STA250 Homework 1 Report

Shuo Li

November 12, 2013

1 Question 1

1. The index plot of the Standard Error values I got from the bootstrap is:

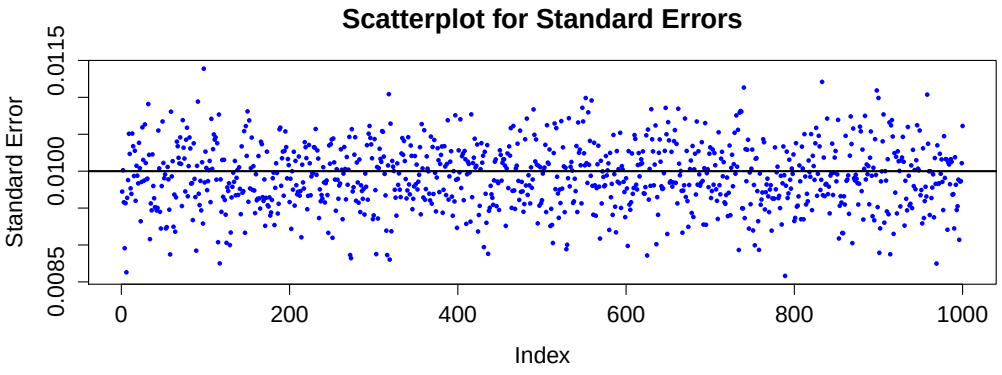

Figure 1:

2. Brief discussion of the algorithm
   The function of the algorithm is to obtain the estimation of regression coefficients from a linear model based on "big data" with $n = 1,000,000$ observations and 1000 covariates.

   The algorithm sample $s = 5$ subsets from the orignial data with observation numbers of $b = n^{0.7}$, which is 15849.

From each extracted subset, we will draw $r = 50$ bootstrap samples, each of size n. This is conducted by sampling from a Multinomial distribution with the probability of each observation to be drew as $1/b$. From this step, we will get the weights for each observations.

Finally, we fit the regression model with the bootstrap samples from the specific subset and plug the weights for the b observations into the function. And the regression coefficients of a weighted linear regression model will be obtained.

From the scatter plot, we can know that the algorithm works pretty well. The standard error for the 1000 regression coefficients all range from 0.0085 to 0.015 and centered around 0.01. The standard error should be around 0.01. And the results of the scatter-plot satisfies this. Thus, we can say that our algorithm is effective and efficient when dealing with this problem.

2 Question 2

1. The 2D histogram or density estimate produced from my MapReduce results is at the top of next page.

2. A description of my MapReduce algorithm.
   I wrote the map-reduce function in Java, which is in the appendix code part. For the map function, it is applied to every data element, and returns a (key,value) pair. The key for each observation is its coordinate range with four parameters, its x-coordinate upper bound, x-coordinate lower bound, y-coordinate upper bound and y-coordinate lower bound. The value for each observation is given as 1.
   For the reduce function, it aggregate the values for every element with the same key(x_lo, x_hi, y_lo, y_hi). To be specific, it adds up the value for the observations within the same xy-coordinate $0.1 * 0.1$ range.
   On Amazon AWS, the program runs for about $15 - 20$ minutes to get the result. So, I think it is pretty efficient.

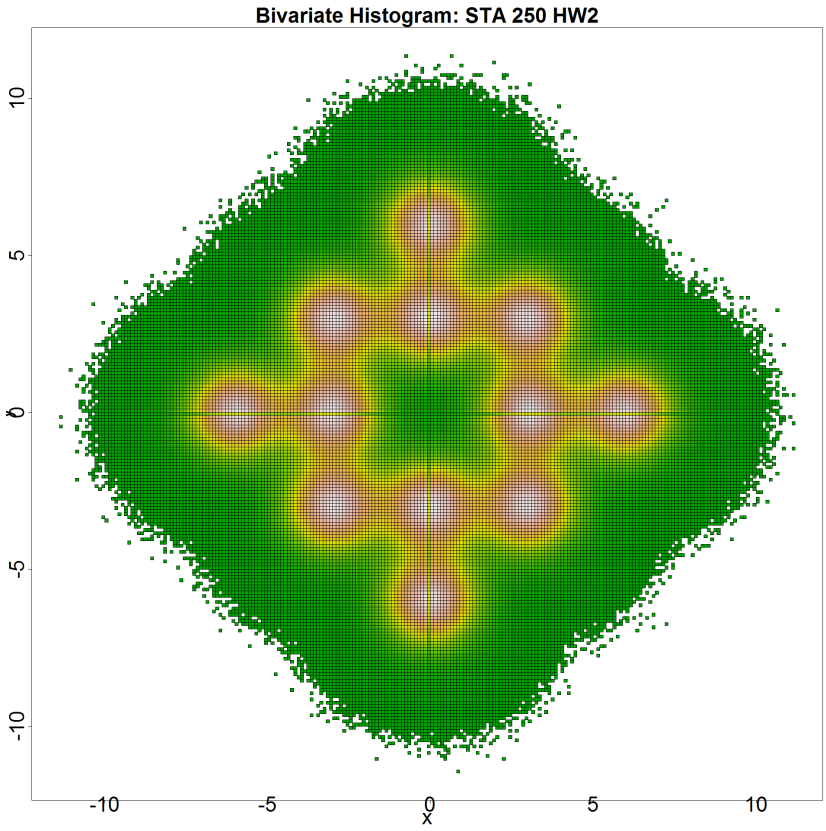

Figure 2:

3 Question 3

The scatterplot of the within-group means (x-axis) vs. the within-group variances (y-axis) is at the top of next page:

4 Appendix Code

```
##Question 1

mini <- FALSE
```

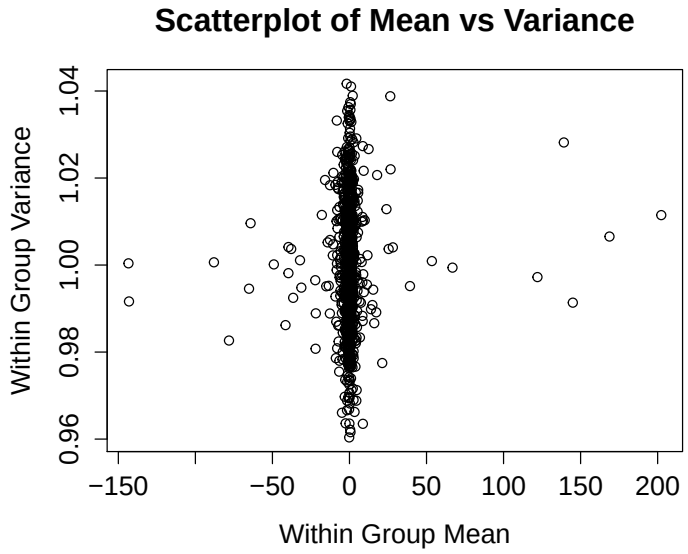

Figure 3:

```
#========== 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=""))

# Find r and s indices:

jobnum=sim_num-1000
if (jobnum %% 50 ==0)
{
  s_index=jobnum/50
  r_index=50
}else
{
  s_index=as.integer(jobnum/50)+1
  r_index=jobnum %% 50
}

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

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


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

# Attach big.matrix :
```

```
dat= attach.big.matrix(dget
     ("/home/pdbaines/data/blb_lin_reg_data.desc"),
                backingpath="/home/pdbaines/data/")

# Remaining BLB specs:

n=nrow(dat)
s = 5
r = 50
gamma = 0.7
b=round(n^gamma)

#sample b rows for the five subsets
set.seed(s_index)
index1=sample(1:n, b, replace = FALSE)

#sample of size n
set.seed(jobnum)
index2=rmultinom(1, n, prob=rep(1/b, b))

# Extract the subset:
data=data.frame(dat[index1, ])

# Fit lm:
coefficient=lm(X1001~.-1, data, weights=index2)$coefficient

# Output file & Save estimates to file:
write.table(coefficient, file =
      paste0("output/","coef_",sprintf("%02d",s_index),
      "_",sprintf("%02d",r_index),".txt"))

## Qustion 2 (Java Code)
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
```

```
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class HW2 {

public static class Map extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text range = new Text();

        public void map(LongWritable key, Text value,
                        OutputCollector<Text, IntWritable> output,
                         Reporter reporter)
                        throws IOException {
                String x = "";
                String y = "";
                double xlo = 0;
                double xhi = 0;
                double ylo = 0;
                double yhi = 0;
                String line = value.toString();

                StringTokenizer tokenizer = new StringTokenizer(line);
                if (tokenizer.hasMoreTokens()) {
                        x = tokenizer.nextToken();
                        double x_value = Double.parseDouble(x);
                        String xv = String.format("%.1f", x_value);
                        double xval = Double.parseDouble(xv);
                        if (x_value > xval - 0.1 &&
                        x_value <= xval) {
                                xlo = xval - 0.1;
                                xhi = xval;
                        } else if (x_value > xval &&
                         x_value <= xval + 0.1) {
                                xlo = xval;
                                xhi = xval + 0.1;
                        }
                }
```

```
                if (tokenizer.hasMoreTokens()) {
                        y = tokenizer.nextToken();
                        double y_value = Double.parseDouble(y);
                        String yv = String.format("%.1f", y_value);
                        double yval = Double.parseDouble(yv);
                        if (y_value > yval - 0.1 &&
                        y_value <= yval) {
                                ylo = yval - 0.1;
                                yhi = yval;
                        } else if (y_value > yval &&
                         y_value <= yval + 0.1) {
                                ylo = yval;
                                yhi = yval + 0.1;
                        }
                }

                String x_lo = String.format("%.1f", xlo);
                String x_hi = String.format("%.1f", xhi);
                String y_lo = String.format("%.1f", ylo);
                String y_hi = String.format("%.1f", yhi);
                range.set(x_lo + "," + x_hi + "," + y_lo + "," + y_hi);
                output.collect(range, one);
        }
}

public static class Reduce extends MapReduceBase implements
                Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values,
                        OutputCollector<Text, IntWritable> output,
                        Reporter reporter)
                        throws IOException {
                int sum = 0;
                while (values.hasNext()) {
                        sum += values.next().get();
                }
                output.collect(key, new IntWritable(sum));
        }
```

```
}

public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(HW2.class);
        conf.setJobName("hw2");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
}
}

##Question 3
CREATE TABLE gro(group INT, value DOUBLE)
             row format delimited fields terminated
             BY '\t' stored AS textfile;
LOAD DATA INPATH '/user/hadoop/data/groups.txt' INTO TABLE gro;

SELECT SUM(value)/COUNT(value)
FROM gro
GROUP BY group;

SELECT (SUM(value * value) - (SUM(value) * AVG(value)))
        / (COUNT(value) - 1)
FROM gro
GROUP BY group;
```