# STA 250 HW2 Codes

Rohosen Bandyopadhyay

November 14, 2013

## 1 BLB_lin_reg_job.R

```
mini <- FALSE

#============================== Setup for running on Gauss... ==============================#

args <- commandArgs(TRUE)

cat("Command-line arguments:\n")
print(args)

####
# sim_start ==> Lowest possible dataset number
###

###################
sim_start <- 1000
###################

if (length(args)==0){
  sim_num <- sim_start + 1
  set.seed(121231)
} else {
  # SLURM can use either 0- or 1-indexing...
  # Lets use 1-indexing here...
  sim_num <- sim_start + as.numeric(args[1])
  sim_seed <- (762*(sim_num-1) + 121231)
}

cat(paste("\nAnalyzing dataset number ",sim_num,"...\n\n",sep=""))


 s = 5  ###  s is no. of susets each of size b
 r = 50 ###  r is no. of bootstrap samples from each subset

# Find r and s indices:
s_index = floor((sim_num -1001)/r) + 1
#### For the 1st 50 jobs it takes 1, next 50 jobs it takes 2 and so on: 1 to 5
r_index = (sim_num -1001)%%r + 1
#### For the 1st 50 jobs it takes 1 to 50, next 50 jobs it again takes 1 to 50 and so on

#================ Run the simulation study ======================#

# Load packages:
library(BH)
library(bigmemory.sri)
library(bigmemory)
library(biganalytics)

# I/O specifications:
```

```
datapath <- "/home/pdbaines/data"
outpath <- "output/"

# mini or full?
if (mini){
rootfilename <- "blb_lin_reg_mini"
} else {
rootfilename <- "blb_lin_reg_data"
}

# filenames:
infilename <- paste0(rootfilename,".txt")
backingfilename <- paste0(rootfilename,".bin")
descriptorfilename <- paste0(rootfilename,".desc")

# Set up I/O stuff:
infile <- paste(datapath,infilename,sep="/")
backingfile <- paste(datapath,backingfilename,sep="/")
descriptorfile <- paste(datapath,descriptorfilename,sep="/")

### Attaching the data matrix using the description file already created by Prof. Baines ###
data <- attach.big.matrix(dget(descriptorfile),backingpath=datapath)

n = nrow(data)
p = ncol(data)

### We choose b = n^0.7 ####
gamma=0.7 ### used to choose b
b=n^gamma ### b is size of each subset

########### Setting same seed to have same subset when s_index is same#######
set.seed(s_index*1000)

########### Sampling a subset of b rows from the whole data ##########
index = sample(1:n, b, replace = FALSE)
subset = data[index,]

##### Nullifying set.seed and ensuring that for the same subset it produces different bootstrap samples #####
set.seed(s_index*1000 + r_index*10)

################# Drawing a bootstrap sample from the subset #####################
#### So basically want to draw a random sample sample from a multinomial distribution
#### Multinomial sample tells you how many times each unique datapoint of the subset is
 repeated in the bootstrap sample

freq = rmultinom(1, n, rep(1/b, b))

############## Fitting a linear regression to the bootstrap sample ###################
### So basically fitting a linear model to the subset of data with weight equals to the multinomial sample
y = subset[ , p]
X = subset[ , 1:(p-1)]
boot_mod = lm( y ~ X -1, weights = freq) # linear model without the intercept term

####### Output file ###############
write(summary(boot_mod)$coefficients[,1], file =
paste0("output/","coef_",sprintf("%02d",s_index),"_",sprintf("%02d",r_index),".txt"))
```

## 2 MapReducer

**Mapper:**

```
#! /usr/bin/python

import sys
import math

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into two numbers
    numbers = line.split('\t')

    # Calculate the key
    # For each pair of observation it produces x_lo and y_lo as the key
    # x_lo and y_lo of the bin in which the observation should be

    n = float((math.floor(10*float(numbers[0])))/10), float((math.floor(10*float(numbers[1])))/10)

    #convert keys into strings from floats
    num_st1 = str(n[0])
    num_st2 = str(n[1])

    print '%s,%s\t%s' % (num_st1, num_st2, 1)
```

**Reducer:**

```
#! /usr/bin/python

import sys

current_bin = None
current_freq = 0
bin = None

# Input comes from STDIN
for line in sys.stdin:
    #remove trailing '\n'
    line = line.strip()
    # extract key and value from the output of Mapper
    bin, freq = line.split('\t')

    # Convert freq (currently string)into int
    try:
        freq = int(freq)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # Assuming that Hadoop has sorted the output from Mapper before passing it to reducer
    if current_bin == bin:
        current_freq += freq
    else:
        if current_bin:
            x_lo, y_lo = bin.split(',')
            x_hi = str(float(x_lo) + 0.1)
            y_hi = str(float(y_lo) + 0.1)
            print '%s,%s,%s,%s,%s' % (x_lo,x_hi,y_lo,y_hi,current_freq)
        current_freq = freq
        current_bin = bin

# Printing the last one
if current_bin == bin:
    x_lo, y_lo = bin.split(',')
    x_hi = str(float(x_lo) + 0.1)
    y_hi = str(float(y_lo) + 0.1)
    print '%s,%s,%s,%s,%s' % (x_lo,x_hi,y_lo,y_hi,current_freq)
```

## 3 Hive:

```
[Creating an empty table with two columns similar as the data]

hive> CREATE TABLE bigdata(
    > groups INT,
    > obs DOUBLE)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '\t'
    > STORED AS TEXTFILE;

[Loading the data into the table bigdata]

hive > LOAD DATA INPATH 'data/groups.txt' INTO TABLE bigdata;

[Checking how many observations have been read in the table bigdata]

hive> SELECT COUNT(*) FROM bigdata;

[Creating an external table to store group means and within group variances at a specific location]

hive> CREATE EXTERNAL TABLE result(
    > mean DOUBLE,
    > variance DOUBLE)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ' '
    > LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE LOCATION '/user/hadoop/results';

[Calculating group means and within group variances and storing them into the external table]

hive> INSERT INTO TABLE result
    > SELECT AVG(obs), VARIANCE(obs)
    > FROM bigdata
    > GROUP BY groups;

[Checking how many observations have been read in the table result]

hive> SELECT COUNT(*) FROM result;
```