.ffdf(file="~/2008.csv.bz2",header=TRUE,
na.string=c("",NA),colClasses=
c(Month="factor",DayOfWeek="factor", Year="factor")))
```

OBS: I have the path name on two lines, when you run it in R you have to have it on one line!

Save the ff data frame

If you save the data frame it will only take a second to open it in R next time.

```
#Save data frame
save.ffdf(ffx08, dir = "~/ffx08",
clone = FALSE, relativepath = TRUE,overwrite = TRUE)
#You use this command next time to load the frame to R
load.ffdf(dir="~/ffx08")
```

Subset ff data frame

You can take a subset of the data

```
sub_ffx08<-subset(ffx08,Month==1)
# Drop levels in Month that now are empty
sub_ffx08$Month<-droplevels(sub_ffx08$Month)
```

Basic commands

```
#The following give you some restricted information
summary(ffx08)
#This only give some ff-information about variable Origin
Origin<-ffx08$Origin
Origin
#Takes little memory in R
print(object.size(Origin), units = "Kb")
#This loads Origin as a vector into R
Origin<-ffx08$Origin[]
#Takes much space in R, but still only one vector must be
#very very long if it shouldn't fit into R-space
print(object.size(Origin), units = "Mb")
#Remember to remove objects when
#you don't need them any longer
rm(Origin)
```

Linear Regression

```
#Install and load the package biglm for linear regression
library(biglm)
#Store a formula
form<-ActualElapsedTime~Month+DayOfWeek
#Run regression on the whole dataset and record the time.

 system.time(reg <-biglm(form, data=ffx08,sandwich=FALSE))
 #Get results
 summary(reg)
```

Chunking

With biglm() one can run regressions on quite large data sets but there is a limit. If we want the hard drive to be the limit we can run the regressions on chunks of the data. It is also faster when the the data is large. I have made a function out of this for-loop I googled.

```
Chunk_lm<-function(form,data,chunkSize=1000,sand=FALSE){
 for (i in chunk(data, by=chunkSize)){
 if (i[1]==1){
 biglmfit <- biglm(form, data=data[i,],sandwich=sand)
 }else{
 message("next chunk is: ", i[[1]],":",i[[2]])
 biglmfit <- update(biglmfit, data[i,],sandwich=sand)
 } }
 biglmfit}
```

```
system.time(reg2<-Chunk_lm(form,data=ffx08,
chunkSize=75000))
```

Chunking robust standard errors

```
#Regression results
summary(reg2)
##Robust standard errors take more time.
##Too much time to try it on the lecture but
##you can use the function call below if you
##want to try it later.
#system.time(reg2<-Chunk_lm(form,data=ffx08,
#chunkSize=75000, sandwich=TRUE))
```

Make our own biglm()

- We could reduce computation time if we could make parallel computing on different chunks of the data at the same time
- My laptop has four cores, but at the moment R is only using one for the computations
- If I could compute on all four it would go faster
  - Not four times faster, since R has to distribute and summon chunks and results from each core, but faster
- However, biglm() and update() only works sequentially (don't work in parallel)
- With the aim to compute in parallel, we will go through how to make our own biglm() function that will work in parallel.

Make our own biglm(): Some sufficient theory

The least squares estimator

$$\hat{\beta} = \left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{Y} \tag{1}$$

is equivalent to:

$$\hat{\beta} = \left(\sum_{i=1}^{n} X_i{}'X_i\right)^{-1}\sum_{i=1}^{n} X_i{}'Y_i$$

where $\sum_{i=1}^{n} X_i{}'X_i$ is a sum of a $(K+1)\times(K+1)$ matrices and $\sum_{i=1}^{n} X_i{}'Y_i$ is a sum of $(K+1)\times 1$ vectors. We can compute both of these sums in chunks, e.g.

$$\hat{\beta} = \left(\sum_{i=1}^{n_1} X_i{}'X_i + \sum_{i=n_1+1}^{n} X_i{}'X_i\right)^{-1}\left(\sum_{i=1}^{n_1} X_i{}'Y_i + \sum_{i=n_1+1}^{n} X_i{}'Y_i\right)$$

Make our own biglm()

See the function in Chunk_lm_simple.R. This code makes chunks of the data and compute $\hat{\beta}$ by summing sums from all chunks

Chunk_lm_simple.R instead of biglm()

```
#Load function source("~/Chunk_lm_simple.R")
system.time(reg3<-Chunk_lm(form,
data=ffx08,chunkSize=75000))
```

This takes a few seconds longer than it did when the function was based on biglm(). However this code is straightforward to parallelize.

Parallel computing with package *snow*

- To parallalize, the self-made function we need to split the job in to pieces and send the pieces to different cores on the laptop or on a cluster.
- This is a job for a programmer but luckily for us this has already been done.
- There are several packages for managing parallel computing with R
- We will use the package *snow*

Some basic commands for *snow*

- **cl <- makeCluster(4)**: Prepare computations, makes a cluster on 4 nodes (one on each core if you have a processor with 4 cores)
- **clusterExport(cl, "data")** Exports "data" to each node
- **clusterEvalQ(cl,library(ffbase))** Loads the package "ffbase" to all 4 nodes
- **idx<-clusterSplit(cl, 1:n)**, splits the sequence $1 : n$ on the 4 nodes
- **v<-clusterApply(cl,idx,loop_chunk)**, run the function "loop_chunk" on the 4 nodes at the same time and stores results in v
- **stopCluster(cl)**: Close down the cluster.

Parallelized Chunk_lm_simple.R

Find the function Chunk_lm_simpleII.R

```
rm(Chunk_lm)
#You have to install the package "snow"
#install.packages("snow")
#Load function source("~/Chunk_lm_simpleII.R")
#Run function. On my laptop this is faster
# than any of the other previous functions.
```

```
  system.time(reg4<-Chunk_lm(form,data=ffx08,
  chunkSize=75000,cores=4))
```

Inference

The script Chunk_lm_simpleII.R doesn't include hypothesis testing. If we are working with huge number observations there is really no motivation for not using heteroskedastic-robust inference. This requires an (robust) estimator of the covariance matrix of the betas:

$$\hat{V}(\hat{\beta}) = \left[\sum_{i=1}^{n} X_i' X_i / n\right]^{-1} \sum_{i=1}^{n} \hat{U}_i^2 X_i' X_i / n \left[\sum_{i=1}^{n} X_i' X_i / n\right]^{-1}$$

Inference

For T-testing, we can use the T-statistic:

$$T = \frac{(\hat{\beta}_k - \beta_k)}{\sqrt{\left(\hat{V}(\hat{\beta})\right)_{(k+1),(k+1)} / n}} \overset{a.}{\sim} N(0,1)$$

And the Wald statistic which can be used for joint testing:

$$W = \left[(R\hat{\beta} - r)\right]' \left[R\hat{V}(\hat{\beta})/nR'\right]^{-1} \left[(R\hat{\beta} - r)\right] \overset{a.}{\sim} \chi^2_Q.$$

The script Chunk_lm.R includes a function which produces the T-tests and provides the covariance matrix necessary for making the Wald test.

Remarks about *Chunk_lm.R*

Chunk_lm.R

- The terms $\sum_{i=1}^{n} X_i' X_i$ can be computed without looping in R (see the R-script). Looping takes time in R so with large data one should try to avoid it. Unfortunately I haven't find a way to compute $\sum_{i=1}^{n} \hat{U}_i^2 X_i' X_i$ without looping over all n observations.
- An additional problem with $\sum_{i=1}^{n} \hat{U}_i^2 X_i' X_i$ is that $\hat{U}_i = Y_i - X_i \hat{\beta}$ requires $\hat{\beta}$, so you first have to run *Chunk_lm* to obtain $\hat{\beta}$ and then run the function a second time suppying $\hat{\beta}$ to the function (set *sandwich* = *TRUE*).
- Thus, computing the robust covariance matrix takes some extra time but it is still faster than doing it with a function based on biglm() due to the parallelization.

Example *Chunk_lm.R*

Chunk_lm.R

```
source("~/Chunk_lm.R") system.time(reg<-
Chunk_lm(form,data=ffx08, chunkSize=75000,cores=4))
beta_h<-reg$coef
#Obtain Heteroscedastic-robust T-tests
#Takes little less than 3 min on my laptop
system.time(reg2<-Chunk_lm(form,data=ffx08,
chunkSize=75000,sandwich=TRUE,beta_h=beta_h,cores=4))
round(reg2$summary,4)
```

Example: Testing Joint hypothesis

$$Y = \beta_0 + \sum_{j=2}^{12} \alpha_j Month_j + \sum_{l=2}^{7} \gamma_l Day_l + U_i$$

Test if $E(Y|March, Day_l) - E(Y|Dec, Day_l) = 0$

$$H_0 : \alpha_3 - \alpha_{12} = 0, \quad H_a : \alpha_3 - \alpha_{12} \neq 0$$

```
beta_h<-reg2$coef; K<-length(beta_h)
R<-rbind(numeric(K)); r<-c(0)
R[,rownames(beta_h)=="Month3"]<-1
R[,rownames(beta_h)=="Month12"]<--1
#Wald-value (V is V(beta_hat)/n)
W<-t(R%*%beta_h-r)%*%solve(R%*%reg2$V%*%t(R))%*%
(R%*%beta_h-r)
#P-value
pchisq(W, df=length(r), lower.tail = FALSE)
```

Append the Airline .csv-files

You can merge (or append) two ff data frames like this

```
##Load the csv-file for 2007
ffx07 <-read.csv.ffdf(file="~/2007.csv.bz2",
header=TRUE, na.string=c("",NA),colClasses=
c(Month="factor",DayOfWeek="factor", Year="factor"))
#Append ffx07 and ffx08 as follows ffx0708<-
ffdfappend(ffx07,ffx08)
save.ffdf(ffx0708, dir = "~/ffx0708", clone = FALSE
, relativepath = TRUE,overwrite = TRUE)
summary(ffx0708)
```

if they have the same columns. We can continue to add csv-files like this but make sure to save the appended data set.

Some final conclusions

- You shouldn't expect to understand the code I have provided here in detail
- However, it will give you something to copy and paste from if you would like to do something on your own in the future
- You can also find much more by just google
  - The ff manual is quite lengthy but you can find examples on the web.