Stat243: Problem Set 2

Sicun Huang

September 18, 2015

(a) First pre-process data in bash. Download the zipped file using wget. Then find the number of rows and columns of dataset without unzipping it using bzcat. Get the column numbers of the 13 columns we want and save them to sampleColNum.txt.

```
wget www.stat.berkeley.edu/share/paciorek/ss13hus.csv.bz2

#count number of rows
bzcat ss13hus.csv.bz2 | wc -l > nRows.txt
#replace commas with newlines and then count the number of lines to get the number of columns
bzcat ss13hus.csv.bz2 | head -n1 | tr ',' '\n'| wc -l > nCols.txt

bzcat ss13hus.csv.bz2 | head -n1 > colNames.txt
#loop through all the column names and use awk to return the field numbers corresponding to the 13 colu
awk -F',' '{
for(i=1; i<=NF; i++){
if ($i == "ST" || $i == "NP"|| $i == "BDSP"|| $i == "BLD"|| $i == "RMSP"|| $i == "TEN"|| $i == "FINCP"|
print(i)
}
}
}' < colNames.txt >> sampleColNum.txt
```

Read in the values we obtained with bash into R.

```
#read in the number of rows, columns, and the 13 column numbers from txt files as numeric values
rows <- as.numeric(readLines("nRows.txt"))
cols <- as.numeric(readLines("nCols.txt"))
sampleCols <- as.numeric(readLines("sampleColNum.txt"))
```

Construct an array of characters with "NULL" on the positions corresponding to the column numbers we don't want and "NA" on the ones we want. We will use this array later to read only the columns we are interested in into R.

```
#create character array with length equals to the total number of columns with "NULL" on each position
colChar <- rep("NULL", cols)
#replace "NULL" with "NA" for the columns we want to include
for (i in 1:cols){
  if( i %in% sampleCols ){
    colChar[i] <- NA
  }
}
```

Take a sample of 10000 row numbers out of the total number of rows without replacement and rearrange in increasing order. These rows will be included in the sample. Note that we subtract 1 from the total number of rows to exclude header.

```r
#take a random sample of the row numbers to get the 10000 rows we want to include in sample
sampleRows <- sample(1:rows-1, 10000, replace=FALSE)
#sort row numbers in increasing number
sampleRows <- sort(sampleRows, decreasing = FALSE)
```

Create a data frame with size of the final sample to avoid having to append rows to a data frame. This will speed up the process.

```r
#preallocate memory; construct data frame with size of the final sample
data <- data.frame(matrix(0, nrow=10000, ncol=13))
```

Open R connection to read in data in 100000-row chunks. Use for loops to extract the rows we want from each chunk and save them in the data frame we created before.

```r
#size of each chunk
blockSize <- 100000

#open connection
con <- bzfile("ss13hus.csv.bz2", open="r")

#read in the first row to get rid of header
read.csv( file=con, nrows=1, header=FALSE)

#loop through whole data set in blocks of 100000
for( i in 1:ceiling( (rows-1)/blockSize ) ){
  #read in the ith block with only the columns we want
  block <- read.csv(con, header=FALSE, sep=",", colClasses=colChar, nrows=100000, stringsAsFactors = FA

  #loop through the sampleRows
  for( j in 1:10000){
    #get the sample row numbers within the ith block
    if( sampleRows[j] > (i-1)*blockSize && sampleRows <= i*blockSize){
      #save to the data frame we created before
      data[j,] <- block[sampleRows[j],]
    }
  }
}
#close connection
close(con)
```

(b) Compare the time needed for read.csv and readLines to read in 100000 rows of data respectively.

```r
#open connection
con <- bzfile("ss13hus.csv.bz2", open="r")

#read in the first row to get rid of header
read.csv( file=con, nrows=1, header=FALSE)

#test time for read.csv
system.time(block1 <- read.csv(con, header=FALSE, sep=",", nrows=100000, stringsAsFactors = FALSE ))

## user  system elapsed
## 19.554   0.181  19.739
```

```r
#test time for readLine
system.time(block2 <- readLines(con, n=100000, skipNul = FALSE))

## user  system elapsed
## 18.116   0.068  18.187
```

We can see that readLines is slightly faster than read.csv here.
(c) I determined the number of rows and columns of the original data using bzcat in bash without unzipping the file, which speeded up the processing time.