SESUG Paper 158-2019

A Random Forest Example of the Boston Housing Data using the Base SAS$^{®}$ and the PROC_R macro in SAS$^{®}$ Enterprise Guide

Melvin Alexander, Analytician

ABSTRACT

This presentation used the Boston Housing data to call and execute R code from the Base SAS® environment to create a Random Forest. SAS makes it possible to run R code via SAS/IML®, SAS/IML Studio®, or SAS/Viya® as described by Gilsen (2018). Interfacing R with these other SAS modules requires separate and installations that incur additional costs.

R has a rich set of machine learning, text mining packages, and advanced graphic capabilities and complements SAS. I will demonstrate the R and Base SAS integration to create a Random Forest using a the %PROC_R macro of Wei (2012).

INTRODUCTION

R integration with Base SAS has been possible using special macros as described by Wei (2012) and Bettinger (2016), or using the Java object described by Hall (2015), Revolution Analytics (2015), and Wujek (2015).

I will demonstrate this capability using the %PROC_R macro by Wei (2012).

To integrate R and base SAS, follow these steps:

1. Download the PROC_R macro code from Wei (2012).
2. Save the file in a location (e.g., "P:\My SAS Files\Random Forest Example using Base SAS PROC_R macro in Enterprise Guide.sas").
3. Open the code and update the path of R executable file in the code below.

```
* add the location path where R version 3.5.0 is installed ;
    %macro quit(rpath=%str(C:\Progra~1\R\R-3.5.0\bin\R.exe));
```

Here the rpath refers to location where the R executable is saved on the same machine where SAS is installed. Be sure to install the R library packages used for the analysis.

4. Open Base SAS and call the PROC_R3 macro, a modification of Xin Wei's PROC_R macro (2012) that executes R code in base SAS. The Proc_R3_fgname.sas program (with the %PROC_R3 macro) suppresses macro variables &fgname and &fgsw from being printed to the SAS log that avoids errors when running the program.
5. Run R inside SAS environment. See the SAS program below:

```
%include "P:\My SAS Files\Proc_R3_fgname.sas"  ;
%Proc_R3(SAS2R =, R2SAS=) ;
  cards4 ;
  /********/
  R code ;
  /*******/
```

```
dev.off()
;;;;
%quit;
```

The macro variables are described as follows:

SAS2R - specifies the names of SAS datasets converted into R dataframes or objects. The dataset can be single file names or multiple files names separated by spaces.
R2SAS - specifies the names of R dataframes or objects that convert into SAS datasets. These objects can be single file names or multiple files names separated by spaces.

Example

Random Forest is an Ensemble-Supervised Learning technique used to improve the predictive performance of Decision Trees by reducing the variance in the Trees through averaging.

Although Decision Trees are simple and easily interpretable among Modelling techniques, they tend to have a poor predictive performance on Generalized Test DataSets. See A.**S.** Walla (**2017**), "Random Forests in R", at https://www.r-bloggers.com/random-forests-in-r/ for more information.

I will use the Response variable 'medv' which is the Median Housing Value from the Boston Housing dataset that came with the MASS package of R to fit 500 Trees and reproduce the graphical output depicted in the Test and Out of Bag Error plot.

Example Run R code from Base SAS and Enterprise Guide

```
*************************************************************************************************
*********** A macro that execute R script in base SAS ******************************************
** MACRO Version:  1.0                                                                        **
** SAS Version:    9.1.3/9.2                                                                  **
** R Version:      2.14.0                                                                     **
** Date:           Nov 24, 2011                                                               **
** Author:         Xin Wei, Ph.D.                                                             **
** Affiliation:    Roche Pharmaceuticals, INC                                                 **
** Instruction:    This SAS macro enables native R language to be embedded in and executed    **
**                 along with a SAS program in the Base SAS environment under Windows OS.     **
**                 This macro executes a user-defined R code in batch mode by calling the     **
**                 unnamed pipe method within base SAS. The R textual and graphical output    **
**                 can be routed to the SAS output window and result viewer, respectively.    **
**                 Also, this macro automatically converts data between SAS datasets and R    **
**                 data frames such that the data and results from each statistical           **
**                 environment can be utilized by the other environment. The objective of this**
**                 macro is to leverage the strength of the R programming language within the **
**                 SAS environment in a systematic manner. Moreover, this macro helps         **
**                 statistical programmers to learn a new statistical language while staying  **
**                 in a familiar environment. The macro variables are described as follows:   **
**                                                                                            **
**                 SAS2R - specifies the names of SAS datasets to be converted to R dataframe.**
**                 Can be single file name or multiple files whose names are separated by     **
**                 space.                                                                     **
**                 R2SAS - specifies the names of R dataframes to be converted to SAS         **
**                 datasets. Can be single file name or multiple files whose names are        **
**                 separated by space.                                                        **
**                 rpath - The full path and file names of R executable file for various R    **
**                 version from 2.11 to 3.5.0                                                 **
**                                                                                            **
**                                                                                            **
** References:    https://www.jstatsoft.org/article/view/v046c02                              **
************************************************************************************************;
```

```
options nomerror nomprint nomlogic nosymbolgen;
options nonotes;
%include "P:\My SAS Files\Proc R.sas"  ;
%Proc_R(SAS2R =, R2SAS = predlim3) ;
cards4 ;
# Random Forest is an Ensemble-Surpervised Learning technique used to improve the
# predictive performance of Decision Trees by reducing the variance in the Trees through
averaging.
# Although Decision Trees are simple and easily interpretable among Modelling techniques,
# they tend to have a poor predictive performance on Generalized Test DataSets.
# See A.S. Walla (2017), "Random Forests in R",
# at https://www.r-bloggers.com/random-forests-in-r/ for more information.

# load libraries
 library(randomForest) # create Random Forest
 library(MASS)         # contains Boston Housing dataset
 library(ggplot2)      # enhanced data visuals

 attach(Boston)

# train=sample(1:nrow(Boston),300)

# Boston.rf=randomForest(medv ~ . , data = Boston , subset = train)
#Boston.rf
head(Boston)
str(Boston)
summary(Boston)


#split Boston into Training and Validation Sets
# Training:Validation Split = 70:30 (random)
set.seed(101)
train <- sample(nrow(Boston), 0.7*nrow(Boston), replace = FALSE)
Boston.rf=randomForest(medv ~ . , data = Boston , subset = train,
       mtry=13, importance=TRUE)
#Boston.rf

#importance(Boston.rf)

#x11()
#plot(Boston.rf)
# from https://www.r-bloggers.com/random-forest-classification-of-mushrooms/
# Variable Importance of Variable Importance having the greatest impact in the prediction of the
Training Set

#x11()
#varImpPlot(Boston.rf,
#           sort = T,
#           main="Variable Importance on the Training Set")

oob.err=double(13)
test.err=double(13)

#mtry is no of Variables randomly chosen at each split
for(mtry in 1:13)
  {
  rf=randomForest(medv ~ . , data = Boston , subset = train,mtry=mtry,
     ntree=500, importance = TRUE)
  oob.err[mtry] = rf$mse[500] #Error of all Trees fitted

  pred<-predict(rf,Boston[-train,]) #Predictions on Test Set for each Tree
  test.err[mtry]= with(Boston[-train,], mean( (medv - pred)^2)) #Mean Squared Test Error

  cat(mtry," ") #printing the output to the console

  }

# list predicted medv (pred) from the test data Boston[-train,]
```

```
#pred
Boston[-train,]$medv
mean_pred <- (( pred - Boston[-train,]$medv)^2)
sd.pred <- mean(sqrt( (pred - Boston[-train,]$medv)^2))

test.err
oob.err

# Find the optimal mtry value (mtry with the smallest OOBError)
# https://www.listendata.com/2014/11/random-forest-with-r.html
x11()
mtry <- tuneRF(Boston[-1],Boston$medv, ntreeTry=500,
                stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
 best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
 print(mtry)
 print(best.m)

# use the best.m mtry value for the matplot
x11()
png( filename="P:/My SAS Files/TestOOB.png", bg="white")
matplot(1:best.m ,main="Figure 1: Plot of Test Error and Out of Bag Error",
cbind(oob.err,test.err), pch=19,
  col=c("red","blue"),type="b",ylab="Mean Squared Error",
  xlab="Number of Predictors Considered at each Split")

 legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19, col=c("red","blue"))
# Bagging from
# http://rpubs.com/kangrinboqe/268745
# Boston.test are the medv values from the test set
# yhat.bag are the predicted medv values using the random forest formula
# from the training data

Boston.test = Boston [-train ,"medv"]
yhat.bag = pred
x11()
plot( yhat.bag , Boston.test, main="Bivariate Plot Test Observations with Fitted Line")
abline (0 ,1)

mean (( yhat.bag - Boston.test)^2)

# Grow the random forest using the mtry argument.
# By default, randomForest() uses p/3 variables to build the random forest
# of regression trees, and sqrt(q)variables when building a random
# forest of classification trees. Here we use mtry = best.m.
# https://rpubs.com/Bio-Geek/71339 is another reference

rf.boston = randomForest( medv~.,data =Boston , subset =train ,
mtry =best.m, importance = TRUE )

# https://uc-r.github.io/random_forests
x11()
plot(rf.boston)

#find which number of trees providing the lowest error
# number of trees with lowest MSE
which.min(rf.boston$mse)
## [1] 214

# RMSE of this optimal random forest
sqrt(rf.boston$mse[which.min(rf.boston$mse)])

yhat.rf = pred
mean ((yhat.rf - Boston.test )^2)

importance(rf.boston)

x11()
png( filename="P:/My SAS Files/VarImp.png", bg="white")
varImpPlot(rf.boston,
           sort = T,
           main="Figure 2: Variable Importance on the Test Set")
```

```
# Get the Predictions on the Test Set for each Tree
pred.Boston.rf = predict (rf , newdata =Boston[-train ,], interval="prediction", predict.all=TRUE
)

#head(t(pred.Boston.rf))

pred.Boston.rf.int <- t(apply(pred.Boston.rf$individual, 1, function(x) {
  c(mean(x) + c(-1.96, 1.96) * sd(x),
  quantile(x, c(0.025, 0.975)))
}))

#https://stats.stackexchange.com/questions/56895/do-the-predictions-of-a-random-forest-model-
have-a-prediction-interval
# Calculate the prediction intervals by taking the assumption of normality.
# From each of the individual trees we have the predicted value (µi)
# as well as the mean squared error (MSEi). So prediction from each
# individual tree can be thought of as N(µi,RMSEi).
# Using the normal distribution properties our prediction from the random
# forest would have the distribution N(?µi/n,?RMSEi/n).
# Apply the rnorm error to this example we get the below results
# for pred.fr.int4.


pred.rf.int4 <- sapply(1:nrow(pred.Boston.rf$individual), function(i) {
   tmp <- pred.Boston.rf$individual[i,] + rnorm(1000, 0, sqrt(rf.boston$mse))
   quantile(tmp, c(0.025, 0.975))
   })

#t(pred.rf.int4)

mean.rf <- pred.Boston.rf$aggregate
sd.rf <- mean(sqrt(rf.boston$mse))
pred.rf.int3 <- cbind(mean.rf - 1.96*sd.rf, mean.rf + 1.96*sd.rf)
#pred.rf.int3
colnames(pred.rf.int3) <- c("LPL1", "UPL1")

predlim2 <- data.frame(cbind(pred.Boston.rf.int,pred.rf.int3))
colnames(predlim2) <- c("X", "Y", "LCL", "UCL", "LPL1", "UPL1")
# summary(pred.Boston.rf.int)

# t(pred.Boston.rf.int)
#predlim <- data.frame(t(pred.Boston.rf.int))
predlim3 <- data.frame(predlim2)#dataframe retrieved as a SAS dataset
#head(predlim)
#colnames(predlim) <- c("X", "Y", "LCL", "UCL")
colnames(predlim3) <- c("X", "Y", "LCL", "UCL", "LPL", "UPL")
#predlim[1:5,]
#predlim3[1:10,]
x11()
#plot(predlim)
#plot(predlim3)

# find outlier observations below LPL or above UPL
#outliers <- subset(predlim3, Y > UCL | Y < LCL)
#outliers
# find outlier observations below LPL or above UPL
outliers3 <- subset(predlim3, Y > UPL | Y < LPL )
print(outliers3)
#specify plot point colors and size that fall outside the confidence limits
predlim3$color <- ifelse(predlim3$Y > predlim3$UPL | predlim3$Y < predlim3$LPL, "red", "black")
predlim3$size <- ifelse(predlim3$Y > predlim3$UPL | predlim3$Y < predlim3$LPL, 3, 1)

x11()
png( filename="P:/My SAS Files/PLErrorBars.png", bg="white")
p1 <- ggplot(predlim3, aes(x = X, y = Y))
p1 + geom_errorbar(aes(ymin=LPL, ymax=UPL ), width=.1) +
          geom_point(color=predlim3$color, size=predlim3$size) +
          geom_abline(intercept=0, slope=1, linetype=2) +
          xlab("yhat.bag") +
          ylab("Boston.test") +
       ggtitle("Figure 3: Prediction Interval Error Bars for the Random Forests")
# Use the rfinterval to construct prediction intervals for random forest
#  predictions using a fast implementation package 'ranger'.
# https://rdrr.io/github/haozhestat/rfinterval/man/rfinterval.html
# from Haozhe Zhang

library(rfinterval)
# medv~.,data =Boston , subset =train
output <- rfinterval(medv~., train_data = Boston[train,], test_data = Boston[-train,],
                     method = c("oob"),
                     symmetry = TRUE,alpha = 0.1)
y <- Boston[-train,]$medv
# get the oob prediction accuracy of rfinterval
oob_prediction_accuracy <- mean(output$oob_interval$lo < y & output$oob_interval$up > y)

#mean(output$sc_interval$lo < y & output$sc_interval$up > y)
#mean(output$quantreg_interval$lo < y & output$quantreg_interval$up > y)

output$oob_interval

rf_oob_pred_int <- data.frame(cbind(y,output$oob_interval))

oob_outliers <- subset(rf_oob_pred_int,rf_oob_pred_int$y < rf_oob_pred_int$lower |
rf_oob_pred_int$y > rf_oob_pred_int$upper)

#specify plot point colors and size that fall outside the confidence limits
rf_oob_pred_int$color <- ifelse(rf_oob_pred_int$y > rf_oob_pred_int$upper | rf_oob_pred_int$y <
rf_oob_pred_int$lower, "red", "black")
rf_oob_pred_int$size <- ifelse(rf_oob_pred_int$y > rf_oob_pred_int$upper | rf_oob_pred_int$y <
rf_oob_pred_int$lower, 3, 1)

x11()
#png( filename="P:/My SAS Files/PLErrorBars.png", bg="white")
p2 <- ggplot(rf_oob_pred_int, aes(x = 1:nrow(rf_oob_pred_int), y = y))
p2 + geom_errorbar(aes(ymin=lower, ymax=upper ), width=.1) +
          geom_point(color=rf_oob_pred_int$color, size=rf_oob_pred_int$size) +
          xlab("Index") +
          ylab("Y's") +

       ggtitle("Figure 5: Out-of-Bag Prediction Intervals for Random Forests by Zhang et
al.(2019)")


oobpi_ratio_percent <- 100 * (nrow(oob_outliers)/nrow(rf_oob_pred_int))

oob_prediction_accuracy
1 - (nrow(oob_outliers)/nrow(rf_oob_pred_int))

dev.off()
;;;;
%quit;

ods graphics / width=700px height=700px;
title "Figure 4: Prediction Limit Plot of Test Medv (Y=Boston,test) by Random Forest
Medv(X=yhat.bag) with Lower and Upper Errors from the SGPLOT Procedure";
proc sgplot data=work.predlim3;
styleattrs datacontrastcolors=(black red);
scatter x=x y=y /
  yerrorlower=lpl yerrorupper=upl  markerattrs=(size=4PT symbol=CircleFilled ) group=color;
 xaxis label= 'yhat.bag';
 yaxis label= 'Boston.test';
run;
```

Table 1 is the R output log output from R generated from executing the R program.

Table 1: R Output Log

******************R OUTPUT***********************

R_OUTPUT_LOG

```
> library(grDevices)
> # Random Forest is an Ensemble-Surpervised Learning technique used to improve the
> # predictive performance of Decision Trees by reducing the variance in the Trees through averaging.
> # Although Decision Trees are simple and easily interpretable among Modelling techniques,
> # they tend to have a poor predictive performance on Generalized Test DataSets.
> # See A.S. Walla (2017), "Random Forests in R",
> # at https://www.r-bloggers.com/random-forests-in-r/ for more information.
>
> # load libraries
> library(randomForest) # create Random Forest
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
Warning message:
package 'randomForest' was built under R version 3.5.1
> library(MASS) # contains Boston Housing dataset
> library(ggplot2) # enhanced data visuals

Attaching package: 'ggplot2'

The following object is masked from 'package:randomForest':

margin

>
> attach(Boston)
>
> # train=sample(1:nrow(Boston),300)
>
> # Boston.rf=randomForest(medv ~ . , data = Boston , subset = train)
> #Boston.rf
> head(Boston)
crim zn indus chas nox rm age dis rad tax ptratio black lstat
1 0.00632 18 2.31 0 0.538 6.575 65.2 4.0900 1 296 15.3 396.90 4.98
2 0.02731 0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8 396.90 9.14
3 0.02729 0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8 392.83 4.03
4 0.03237 0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7 394.63 2.94
5 0.06905 0 2.18 0 0.458 7.147 54.2 6.0622 3 222 18.7 396.90 5.33
6 0.02985 0 2.18 0 0.458 6.430 58.7 6.0622 3 222 18.7 394.12 5.21
medv
1 24.0
2 21.6
3 34.7
4 33.4
5 36.2
6 28.7
> str(Boston)
'data.frame':506 obs. of 14 variables:
$ crim : num 0.00632 0.02731 0.02729 0.03237 0.06905 ...
$ zn : num 18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
$ indus : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
$ chas : int 0 0 0 0 0 0 0 0 0 0 ...
$ nox : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
$ rm : num 6.58 6.42 7.18 7 7.15 ...
$ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
$ dis : num 4.09 4.97 4.97 6.06 6.06 ...
$ rad : int 1 2 2 3 3 3 5 5 5 5 ...
$ tax : num 296 242 242 222 222 222 311 311 311 311 ...
$ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
$ black : num 397 397 393 395 397 ...
$ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
$ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
> summary(Boston)
crim zn indus chas
Min. : 0.00632 Min. : 0.00 Min. : 0.46 Min. :0.00000
1st Qu.: 0.08204 1st Qu.: 0.00 1st Qu.: 5.19 1st Qu.:0.00000
Median : 0.25651 Median : 0.00 Median : 9.69 Median :0.00000
Mean : 3.61352 Mean : 11.36 Mean :11.14 Mean :0.06917
3rd Qu.: 3.67708 3rd Qu.: 12.50 3rd Qu.:18.10 3rd Qu.:0.00000
Max. :88.97620 Max. :100.00 Max. :27.74 Max. :1.00000
nox rm age dis
Min. :0.3850 Min. :3.561 Min. : 2.90 Min. : 1.130
1st Qu.:0.4490 1st Qu.:5.886 1st Qu.: 45.02 1st Qu.: 2.100
Median :0.5380 Median :6.208 Median : 77.50 Median : 3.207
Mean :0.5547 Mean :6.285 Mean : 68.57 Mean : 3.795
3rd Qu.:0.6240 3rd Qu.:6.623 3rd Qu.: 94.08 3rd Qu.: 5.188
Max. :0.8710 Max. :8.780 Max. :100.00 Max. :12.127
rad tax ptratio black
Min. : 1.000 Min. :187.0 Min. :12.60 Min. : 0.32
1st Qu.: 4.000 1st Qu.:279.0 1st Qu.:17.40 1st Qu.:375.38
Median : 5.000 Median :330.0 Median :19.05 Median :391.44
Mean : 9.549 Mean :408.2 Mean :18.46 Mean :356.67
3rd Qu.:24.000 3rd Qu.:666.0 3rd Qu.:20.20 3rd Qu.:396.23
Max. :24.000 Max. :711.0 Max. :22.00 Max. :396.90
lstat medv
Min. : 1.73 Min. : 5.00
1st Qu.: 6.95 1st Qu.:17.02
Median :11.36 Median :21.20
Mean :12.65 Mean :22.53
3rd Qu.:16.95 3rd Qu.:25.00
Max. :37.97 Max. :50.00
>
>
> #split Boston into Training and Validation Sets
> # Training:Validation Split = 70:30 (random)
> set.seed(101)
> train <- sample(nrow(Boston), 0.7*nrow(Boston), replace = FALSE)
> Boston.rf=randomForest(medv ~ . , data = Boston , subset = train,
+ mtry=13, importance=TRUE)
> #Boston.rf
>
> #importance(Boston.rf)
>
> #x11()
> #plot(Boston.rf)
> # from https://www.r-bloggers.com/random-forest-classification-of-mushrooms/
> # Variable Importance of Variable Importance having the greatest impact in the prediction of the Training Set
>
```

```
> #x11()
> #varImpPlot(Boston.rf,
> # sort = T,
> # main="Variable Importance on the Training Set")
>
> oob.err=double(13)
> test.err=double(13)
>
> #mtry is no of Variables randomly chosen at each split
> for(mtry in 1:13)
+ {
+ rf=randomForest(medv ~ . , data = Boston , subset = train,mtry=mtry,
+ ntree=500, importance = TRUE)
+ oob.err[mtry] = rf$mse[500] #Error of all Trees fitted
+
+ pred<-predict(rf,Boston[-train,]) #Predictions on Test Set for each Tree
+ test.err[mtry]= with(Boston[-train,], mean( (medv - pred))) #Mean Squared Test Error
+
+ cat(mtry," ") #printing the output to the console
+
+ }
1 2 3 4 5 6 7 8 9 10 11 12 13 >
> # list predicted medv (pred) from the test data Boston[-train,]
>
> #pred
> Boston[-train,]$medv
[1] 21.6 22.9 20.4 23.1 19.6 14.8 24.7 24.7 21.2 20.0 16.6 20.5 19.6 16.0 22.2
[16] 23.5 24.2 21.7 23.4 20.0 21.2 22.5 23.6 28.7 22.6 20.6 26.5 19.5 19.8 21.7
[31] 18.5 19.2 20.4 19.3 16.2 19.2 15.6 17.8 11.8 15.6 14.6 17.8 13.1 23.3 27.0
[46] 50.0 22.7 17.4 22.6 29.4 39.8 36.2 32.5 26.4 37.0 50.0 34.9 48.5 24.4 20.0
[61] 19.3 28.1 25.0 28.7 31.6 31.5 31.7 24.0 20.1 24.3 26.2 42.8 21.9 44.0 50.0
[76] 30.1 30.7 50.0 43.5 24.4 32.0 45.4 50.0 32.2 20.1 23.2 28.5 29.0 24.8 22.0
[91] 36.1 16.1 22.1 21.6 19.8 23.1 23.1 25.0 23.0 22.2 19.5 20.6 19.0 23.1 22.9
[106] 24.1 17.8 22.6 27.5 50.0 13.8 15.0 13.9 13.3 13.1 11.5 15.1 23.2 13.8 12.7
[121] 8.5 6.3 5.0 7.5 8.3 9.5 16.1 14.3 15.4 14.9 15.2 14.1 14.9 17.7 20.2
[136] 19.9 19.0 29.8 13.8 14.6 23.0 20.6 19.1 7.0 8.1 13.6 21.8 23.1 16.8 22.4
[151] 20.6 22.0
> mean_pred <- (( pred - Boston[-train,]$medv))
> sd.pred <- mean(sqrt( (pred - Boston[-train,]$medv)))
>
> test.err
[1] 23.40122 16.55777 13.97286 13.18084 12.66905 12.42798 11.95564 11.99878
[9] 11.94832 11.86225 11.76736 11.88645 11.28114
> oob.err
[1] 20.41008 13.79837 12.06895 11.49062 11.68513 11.83577 12.35682 12.32138
[9] 12.80082 12.63496 12.90943 12.86492 12.51172
>
> # Find the optimal mtry value (mtry with the smallest OOBError)
> # https://www.listendata.com/2014/11/random-forest-with-r.html
> x11()
> mtry <- tuneRF(Boston[-1],Boston$medv, ntreeTry=500,
+ stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
mtry = 4 OOB error = 1.628136
Searching left ...
mtry = 3 OOB error = 2.597524
-0.5953969 0.01
Searching right ...
mtry = 6 OOB error = 0.7320872
0.5503527 0.01
```

```
mtry = 9 OOB error = 0.211393
0.7112461 0.01
mtry = 13 OOB error = 0.0324375
0.8465536 0.01
> best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
> print(mtry)
mtry OOBError
3 3 2.5975237
4 4 1.6281364
6 6 0.7320872
9 9 0.2113930
13 13 0.0324375
> print(best.m)
[1] 13
>
> # use the best.m mtry value for the matplot
> x11()
> png( filename="P:/My SAS Files/TestOOB.png", bg="white")
> matplot(1:best.m ,main="Figure 1: Plot of Test Error and Out of Bag Error", cbind(oob.err,test.err), pch=19,
+ col=c("red","blue"),type="b",ylab="Mean Squared Error",
+ xlab="Number of Predictors Considered at each Split")
>
> legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19, col=c("red","blue"))
> # Bagging from
> # http://rpubs.com/kangrinboqe/268745
> # Boston.test are the medv values from the test set
> # yhat.bag are the predicted medv values using the random forest formula
> # from the training data
>
> Boston.test = Boston [-train ,"medv"]
> yhat.bag = pred
> x11()
> plot( yhat.bag , Boston.test, main="Bivariate Plot Test Observations with Fitted Line")
> abline (0 ,1)
>
> mean (( yhat.bag - Boston.test))
[1] 11.28114
>
> # Grow the random forest using the mtry argument.
> # By default, randomForest() uses p/3 variables to build the random forest
> # of regression trees, and sqrt(q)variables when building a random
> # forest of classification trees. Here we use mtry = best.m.
> # https://rpubs.com/Bio-Geek/71339 is another reference
>
> rf.boston = randomForest( medv~.,data =Boston , subset =train ,
+ mtry =best.m, importance = TRUE )
>
> # https://uc-r.github.io/random_forests
> x11()
> plot(rf.boston)
>
> #find which number of trees providing the lowest error
> # number of trees with lowest MSE
> which.min(rf.boston$mse)
[1] 214
> ## [1] 214
>
> # RMSE of this optimal random forest
> sqrt(rf.boston$mse[which.min(rf.boston$mse)])
```

```
[1] 3.563423
>
> yhat.rf = pred
> mean ((yhat.rf - Boston.test ))
[1] 11.28114
>
> importance(rf.boston)
%IncMSE IncNodePurity
crim 17.2466172 1319.61406
zn 2.3771903 64.85451
indus 7.7979414 173.15059
chas 0.9922664 27.28803
nox 17.9284278 885.90481
rm 42.9534837 8941.67235
age 10.3925827 465.26823
dis 19.3746532 1436.51395
rad 5.0243841 138.74945
tax 8.8181222 258.10621
ptratio 17.3433894 529.80473
black 8.2376423 347.55626
lstat 39.1764411 14571.55721
>
> x11()
> png( filename="P:/My SAS Files/VarImp.png", bg="white")
> varImpPlot(rf.boston,
+ sort = T,
+ main="Figure 2: Variable Importance on the Test Set")
>
> # Get the Predictions on the Test Set for each Tree
> pred.Boston.rf = predict (rf , newdata =Boston[-train ,], interval="prediction", predict.all=TRUE )
>
> #head(t(pred.Boston.rf))
>
> pred.Boston.rf.int <- t(apply(pred.Boston.rf$individual, 1, function(x) {
+ c(mean(x) + c(-1.96, 1.96) * sd(x),
+ quantile(x, c(0.025, 0.975)))
+ }))
>
> #https://stats.stackexchange.com/questions/56895/do-the-predictions-of-a-random-forest-model-have-a-prediction-
interval
> # Calculate the prediction intervals by taking the assumption of normality.
> # From each of the individual trees we have the predicted value (µi)
> # as well as the mean squared error (MSEi). So prediction from each
> # individual tree can be thought of as N(µi,RMSEi).
> # Using the normal distribution properties our prediction from the random
> # forest would have the distribution N(?µi/n,?RMSEi/n).
> # Apply the rnorm error to this example we get the below results
> # for pred.fr.int4.
>
>
> pred.rf.int4 <- sapply(1:nrow(pred.Boston.rf$individual), function(i) {
+ tmp <- pred.Boston.rf$individual[i,] + rnorm(1000, 0, sqrt(rf.boston$mse))
+ quantile(tmp, c(0.025, 0.975))
+ })
>
> #t(pred.rf.int4)
>
> mean.rf <- pred.Boston.rf$aggregate
> sd.rf <- mean(sqrt(rf.boston$mse))
> pred.rf.int3 <- cbind(mean.rf - 1.96*sd.rf, mean.rf + 1.96*sd.rf)
> #pred.rf.int3
> colnames(pred.rf.int3) <- c("LPL1", "UPL1")
>
> predlim2 <- data.frame(cbind(pred.Boston.rf.int,pred.rf.int3))
> colnames(predlim2) <- c("X", "Y", "LCL", "UCL", "LPL1", "UPL1")
> # summary(pred.Boston.rf.int)
>
> # t(pred.Boston.rf.int)
> #predlim <- data.frame(t(pred.Boston.rf.int))
> predlim3 <- data.frame(predlim2) ) # dataframe retrieved as a SAS dataset
> #head(predlim)
> #colnames(predlim) <- c("X", "Y", "LCL", "UCL")
> colnames(predlim3) <- c("X", "Y", "LCL", "UCL", "LPL", "UPL")
> #predlim[1:5,]
> #predlim3[1:10,]
> x11()
> #plot(predlim)
> #plot(predlim3)
>
> # find outlier observations below LPL or above UPL
> #outliers <- subset(predlim3, Y > UCL | Y < LCL)
> #outliers
> # find outlier observations below LPL or above UPL
> outliers3 <- subset(predlim3, Y > UPL | Y < LPL )
> print(outliers3)
X Y LCL UCL LPL UPL
89 25.760744 42.03827 24.40125 41.30000 26.755049 41.04396
160 11.007926 44.56819 16.50000 50.00000 20.643599 34.93251
161 11.026707 45.11743 21.28250 50.00000 20.927612 35.21653
162 31.791573 58.63960 24.85563 50.00000 38.071129 52.36004
176 18.816438 34.68701 22.56500 33.81687 19.607265 33.89618
181 26.264490 53.34783 27.44217 50.00000 32.661702 46.95062
182 17.567757 32.88522 18.70000 29.60000 18.082032 32.37095
184 19.127892 38.31346 23.08958 48.57500 21.576219 35.86513
185 13.614091 31.82808 15.93800 29.60000 15.576629 29.86554
191 24.144719 38.49247 23.50000 36.84200 24.174135 38.46305
196 35.293828 53.48582 33.42375 50.00000 37.245365 51.53428
204 37.114011 54.32917 34.48325 50.00000 38.577132 52.86605
210 10.181892 26.35321 13.14875 27.10000 11.123095 25.41201
212 11.714089 26.30449 12.95000 23.70000 11.864832 26.15375
228 24.773760 42.36705 24.19000 46.60000 26.425945 40.71486
230 15.376361 46.92049 22.80000 50.00000 24.003969 38.29288
232 24.015587 43.08922 23.58000 48.23937 26.407945 40.69686
254 27.924854 51.69295 28.04750 50.00000 32.664442 46.95336
257 26.105036 47.99091 31.17063 50.00000 29.903515 44.19243
258 30.478285 57.92220 21.90000 50.00000 37.055782 51.34470
260 18.626772 47.78057 22.80000 50.00000 26.059215 40.34813
267 12.975698 40.00974 14.79750 36.53150 19.348259 33.63717
268 27.298717 55.58726 27.48250 50.00000 34.298529 48.58744
269 27.603703 55.24364 31.26188 50.00000 34.279215 48.56813
276 25.673614 40.20296 25.58917 43.80000 25.793829 40.08274
281 37.555304 53.04398 36.40000 50.00000 38.155182 52.44410
284 34.617516 53.54363 33.35792 50.00000 36.936115 51.22503
287 13.520989 31.50145 16.50000 29.60000 15.366762 29.65568
291 24.595057 40.08338 26.60000 39.41550 25.194759 39.48367
312 16.565875 32.86779 19.91188 38.35250 17.572372 31.86129
366 2.336492 47.63711 11.90000 50.00000 17.842342 32.13126
372 12.347201 64.68781 15.00000 50.00000 31.373049 45.66196
```

[1]

```
376 5.237738 46.36445 10.24750 50.00000 18.656635 32.94555
378 4.470051 23.77367 10.94750 26.84167 6.977402 21.26632
417 4.722001 20.74898 7.96425 24.34312 5.591032 19.87995
433 12.823711 28.29159 10.92375 25.01750 13.413192 27.70211          [ 1 ]
474 14.902018 34.24336 17.80000 34.04937 17.428232 31.71715
502 15.484971 31.42643 11.90000 29.31962 16.311242 30.60016
503 9.406889 27.91472 11.90000 25.00000 11.516349 25.80526
505 16.967231 36.35976 22.73562 41.30000 19.519039 33.80795
> #specify plot point colors and size that fall outside the confidence limits
> predlim3$color <- ifelse(predlim3$Y > predlim3$UPL | predlim3$Y < predlim3$LPL, "red", "black")
> predlim3$size <- ifelse(predlim3$Y > predlim3$UPL | predlim3$Y < predlim3$LPL, 3, 1)
>
> x11()
> png( filename="P:/My SAS Files/PLErrorBars.png", bg="white")
> p1 <- ggplot(predlim3, aes(x = X, y = Y))
> p1 + geom_errorbar(aes(ymin=LPL, ymax=UPL ), width=.1) +
+ geom_point(color=predlim3$color, size=predlim3$size) +
+ geom_abline(intercept=0, slope=1, linetype=2) +
+ xlab("yhat.bag") +
+ ylab("Boston.test") +
+
+ ggtitle("Figure 3: Prediction Interval Error Bars for the Random Forests")
>> # Use the rfinterval to construct prediction intervals for random forest
> #  predictions using a fast implementation package 'ranger'.
> # https://rdrr.io/github/haozhestat/rfinterval/man/rfinterval.html
> # from Haozhe Zhang
>
> library(rfinterval)
Warning message:
package ‘rfinterval’ was built under R version 3.5.3
> # medv~.,data =Boston , subset =train
> output <- rfinterval(medv~., train_data = Boston[train,], test_data = Boston[-train,],
+                     method = c("oob"),
+                     symmetry = TRUE,alpha = 0.1)
> y <- Boston[-train,]$medv
> # get the oob prediction accuracy of rfinterval
> oob_prediction_accuracy <- mean(output$oob_interval$lo < y & output$oob_interval$up > y)
>
> #mean(output$sc_interval$lo < y & output$sc_interval$up > y)
> #mean(output$quantreg_interval$lo < y & output$quantreg_interval$up > y)
>
> #output$oob_interval
>
> rf_oob_pred_int <- data.frame(cbind(y,output$oob_interval))
>
> oob_outliers <- subset(rf_oob_pred_int,rf_oob_pred_int$y < rf_oob_pred_int$lower | rf_oob_pred_int$y >
rf_oob_pred_int$upper)
>
> #specify plot point colors and size that fall outside the confidence limits
> rf_oob_pred_int$color <- ifelse(rf_oob_pred_int$y > rf_oob_pred_int$upper | rf_oob_pred_int$y <
rf_oob_pred_int$lower, "red", "black")
> rf_oob_pred_int$size <- ifelse(rf_oob_pred_int$y > rf_oob_pred_int$upper | rf_oob_pred_int$y <
rf_oob_pred_int$lower, 3, 1)
>
> x11()
> #png( filename="P:/My SAS Files/OOBRFPIs.png", bg="white")
> p2 <- ggplot(rf_oob_pred_int, aes(x = 1:nrow(rf_oob_pred_int), y = y))
> p2 + geom_errorbar(aes(ymin=lower, ymax=upper ), width=.1) +
+          geom_point(color=rf_oob_pred_int$color, size=rf_oob_pred_int$size) +
+          xlab("Index") +
+          ylab("Y's") +
+
+       ggtitle("Figure 5: Out-of-Bag Prediction Intervals for Random Forests by Zhang et al.(2019)")
>
>
> oobpi_ratio_percent <- 100 * (nrow(oob_outliers)/nrow(rf_oob_pred_int))
>
> oob_prediction_accuracy
[1] 0.8486842
> 1 - (nrow(oob_outliers)/nrow(rf_oob_pred_int))
[1] 0.8486842
> dev.off()
windows

> write.csv(predlim3,'predlim3.csv',row.names=F)
> q()
> proc.time()
user system elapsed
15.06 0.68 20.61
```

[1] Observations containing prediction interval outliers are listed after the *print(outliers3)* command in Table 1 and shown in Figure 3.

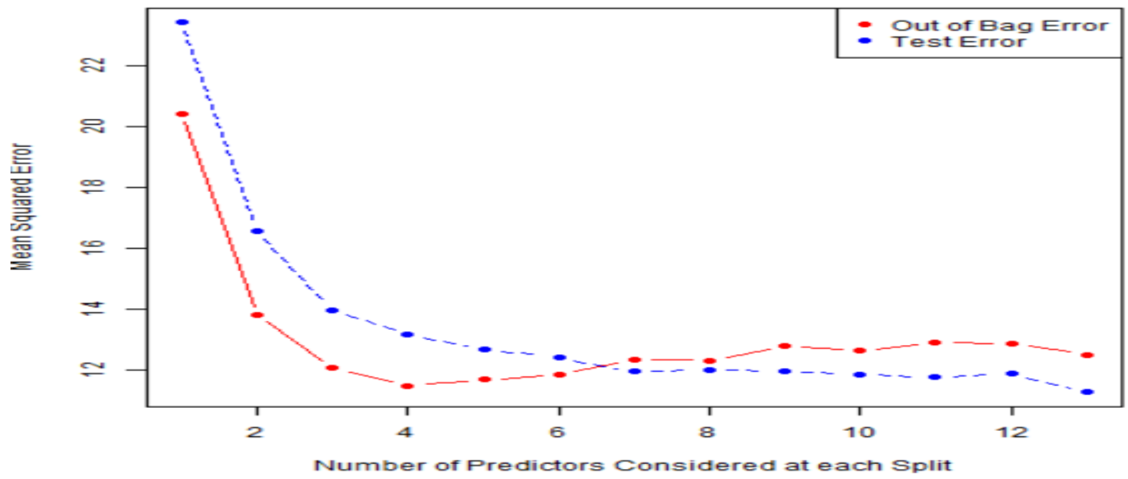


Figure 1: Plot of Test Error and Out of Bag Error

From Figure 1, four is the number of predictor variables that produced the smallest mean squared error for the fitted trees (Out of Bag error).

Figure 2: Variable Importance on the Test Set

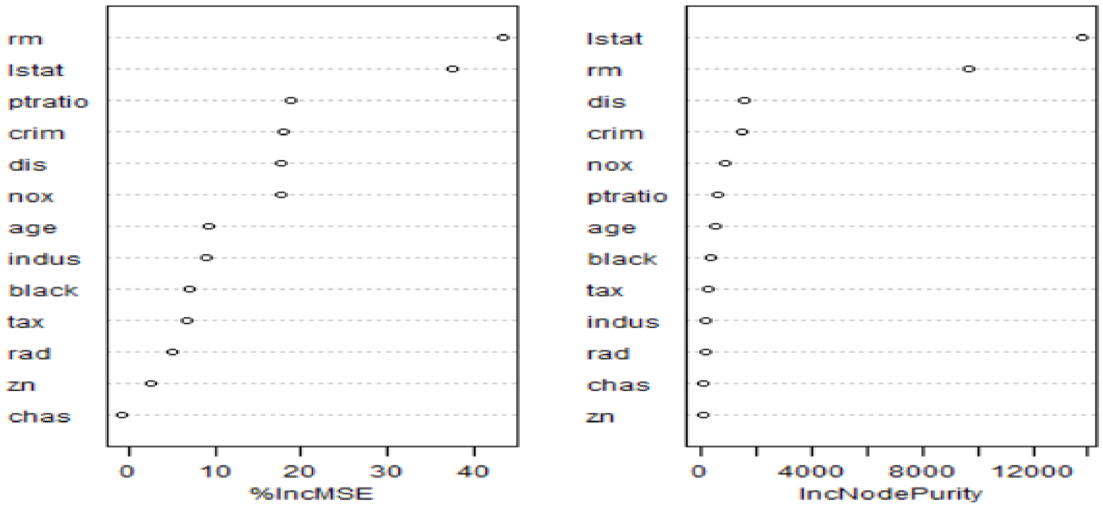
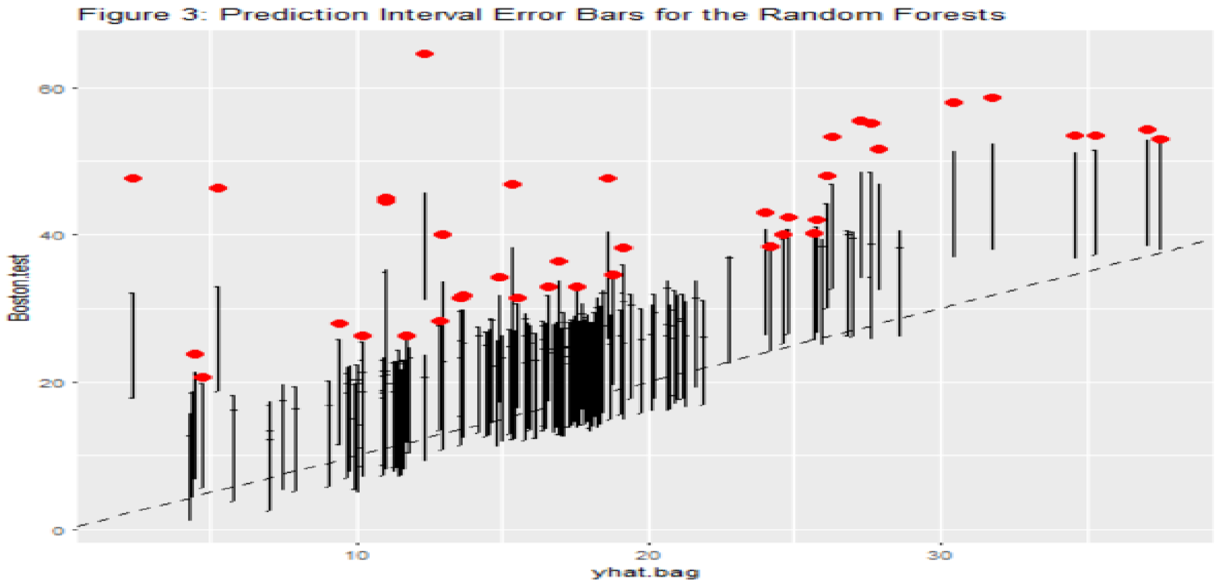


Figure 2 shows Variable Importance of Variable Importance that have the greatest impact in the prediction of the Training Set. The %IncMSE measures the increase in mse of predictions(estimated with out-of-bag-CV) as a result of variables are permuted (or randomly shuffled).

The plot lists lstat and rm as the important variables that have the higher %IncMSE values, in decreasing order of importance. The IncNodePurity plot shows the loss function of the same variables (lstat and rm) from which the best splits are chosen.

Figure 3 shows a plot of the Prediction Interval Error Bars of the Boston test set data against the fitted data from the predict function (yhat.bag) described by Rickert (2016) .

Figure 3: Prediction Interval Error Bars of the Boston Housing Test and Fitted Predictions from SAS Enterprise Guide



Notice that all of the data points were above the dashed line of equality. The Random Forests have different confidence levels for all of its estimates, 40 of the 152 estimates (in red) fall above the upper prediction limits.

Figure 4 is a replica of Figure 3 using the SGPLOT Procedure on the predlim3 SAS dataset written from the R dataframe.

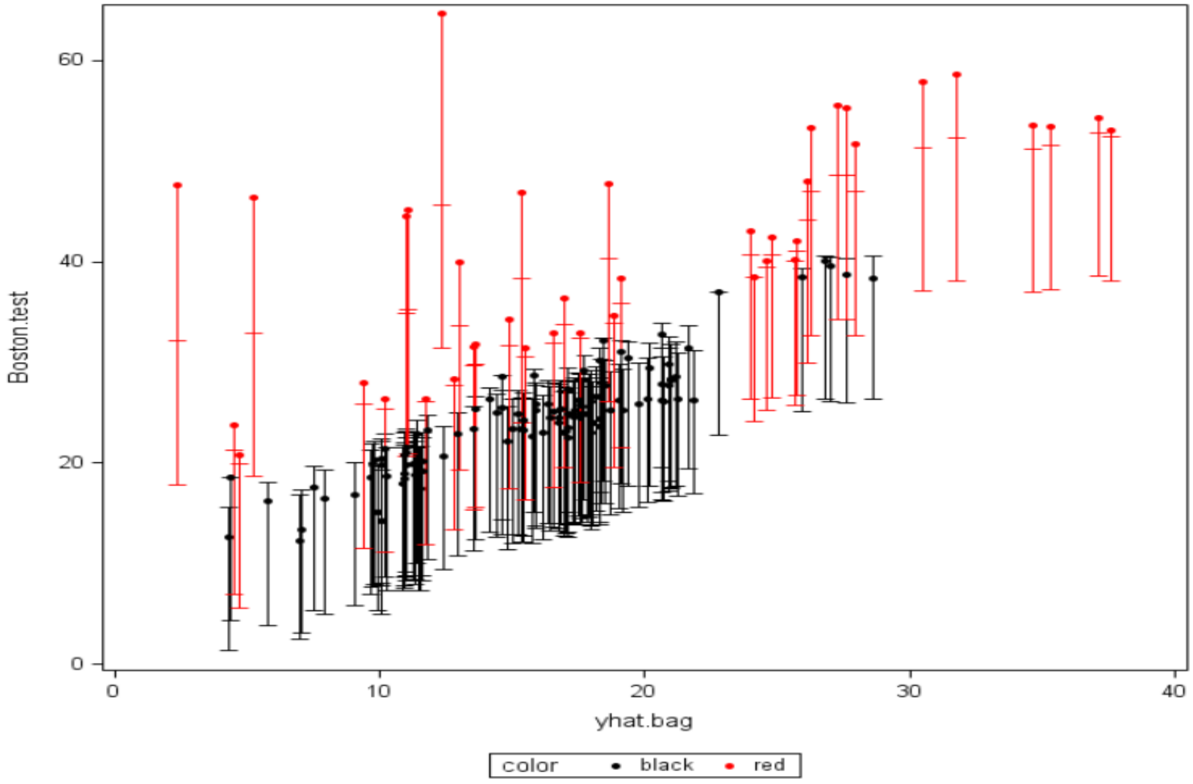
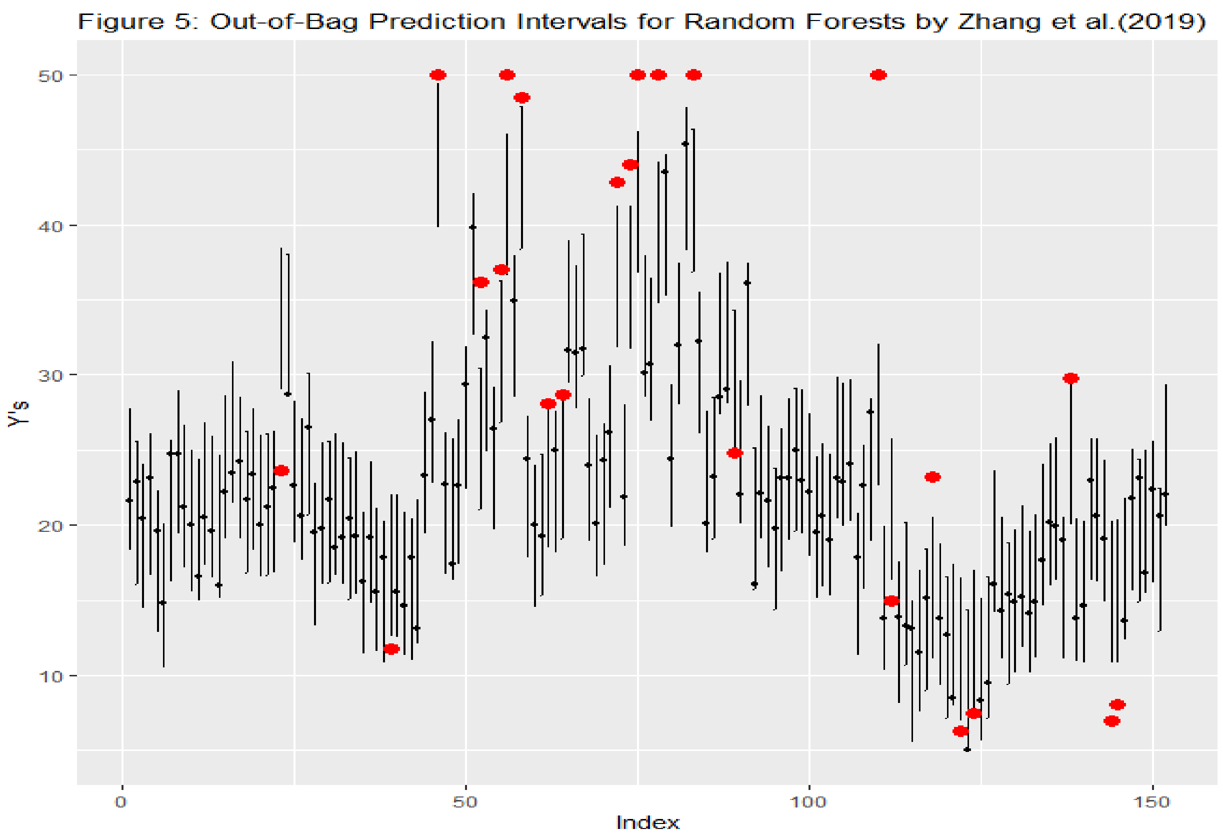


Figure 4: Prediction Limit Plot of Test Medv (Y=Boston,test) by Random Forest Medv (X=yhat.bag) with Lower and Upper Errors from the SGPLOT Procedure

Figure 5 shows the Out-of-Bag prediction intervals for the Random forests using H. Zhang's rfinterval Package with outlying Y's (medv's) from the test data in red. The prediction accuracy of the oob trees using Zhang et al. (2019) approach is 0.8487 or about 85% (1 – 23/152 = 0.8486 where 23 of the red outlying y's from the 152 observations of the test data). Figure 5 differs from Figure 3 because of the different formulas for computing the prediction intervals and the x-axis values for both plots.



CONCLUSION

Combining R with base SAS gives users enhanced analytics capabilities without the sole reliance of SAS/IML integration.

REFERENCES

Wei, X. (2012), "%PROC_R: A SAS Macro that Enables Native R Programming in the Base SAS Environment" *Journal of Statistical Software*, v.46, Code Snippet 2, https://www.jstatsoft.org/article/view/v046c02 .

Bettinger, R. (2016), "%SUBMIT R: A SAS(R) Macro to Interface SAS and R", *SESUG 2016 Conference Proceedings*, https://analytics.ncsu.edu/sesug/2016/AD-118_Final_PDF.pdf.

Gilsen, B. (2018), "Using the R interface SAS® to Call R Functions and Transfer Data", *SESUG 2018 Conference Proceedings*, https://www.lexjansen.com/sesug/2018/SESUG2018_Paper-119_Final_PDF.pdf.

Hall, P. (2015), “Tip: Open Source Integration Using the Base SAS Java Object”, https://communities.sas.com/t5/SAS-Communities-Library/Tip-Open-Source-Integration-Using-the-Base-SAS-Java-Object/ta-p/223697.

Revolution Analytics (2015), “Call R and Python from base SAS”, https://blog.revolutionanalytics.com/2015/05/call-r-and-python-from-base-sas.html .

Rickert, J. (2016), “Confidence Intervals for Random Forests”, https://blog.revolutionanalytics.com/2016/03/confidence-intervals-for-random-forest.html.

Wujek, B (2015), “Open Source Integration Using the Base SAS Java Object”, https://github.com/sassoftware/enlighten-integration/tree/master/SAS_Base_OpenSrcIntegration.

Walia, A.S. (2017), “Random Forests in R”, https://www.r-bloggers.com/random-forests-in-r/.

Zhang, H. (2019), “rfinterval: Predictive Inference Random Forest”, https://cran.r-project.org/web/packages/rfinterval/index.html

Zhang, H., Zimmerman, J., Nettleton, D., and Norman, D.J. (2019), “Random Forest Prediction Intervals,” June, 2019, *The American Statistician*, https://doi.org/10.1080/00031305.2019.1585288

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Melvin Alexander
Phone: (410) 458-7129
E-mail: melvin.alexander@verizon.net