SCHOOL OF MATHEMATICS AND STATISTICS

University of St Andrews

MT5763 GROUP PROJECT:

OPTIMISED BOOTSTRAPPING IN R AND SAS

MODULE CONVENOR: CARL DONOVAN

STUDENT IDS: 180029290, 180028774, 180016660, 180016064, 140008064

WORD COUNT: 2070

SUBMISSION DEADLINE: 05/11/2018

1. INTRODUCTION TO THE PROBLEM

The purpose of this project is to take a simple bootstrapping function and apply the optimisation techniques in order to improve the speed, efficiency and flexibility of the function. Bootstrapping generates 95% confidence intervals for parameter estimates without requiring any distributional assumptions. This is completed by resampling with replacement from the available data. With high repetition, more robust estimates can be generated with lower sampling uncertainty.

In the current report, we will introduce two approaches to bootstrapping optimisation by looking at its implementation via two softwares: RStudio [2] and SAS [3]. We start by exploring the implementation in R, by first describing the improvements we wish to make the necessary mathematical background and then how this was implemented and finally, how it can be used. We then do the same for SAS.

2. R

The optimised code is based on the provided sample, and was created by inspecting the code, and identifying the following parts that could be made faster:

- The *linear model* is a slow function to run, it could then be reformulated as a matrix.
- The *rbind* function is to be replaced with a more efficient alternative.
- Avoid repetition by taking out the *for* loop and *if* statements.
- There are no defined vector sizes, so calculations are repeated more than necessary.
- Similarly we wish to avoid using the function *nrow* multiple times.
- There is only space for a limited number of *x covariates*; we wish to extend on this.
- Parallelize the calculations, using vectorisation wherever possible.

2.1 WHAT MATHS SUPPORTS THE OPTIMISATION?

To optimize the linear model and consider multiple covariates, we rewrite the model into a matrix format, using the following theory. A linear model can be written as $y_i = \beta_0 + \beta_1 x_i + \beta_2 w_i + \cdots + \beta_m z_i + \varepsilon$ where ε is the error.

Writing this in a matrix form $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & x_1 & w_1 & \cdots & z_1 \\ 1 & x_2 & w_2 & & z_2 \\ & \vdots & & \ddots & \vdots \\ 1 & x_n & w_n & \cdots & z_n \end{bmatrix}}_{m \text{ number of columns}} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}$ or simply $Y = X\beta$.

Estimating two or more slope coefficients is straightforward using least squares. We find estimates $\beta_0, \beta_1, \ldots, \beta_m$ that best fit the data by using $\hat{\beta} = (X^T X)^{-1}(X^T Y)$.*

* Taken from Chapter 3 of MT5753: Statistical Modelling notes from 2017.

2.2 How does it work?

We start by parallelising the calculations, by making use of the multiple available computer cores (leaving one aside for general functionality). We then take the response parameter *y* and *x covariates* defined by the user to create our dataset. We scale this data, which allows us to have a more accessible range of values to work with. Then we take this dataset and sample from it with replacement to create a new, sample dataset. We then calculate the parameter estimates for the linear model fitted on the sampled data using the matrix defined above. We repeat this approach a large number of times, take all of these new parameter estimates and return them to the user. This will allow the user to obtain confidence intervals on these parameter estimates.

The optimum function is given by `oneBootToRuleThemAll()` This function takes *at least* three arguments:

- `nBoot`: the number of iterations,
- `yDat`: the response vector,
- `...`: one or more vectors that will form the columns of covariate design matrix

and uses a helper function `parBootAnyCovars()` which takes the data assembled and runs the bootstrapping on it repeatedly using parallelisation on the available cores on the user's computer.

In order to run our R bootstrapping function please initialise the data you wish to use, for example:

```
fitness <- read.csv("~/Desktop/5763-A2/Software-Proj-2/data/fitness.csv")
x <- fitness$Age
y <- fitness$Oxygen
```

and then run the `oneBootToRuleThemAll()` function such as

```
oneBootToRuleThemAll( 100000, regData$y, regData$x)
```

This returns a matrix of scaled parameter estimates, where the first column is that of the intercept, and the following ones are of the x covariates, in order in which they were defined.

2.3 Further Notes

We have implemented four optimised functions, each better than the last. We considered it best to include all of them so we can show our thinking and progress.

`oneBootToRuleThemAll()` has been tested on numerical, factor and interaction terms, hence we can say that the bootstrapping function has been extended not only for any number of covariates but also types.

Efficiency results of the optimisation techniques applied are confirmed in Table 1 - these are also visualised in Figure 2-4 to clearly demonstrate the significant improvements seen.

A profile was taken as well to understand where the bottlenecks (if any) existed, or exist in our final function, which confirms that the parallelisation of the bootstrapping method is the only main time constraint, which is exactly as desired.

3. SAS

The given code is a small and inefficient SAS macro to perform nonparametric bootstrapping. Several areas of improvement are noticed and the SAS macro language is implemented further to increase efficiencies. To implement these, the "*Don't Be Loopy: Re-Sampling and Simulation the SAS Way*" by David Cassell (2007) [4] served as a key resource. The following areas where identified as opportunities to improve the efficiency of the macro:

- Using `DATA` statement to generate the number of rows in the data set.
- Do while loops are inefficient in SAS.
- Unnecessary use of `IF THEN ELSE` statement to store results of Bootstrap method.
- Duplication of `DATA` statement to separately remove unwanted variables from `PROC REG` statement and also rename those variables.

3.1 How improvements were implemented

By utilising optional features in the `PROC SURVEYSELECT` statement we were able to remove the `DATA` statement and requirement of `DO WHILE` loops used in the original code provided.

`PROC SURVEYSELECT` takes several arguments which helped improve the efficiency, specifically;

- The `METHOD` is used to specify the type of random sampling, the Unrestricted Random Sampling (URS) technique was chosen - as per default.
- By using the `SAMPRATE` statement, the prior need to calculate the number of rows of the dataset is replaced by specifying SAS to sample 100% of the original dataset.
- The `REP` statement specifies the number of bootstraps to be generated. This also generates a variable called `REPLICATE` which is ideal to use as a by-variable. Currently, the `REP` is set as equal to `&NumberOfLoops`.

Once the required dataset from `PROC SURVEYSELECT` has been constructed, a regression model is fitted on the new resampled dataset. The regressions are conducted using the `PROC REG` statement which calculates the required number of regression parameter estimates (coefficients) from the dataset using the `BY REPLICATE` statement as this makes the required number of regressions on the resampled datasets. Finally, a `PROC MEANS` statement was called to calculate the mean estimate and 95% confidence intervals of the intercept and parameter(s) in the regression model.

`SASFILE` statement was implemented to load the dataset fully into the RAM and is, therefore, more efficient as opposed to re-loading the data per bootstrap iteration as in standard `PROC SURVEYSELECT` procedure. This removes such an inefficiency by utilising superior memory allocation. That said, this data transfer incurs a minimum

time requirement for the said process to complete, the effect of this requirement makes the utility of `SASFILE` not always certain. In the current analysis, however, no decreases in time were observed hence the statement was used.

4. SUMMARY OF RESULTS

In conclusion, we now have two significantly improved Bootstrapping functions which follow the methodologies outlined above in sections 2 and 3 (R and SAS respectively).

To help visualise how the techniques used in these new functions differ relative to the baseline code provided, a flow chart was created. Figure 1 clearly illustrates the different approaches required to achieve efficient code in the differing software applications used.

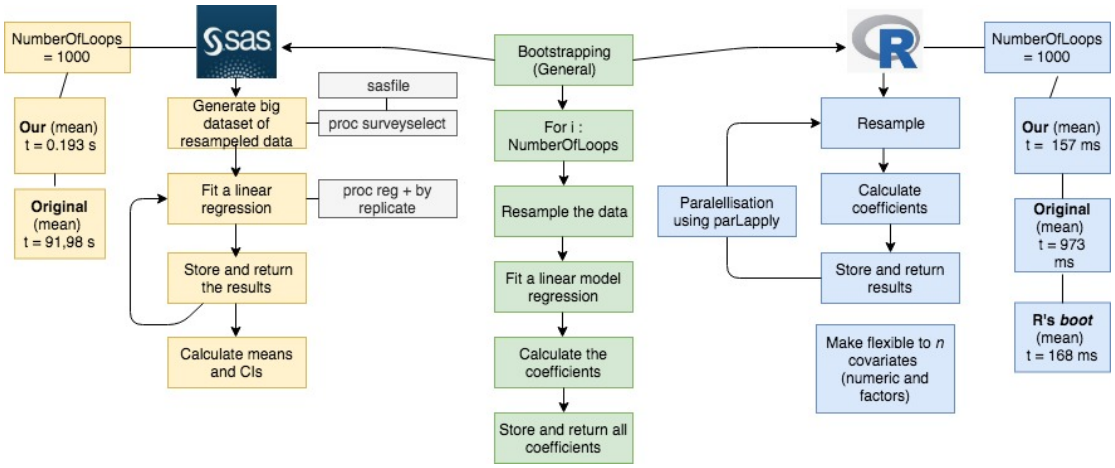

Figure 1 - Visualisation of bootstrapping approaches taken for R and SAS

4.1 IMPROVEMENTS IN EFFICIENCY OF R BOOTSTRAPPING METHODS

Improvements made to R functions saw the time taken to complete the Bootstrapping method reduce from 973.104 milliseconds on average to 157.423 - equivalent to 6 fold improvement in speed. A comparison was also completed using 'boot' function from the 'boot' package in R. On average results showed our function was 11 seconds quicker (7% improvement). All results quoted can be found in Table 1 below and are visualized in Figures 2-4.

R Functions	BootStrap Iterations	Average Time to Complete (in milliseconds)	Lower Quartile Time to Complete (in milliseconds)
Original - lmBoot	1,000	973.104	127.146
R Boot Comparision - usingBootLibrary	1,000	168.185	152.613
Final - oneBootToRuleThemAll	1,000	157.423	127.146

Table 1 - Summary of completion time of R functions(in milliseconds rounded to 3 decimal places). Times generated using the 'microbenchmark' function in R, from the 'microbenchmark' package.

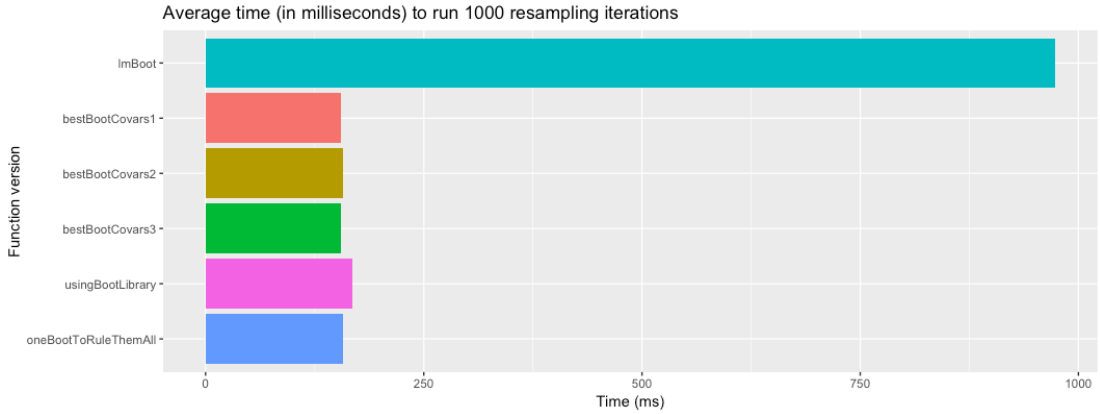

Figure 1 - Average times to complete resampling for 1000 iterations in milliseconds

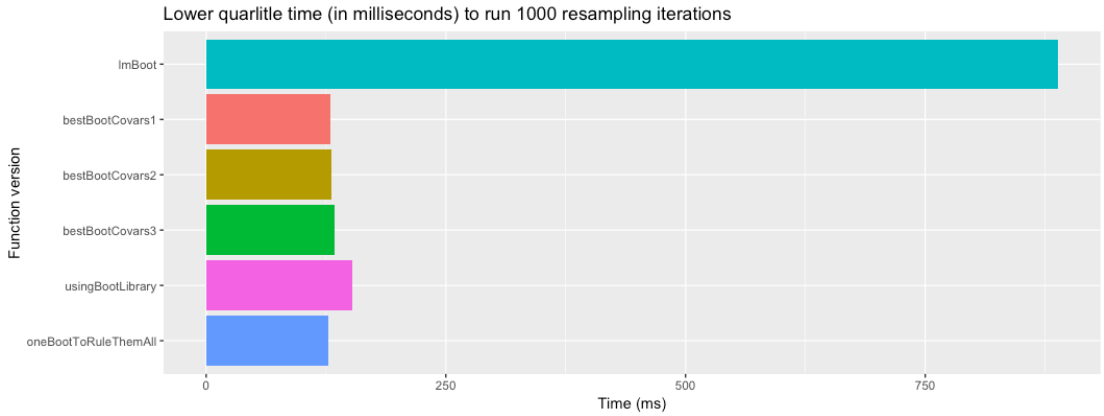

Figure 2 - Lower quartile times to complete resampling for 1000 iterations in milliseconds

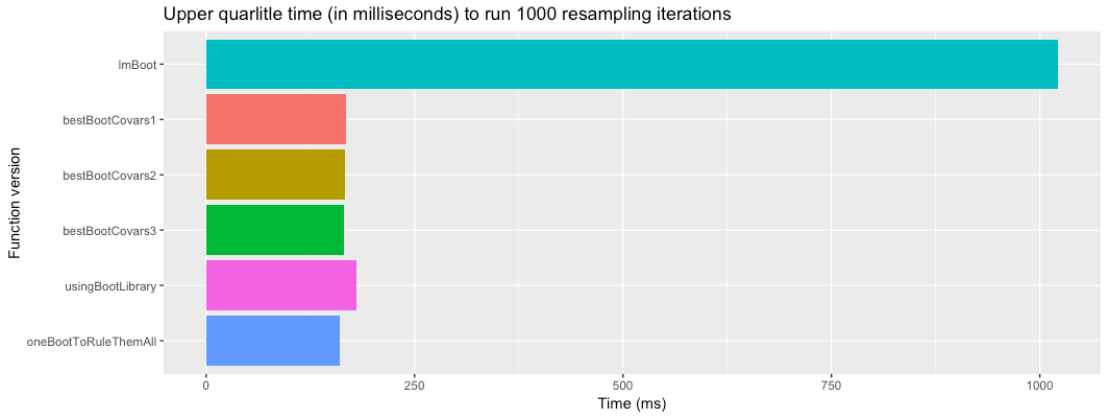

Figure 3 - Upper quartile times to complete resampling for 1000 iterations in milliseconds

4.2 Improvements in efficiency of SAS Bootstrapping methods

Improvements made to the SAS macro saw the time taken to complete the Bootstrapping method falling from 91.980 seconds to 0.193 seconds - equivalent to 476 fold improvement in speed.

SAS Macro	BootStrap Iterations	Number of Timing Tests Performed	Average Time to Complete (in seconds)
Original	1,000	10	91.980
Update 1 - wt. SASFile Load	1,000	10	0.187
Update 2 - incl. SASFile Load	1,000	10	0.183
Final w. RTF Output File	1,000	10	0.193

Table 2 - Summary of completion time of SAS Macros (in seconds rounded to 3 decimal places). where 'wt.' stands for without, 'incl.' stands for include & 'w.' stands for with.

REFERENCES

[1] R: A Language and Environment for Statistical Computing, R Core Team, R Foundation for Statistical Computing, Vienna, Austria, 2018, https://www.R-project.org/

[2] RStudio Team (2018). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL http://www.rstudio.com/

[3] SAS 9.4, SAS Institute Inc., Cary, NC, USA

[4] Cassell, David L. 2007. *Don't Be Loopy: Re-Sampling And Simulation The SAS® Way*. Ebook. 1st ed. Corvallis. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.487.6242.

APPENDIX I - FINAL R BOOTSTRAPPING FUNCTION

```
parBadBoot <- function( index, scaleData, N )
{
  # sample data with replacement
  bootData <- scaleData[sample( 1:N, N, replace = T ),]
  Xmat <- bootData[,1:2]
  Ymat <- bootData[,3]

  # fit the model under this alternative reality
  # Changed the lm part to matrix form
  beta <- solve( t( Xmat ) %*% Xmat ) %*% t( Xmat ) %*% Ymat
  bootResults <- beta
}

oneBootToRuleThemAll <- function( nBoot, yDat, ... )
{
  # parallelize
  library(parallel)
  nCores <- detectCores()
  myClust <- makeCluster(nCores-1, type = "FORK")

  # save x covariates into list
  xDat <- list(...)
  # create data matrix in one step by using scale function only once
  scaleData <- cbind(1, scale(do.call(cbind, xDat, yDat), scale =F ))

  N <- length( yDat )

  # pass scaled data to helper function to bootstrap
  tempbootResults <- parLapply( myClust, 1:nBoot, parBootAnyCovars,
                                scaleData = scaleData, N = N )
  # save all bootstap data into matrix and output
  m <- matrix( unlist(tempbootResults), ncol = length(xDat)+1)

  return( m )
}
```

APPENDIX II - FINAL SAS BOOTSTRAPPING FUNCTION

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/* FINAL MACRO for Group Assignment 2 for MT5763 */
/*Our Team SAS macro - name and comments to be added by the group
once Sam has stepped through*/
/*what the macro actual does*/
/*Inputs: */
/*- NumberOfLoops: the number of bootstrap iterations */
/*- Dataset: A SAS dataset containing the response and covariate */
/*- XVariable: The covariate for our regression model (gen.
continuous numeric) */
/*- YVariable: The response variable for our regression model (gen.
continuous numeric) */
/*Outputs: */
/*- RTF file: */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
%macro regBootFour(NumberOfLoops, DataSet, XVariable, YVariable);

    /* SASFILE statement allows us load the data set into buffers in
RAM which hold the file, then reads the data into memory */
    sasfile &Dataset load;
    /* PROC SURVEYSELECT allows us to generate random samples of many
kinds from an input dataset*/
    /* Invoke PROC SURVEYSELECT and assign it the input and output
data set names*/
    /* Specify a random seed */
    /* Use METHOD= option to specify the type of random sampling:
Unrestricted Random Sampling(URS)*/
    /* Employ the SAMPRATE= option to get a sample of the same size
as original data set without having to figure out the size of dataset
first*/
    /* Set the number of bootstrap samples that we wish generate*/
    proc surveyselect data = &DataSet out = bootData4
            seed = -180029290
            method = urs
            noprint
            samprate = 100
            rep = &NumberOfLoops;
      run;

    sasfile &Dataset close;
    /* SASFILE CLOSE frees up the RAM buffers when we're done with
the file*/
    /* Perform a regression on this randomised dataset and generate
parameter estimates*/
    /* Use the variable REPLICATE as the by-variable in our
procedure*/
    /* Take the already-written code and insert BY statement to get
the necessary bootstrap realizations */
      proc reg data = bootdata4 outest = parameterestimates  noprint;
            by Replicate;
            model &yvariable = &xvariable;
      run;
    /* Assign new results dataset and rename the resultant
variables*/
      data ResultHolder4;
            set ParameterEstimates;
            keep Intercept &XVariable;
            rename Intercept = RandomIntercept &XVariable =
RandomSlope;
```

```
      run;
    /* To simplify the contents of our list file or output window, we
bracket the procedure with ODS LISTING statements*/
    /* ODS LISTING CLOSE turns off the ODS destination that has our
list output*/
    /* While ODS LISTING turns it back on*/
    /* This approach replaces the old NOPRINT option*/
    /* Set the final output results to an RTF file by using the
Output Delivery System*/
    /* Invoke Proc Means procedure to calculate 95% confidence
intervals for the mean, and the mean estimate of each parameter*/
    ods listing close;
      ods                 rtf                 file                =
'C:\Users\Lenovo\Desktop\SAS_project\teamOutput.rtf';
           proc means data = ResultHolder4 mean lclm uclm alpha =
0.025;
                 var RandomIntercept RandomSlope;
           run;
      ods rtf close;
      ods listing;

%mend;
```