STA 250: HW2: BIG DATA MODULE

CHRISTOPHER CONLEY

Contents

1. Question 1: Bag of Little Bootstraps (BLB)

1.1. **Why BLB over other methods.** The BLB algorithm provides robust quality assessment (i.e. standard errors) of an estimator with the same properties as the bootstrap (e.g. asymptotic consistency) with a focus on being highly scalable to large data sets on distributed computing environments. For big data, the bootstrap was computationally burdensome to repeatedly resample a data set of the original size hundreds of times, where estimation could take days or longer. Work to reduce number of resamples (Efron 1988, Efron & Tibshirani 1993), or other modifications such

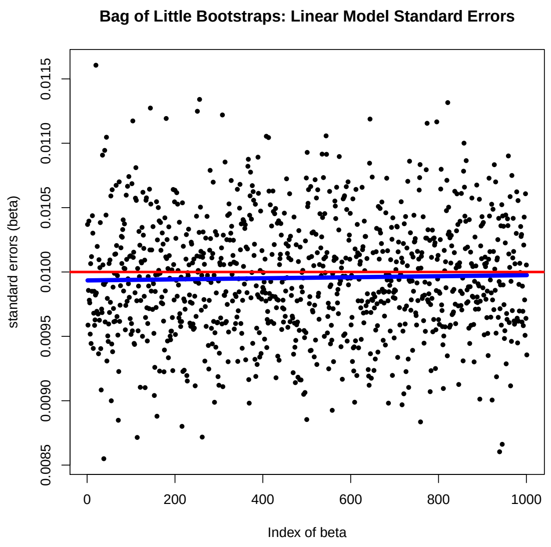

FIGURE 1. The standard errors (s.e) of the linear regression coefficients, β produced by BLB. They are centered near the red line at 0.01, representing the expected neighborhood of the s.e. The blue line is the cross-validated cubic spline smoothed fit to the BLB s.e. which is very close in proximity to the red line.

as the subsampling (Politis et al. 1997) and m out of n bootstrap (Bickel 1997) have failed to reduce the computational burden and still remain entirely robust. With m out of n bootstrap, knowledge of the convergence rates of the estimator is needed for rescaling purposes because the variability of the subsample is different from the whole data set. Extra mathematical conditions such as these limited the practical implementation of this general tool. Kleiner *et al.* successfully proposed an automatic, accurate, and highly scalable version of the Bootsrap.

1.2. **What the BLB does.** Let n be the size of the original data set.

(1) Split the data into $1, \ldots, S$ subsamples of size $b < n$.
(2) For each subsample of at most b unique data points, resample with replacement to the size of the original data set $1, \ldots, R$ times according to a Multinomial with equal probability on the b outcomes.
(3) Compute the quality assessment of the estimator (i.e. standard error) on the R estimates within the subsample.

(4) Average over the S standard errors to get the final quality assessment of the estimator.

1.3. **Key Computational Advantage.** Although the size the subsamples $b << n$ might appear to introduce bias, the weighted resampling based on the Multinomial weights gives the effect of sampling on the scale of the original data set size without the added computational burden or the need for rescaling the quality assessment. In the application of multiple linear regression on 1 million observations with 1K covariates, the subsamples is on the order of thousands (i.e $b = n^{0.7} \approx 1,500$), but the multinomial-weighted least squares fit represents a resample of the larger data set, yet is very computationally efficient because the number of unique observations is at most, the size of b.

Utility Functions

```
##################################################################################
#'@title BLB indexer
#'@description index S subsets and R replications from S*R boostrap procedures
#'@return A vector (s,r) of the index s in 1,..., S, and r in 1,...,R.
#'@examples
#'#see that it works
#'sapply(1:250, function(i) sr_index(i, S, R))
sr_index <- function(jobnum, S, R) {

  #error
  if( jobnum > S*R) {
    stop("job number greater than largest possible index")
  }

  #delimiter of the S subsets
  SR <- S*R
  s_delim <- seq(from = R, to = SR, by = SR / S) + 1
  #offset of R replications
  r_offset <- seq(from = 0, to = SR - R, by = R)


  s <- which(jobnum< s_delim)[1]
  r <- jobnum - r_offset[s]
  c(s,r)
}


plot_blb_se <- function(result_file = "final/blb_lin_reg_data_s5_r50_SE.txt") {
  blb_se <- read.table(file = result_file, header = TRUE)
  #par(cex = 0.5)
  plot(blb_se[,1], pch = 19, cex = 0.65,
       main = "Bag of Little Bootstraps: Linear Model Standard Errors",
       ylab = "standard errors (beta)",
       xlab = "Index of beta")
  sm <- smooth.spline(x = seq_along(blb_se[,1]), blb_se[,1])
  lines(sm$x, sm$y, col = "blue", lwd = 5)
  abline(h = 0.01, col = "red", lwd = 3)
}
#pdf("blb_lin_reg_se.pdf")
#plot_blb_se()
#dev.off()
```

Running the BLB. ____

```
mini <- TRUE
local_machine <- TRUE
verbose <- TRUE
#================================ 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=""))

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

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

# I/O specifications:
#local machine or gauss?
if(!local_machine) {
    datapath <- "/home/pdbaines/data"
} else {
    datapath <- "/Users/dreambig/myrepos/sta250/Stuff/HW2/BLB"
}
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="/")

# Attach big.matrix :
if (verbose){
    cat("Attaching big.matrix...\n")
}
dat <- attach.big.matrix(dget(descriptorfile),backingpath=datapath)

# dataset size (1m):
n <- nrow(dat)

# number of covariates (last column is response):
d <- ncol(dat)-1

# Remaining BLB specs:
```

```
S <- 5
R <- 50
gamma <- 0.7
b <- as.integer(n^gamma)

# Find r and s indices:
source("BLB_utils.R")
sr_pair <- sr_index(sim_num - sim_start, S, R)
s_index <- sr_pair[1]; r_index <- sr_pair[2];

# Extract the subset:

#each subset of S has the same observations
set.seed(s_index)
subset_s <- sample.int(n = n, size = b)
dat_s <- dat[subset_s,]

#data frame is convenient representation for lm() functionality
df_s <- data.frame(y_s = dat_s[,d + 1], X_s = dat_s[,seq_len(d)])

# Reset simulation seed:
#each rth replication has a different resample of rows based on the multinomial
if (length(args)==0) {
    set.seed(121231)
} else {
    set.seed(sim_seed)
```

```
}


# Bootstrap dataset:
bootstrap_resamples <- rmultinom(n = 1, size = n, prob = rep(1/b, times = b))



# Fit lm:
fit_sr <- lm(y_s ~ ., data = df_s, weights = bootstrap_resamples)


# Output file:
outfile = paste0("output/","coef_",sprintf("%02d",s_index),"_",sprintf("%02d",r_index),".t
# Save estimates to file:
beta_lm_estimates <- data.frame(beta_lm_coef = summary(fit_sr)$coefficients[,1])
write.table(x = beta_lm_estimates, file = outfile)
```

2. Question 2: MapReduce for bivariate histogram

2.1. **Map.** The mapping task was to associate each $(x, y) \in R^2$ in a 2-dimensional bin. The mapper implemented was highly abstracted and could be applied to a a D dimensional histogram. Further, I was careful to implement an "if" clause to catch the case when a number fell on the boundary (e.g. 5.5 should be in the bin 5.4 to 5.5) and forced what has been termed as "left inclusion". The map function separated the low and high bin boundaries with commas to keep the key value as similar to the desired output in the end.

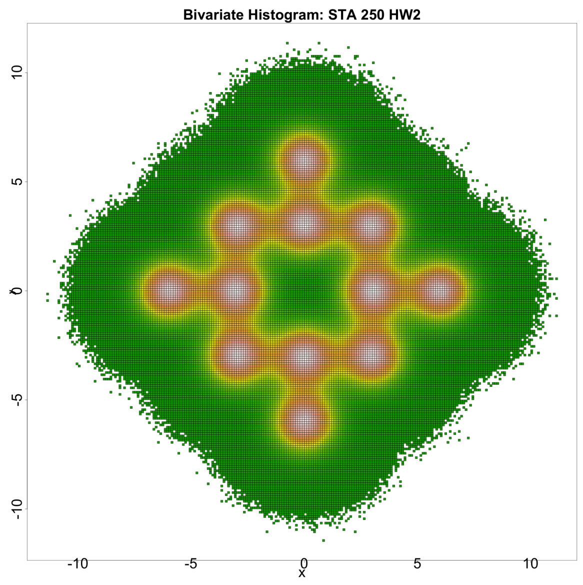

FIGURE 2. The bivariate histogram of the full data set which was computed on the Hadoop Distributed File System on Amazon Elastic Map Reduce framework.

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


# From: http://www.michael-noll.com/tutorials/...
#writing-an-hadoop-mapreduce-program-in-python/
import sys
import math
#the number of lines
D = 4
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line which should be a pair of numbers (x_1, x_2)
    x_d = line.split()
```

```
#keep pairs on same line counters
out_line = [None]*D
d = 0
for x in x_d:
    x = float(x)
    #boundary precision to the tenths decimal place
    x_lo = math.floor(x*10)/10
    x_hi = math.ceil(x*10)/10
    #values lying on the boundary should
    #follow left exclusion principle
    #(e.g. 5.4 < x <= 5.5) when x = 5.5
    if x_lo == x_hi:
        x_lo = x_hi - 0.1
    out_line[d] = x_lo
    out_line[d + 1] = x_hi
    d += 2
# write the results to STDOUT (standard output);
# what we output here will be the input for the
# Reduce step, i.e. the input for reducer.py
trivial_freq = "1"
#delimit the boundaries with a  comma.
key = ','.join(map(str, out_line))
# <key, value>  pair must be delimited by a tab (\t) character
value = "\t{0}".format(trivial_freq)
print  key + value
```

2.2. **Reduce.** The problem structure of the reducer was so similar to the example in class; only instead of counting words, the task was to count the frequency of pre-sorted bin occurrences . Only a minor modification was made to the lecture code, where the final count of the bins was delimited with the key by a comma. Everything else was kept as is. While this code could be improved, it sufficed for this exercise. Because of the similarity of the code, it was relegated to the appendix.

2.3. **Performance & Improvements.** The code took $\approx$ 20 minutes to run. As I am not a "pythonista", certainly improvements could be made to improve the readability and speed. Its hard to know what those improvements might be since "ignorance is bliss".

2.4. **Amazon Elastic Map Reduce Session Commands.**

```
#!/bin/bash

# ==============Step 1================================== #
#from a terminal login to the AWS cluster that was set up
#from the job workflow
ssh -i topher_aws_ec2.pem  hadoop@ec2-50-112-205-143.us-west-2.compute.amazonaws.com

#from a new terminal (inside the sta250/Stuff/HW2/Streaming directory)
#copy code to the AWS cluster that was just set up
scp -i topher_aws_ec2.pem *.py hadoop@ec2-50-112-205-143.us-west-2.compute.amazonaws
scp -i topher_aws_ec2.pem *.sh hadoop@ec2-50-112-205-143.us-west-2.compute.amazonaws
# ==============Step 2================================== #
#create data directory on HDFS
hadoop fs -mkdir data
#copy from special S3 data structure to HDFS
```

```
hadoop distcp s3://sta250bucket/bsamples data/
#...
#wait while it copies
#...
#check success of copy
hadoop fs -ls data/bsamples

# ==============Step 3================================== #
#make sure your code works in the AWS environment
hadoop fs -copyToLocal data/bsamples/out_mini.txt ./
cat out_mini.txt | ./mapper.py | sort -k1,1 | ./reducer.py

# ==============Step 4================================== #
#officially run the MapReduce script, keep track of job number
./hadoop_hw.sh
#in a new terminal, check status of job by:
ssh -i topher_aws_ec2.pem  hadoop@ec2-50-112-205-143.us-west-2.compute.amazonaws.com
#once in AWS environment
lynx http://localhost:9100/

# ==============Step 5================================== #
#If successful, copy the files from HDFS to AWS cluster
hadoop fs -copyToLocal binned-output ./
#Now copy from AWS cluster to local machine
scp -i topher_aws_ec2.pem -r hadoop@ec2-50-112-205-143.us-west-2.compute.amazonaws.com:~/b
```

```
# ==============Step 6=================================== #
#Clean up HDFS on AWS
hadoop fs -rmr data
hadoop fs -rmr binned-output
#Terminate all AWS sessions
```

3. Question 3: Hive Basics

The following code was used generate the within group means and variances. Its primary advantage is that it could read in a tab-delimited file of arbitrarily many columns without explicit declaration. The limitation of the code is that the output of the hive querying code is not formatted appropriately and my efforts to write out to an external table that was formatted correctly did not work.

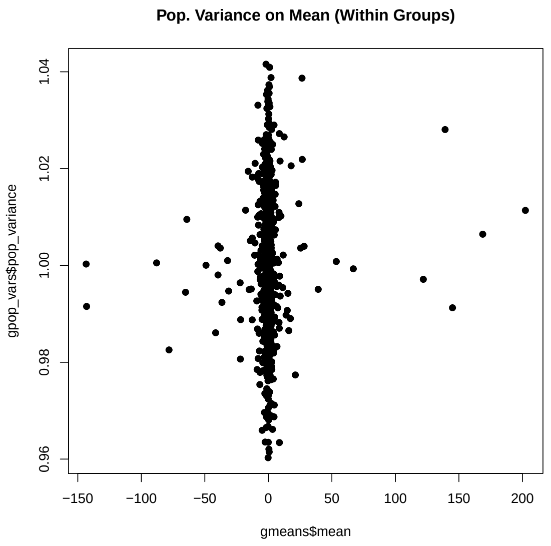

FIGURE 3. The variance (y-axis) is plotted against the mean(x-axis). The within-group means are centered at zero and the within-group variance is centered about 1. There appears to be no obvious dependence between the withing-group mean and variances. This plot contains the population estimates of the variances.

3.1. **Amazon Hive Session Commands.**

```
# ===============Login into AWS================================== #

ssh -i topher_aws_ec2.pem  hadoop@[dns name ]


#================== Copy data to Hadoop ======================#

#for testing

hadoop fs -mkdir data

hadoop distcp s3://sta250bucket/mini_groups.txt data/

hadoop fs -ls data/


#for final data
```

```
hadoop fs -mkdir final_data
hadoop distcp s3://sta250bucket/groups.txt final_data/
hadoop fs -ls final_data/

#inspect file
hadoop fs -copyToLocal final_data/ ./
less final_data/groups.txt

#directory for writing to file
mkdir -p output/means
mkdir output/unbiased_sample_vars
mkdir output/pop_var

# ===========Launch Hive environment============================= #
hive

#avoid great pain, kind of cryptic
set hive.base.inputformat=org.apache.hadoop.hive.ql.io.HiveInputFormat;
set mapred.min.split.size=134217728;

# ================Load data into Hive=========================== #


CREATE EXTERNAL TABLE final_groups (
id int,
value double)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION '/user/hadoop/final_data/';

#make sure "final_groups" table was created
SHOW TABLES;
#To list columns and column types of table.
DESCRIBE EXTENDED final_groups;

# ================Compute within-group means, variances========================== #

#RESOURCE FOR User Defined Functions (UDF)
#https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

#mean
INSERT OVERWRITE LOCAL DIRECTORY 'output/means/'
SELECT final_groups.id, avg(final_groups.value)
FROM final_groups
GROUP BY final_groups.id;

#unbiased sample variance
INSERT OVERWRITE LOCAL DIRECTORY 'output/unbiased_sample_vars'
SELECT final_groups.id, var_samp(final_groups.value)
FROM final_groups
GROUP BY final_groups.id;
```

```
#population variance (is it any different)
INSERT OVERWRITE LOCAL DIRECTORY 'output/pop_var/'
SELECT final_groups.id, var_pop(final_groups.value)
FROM final_groups
GROUP BY final_groups.id;
```

4. Appendix A: Reducer Code

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

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# 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
        word, count = line.split('\t', 1)

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

        # this IF-switch only works because Hadoop sorts map output
        # by key (here: word) before it is passed to the reducer
        if current_word == word:
                current_count += count
        else:
                if current_word:
                        # write result to STDOUT
                        print '%s,%s' % (current_word, current_count)
                current_count = count
                current_word = word

# do not forget to output the last word if needed!
if current_word == word:
        print '%s,%s' % (current_word, current_count)
```