STA 250, HW 2

Longphi Nguyen

November 13, 2013

Problem 1

quantile	0%	25%	50%	75%	100%
SE	0.008525917	0.009691226	0.009961909	0.010279175	0.011785323

Table 1: Quantiles of SE values

The algorithm calculates r =job mod 50 and s =job div 50 to set the seed values per job. There are 5 subsamples, obtained by randomly selecting n^γ observations (with uniform probability and without replacement) from the full dataset. For each subsample, there will be 50 boostrap samples obtained using a multinomial distribution with size n. Then, use the bootstrap sample to do a least squares fit. From Table (1), the SE's are fairly small, as expected. And, from Figure (1), there are no patterns in the SE's that would indicate a noticeable violation of linearity. So, it is expected to that this approach would work fairly well in estimating the SE's.

Problem 2

1. **Mapper("x y")**. Outputs "$x_{hi}, y_{hi}, 1$", which are x and y rounded up to the first decimal.
2. **Reducer("x,y ,1")**. A box is determined by $(x - .1, x, y - .1, y)$. Whenever new inputs x^* and y^* fall in this box, increase the bin count by 1; else, print the number of counts for the box, form a new box with x^* and y^* being the upper bounds, and reset the count to 1.
3. **Implementation**. Mapper() rounds x and y up since the lower bound is supposed to be strictly less than, so we don't want to accidentally output something that looks to be a part of a different box. Also, Mapper only needs to print the upper bounds, since the lower bounds can be determined from them by subtracting .1. Since Hadoop will sort by these upper bounds, the Reducer() then relies on this sorting to determine when a new box occurs.
4. **Time**. Using 2 large core instances, the Mapper() finished in $12:33$ minutes, and the Reducer() finished in $22:45$ minutes. Based on Figure (2), the results seem reasonable because it's pretty.
5. **Improvement**. My Reducer() has an if statement to avoid printing the first read line. However, this check is done for each stdin line, which shouldn't be necessary. So, to avoid redundancy (and possibly a little computation time), this should be fixed. This can likely be done by doing what needs to be done for the first line before the for loop.

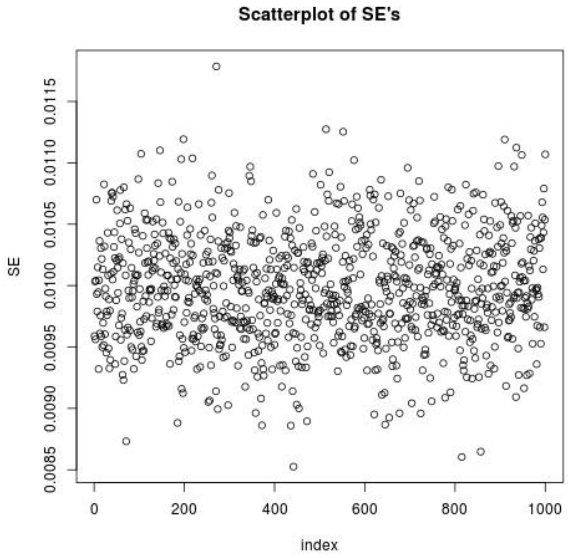

Figure 1: SE for Little Bag of Bootstraps

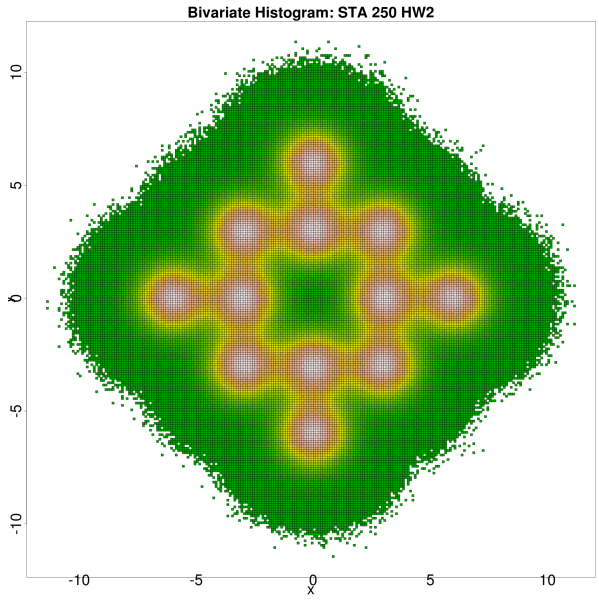

Figure 2: Histogram of Bivariate

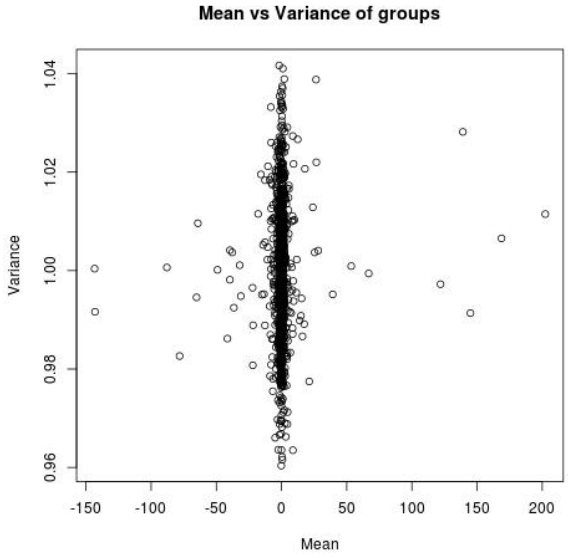

Figure 3: Within-group mean vs variance

Problem 3

Figure (3) shows that there is little variation in the within-group variances, but there is a large variation in the within-group means. Table (2) and Table (3) confirm this numerically.

quantile	0%	10%	25%	50%	75%	90%	100%
Within-Group Mean	-143.43	-3.19	-1.12	-0.04	0.87	2.34	202.34

Table 2: Within-Group mean quantiles show that there is quite a bit of variation in means.

	0%	10%	25%	50%	75%	90%	100%
Within-Group Variance	0.96	0.98	0.99	1.00	1.01	1.02	1.04

Table 3: Within-Group mean quantiles show that the variations were all near 1.

Code

Q1 - *BLB_lin_reg_job.R*

```
library(BH)
library(bigmemory.sri)
library(bigmemory)
library(biganalytics)

#
# ~~==| Preset values |==~~
#
path <- "/home/pdbaines/data"
outpath <- "output/"

mini <- FALSE
# mini or full? n and number of parameters is determined based on this
```

```
# Of course, general code would determine these by reading the file,
# which is unnecessary here.
if (mini){
  rootfilename <- "blb_lin_reg_mini"
n<-10000
numParameters<-41
} else {
  rootfilename <- "blb_lin_reg_data"
n<-1000000
numParameters<-1001
}

s=5
r=50
gamma=0.7
datafile=paste0(path, "/", rootfilename, ".txt")

#
# ~~==| Setup for running on Gauss |==~~
#
args <- commandArgs(TRUE)
cat("Command-line arguments:\n")
print(args)
sim_start <- 1000
if (length(args)==0){
sim_num <- sim_start + 1
set.seed(121231)
} else {
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=""))
job=sim_num-sim_start

#
# ~~==| Preset Values |==~~
#
# n = countLines(datafile) # Get total observations. Already known.
n.subset = floor(n^gamma) # Size of the subset data.

dat<-attach.big.matrix(dget(paste0(path, "/blb_lin_reg_data.desc"))
,backingpath=path) # only for full data

#
# ~~==| Preliminary Steps 1(a): Creating subset sample. |==~~
#
# A function that samples from data to form the subsample.
getSampleFrom<-function(datafile, n, n.subset, numParameters)
{
# Now, sample n^gamma indexes
sampledRows = sample(n, size=n.subset)
  # Use those indexes to get the subsample.
sample=dat[sampledRows,]

invisible(sample) # Same as a return() statement, but faster.
} # end getSampleFrom()
```

```
# Determine s and r based on job number
r_index=((job-1) %% r) + 1 # This is mod(job-1, r)+1
s_index=((job-1) %/% r) + 1 # This is div(job-1, r)+1
set.seed(s_index) # subsample will be the same for each s_index
sample=getSampleFrom(datafile, n, n.subset, numParameters)

#
# ~~==| Bootstrap Sampling 1(b) |==~~
#
set.seed(s_index*r_index) # bootstrap will be different for each s_index
# Determine how many times an observation from subsample appears in the bootstrap.
bs = rmultinom(1, n, prob=rep(1/n.subset, n.subset))

y=sample[,ncol(sample)] # This is the response vector.
obs=sample[,-ncol(sample)] # This is the data minus the response vector.
bs.fit = lm(y ~ obs, weights=bs)$coefficients # Size: small. Hope it fits.

# Write the results from the fit to file 'coef_s_r.txt'
outfile = paste0("output/", "coef_", sprintf("%02d",s_index),
 "_", sprintf("%02d",r_index), ".txt")
write.table(bs.fit, file=outfile,
row.names=FALSE, col.names=FALSE, quote=FALSE, sep=", ")
```

Q2 - *mapper.py*

```
#!/usr/bin/env python

import sys
import math
i
# input comes from STDIN (standard input)
for line in sys.stdin:
    # Remove leading and trailing whitespace
    line = line.strip()
    # Split the line into words
    words = line.split("\t")

    # Get x, y
    x=float(words[0])
    y=float(words[1])

    # Round x and y to 1st decimal.
    x=math.ceil(x*10)/10.0
    y=math.ceil(y*10)/10.0

# Print (key, value) as (x,y ,1)
    print '%s,%s\t,1' % (x,y)
```

Q2 - *reducer.py*

```
#!/usr/bin/env python

import sys

# Initialize a bunch of variables.
x, y, x_lo, x_hi, y_lo, y_hi = None, None, None, None, None, None
```

```
current_count = 0 # Counts the number of observations in current bin.

# Input comes from STDIN.
for line in sys.stdin:
# Remove leading and trailing whitespace
line = line.strip()
# Parse the input we got from mapper.py, which has format (x,y ,count)
x, y, count = line.split(',')

# Convert count, x, y to appropriate type.
try:
count = int(count)
x = float(x)
y = float(y)
except ValueError:
# Conversion failed. Just ignore it.
continue

  # Since Hadoop sorts by key, we can take bins the be in order.
if x_lo < x <= x_hi and y_lo < y <= y_hi: # (x,y) is in current box
current_count += count
else: # update to a new box
if current_count > 0: # need this check so we don't print the first x and y
print '%.1f,%.1f,%.1f,%.1f,%.0f' % (x_lo, x_hi, y_lo, y_hi, current_count)

    # A new box!
x_lo, x_hi = x-.1, x
y_lo, y_hi = y-.1, y

# Number of observations in box increases.
current_count = count

# print the last group
print '%.1f,%.1f,%.1f,%.1f,%.0f' % (x_lo, x_hi, y_lo, y_hi, current_count)
```

Q3 - *Q3HiveCode*

```
CREATE TABLE table1(groups SMALLINT, val FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/home/hadoop/groups.txt' OVERWRITE INTO TABLE table1;

SELECT groups, avg(val), var_samp(val)
FROM table1
GROUP BY groups;
```