STAT652 Project Report

Ritesh Patel, 301372956

December 01, 2018

1 Introduction

This project aims to study the cognitive health status of an individual using the data provided by Canadian Community Health Survey (CCHS), in particular we try to predict cognition condition of an individual. The first part of project uses "HUI" dataset to predict "HUIDCOG" cognition category (6 levels) of a person from a given set of 8 health utility indexes. In the second part we use "HStrain" dataset to predict "HUIDHSI" score a derived variable that provides a description of an individual's overall functional health, based on 8 HUI derived variables mentioned above. The version of the index used in CCHS is adapted from the HUI Mark 3 (HUI3).

2 Data

HUI dataset contains 20000 observations with 8 qualitative predictor variables. On the other hand, HStrain dataset contains 10000 observations without 'NOT STATED' values with 590 explanatory variables. The data dimension for HStrain needs to be reduced before we can apply any statistical methods to predict 'HUIDHSI' score.

3 Methods

3.1 Data Preprocessing done for HUI and HStrain datasets

Applying data cleaning and dimensionality reduction methods on a dataset enhances the quality of data, reduces storage space requirements, decreases the time required to process data, also because of lower dimension of feature set a simpler model would be obtained which will avoid overfitting, all these factors combined together work towards building a parsimonious model.

For HUI dataset we cleaned the data by dropping the rows containing 'NOT STATED' column values. Also, since there are only a small number of rows that have 'NOT STATED' column values, dropping them didn't show a significant reduction in dataset size. Before diving into prediction, we split the data into train and validation for HUI dataset. We randomly choose 70 % of observations as training data and the rest as validation data.

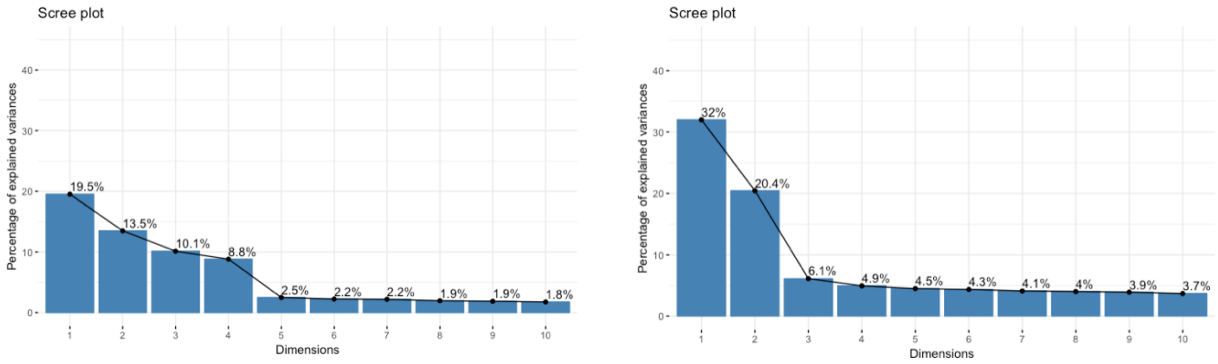
For HStrain dataset since the data didn't have missing values aka. 'NOT STATED' values. Since we have 590 explanatory variables, before we apply any statistical learning methods, we need to reduce the data dimensionality as there many redundant/duplicate features. For this, we first manually understood different categories of explanatory variables. Next, we decided to remove predictors that we think are not important for predicting HUIDHSI. For example, variables that starts with ADM are to do with administration and doesn't tell much about an individual. We are left with 37 categories of variables as seen in below image.

```
ADL ALC CAG CCC CGE CIH CR1 CR2 DHH DPS DS2 EDU FAL GEN GEO HC2 HUI HUP HWT IAL IN2 LBF LON MED NUR OH3 OWN PAZ RET RPL
  4   5  45  31   8  27  34  19   9  33   4   2  15  10   2  19   1   1   5   6   8  19   4  33  12  27   2  49  32  19
SDC SLP SLS SMK SPA SSA TRA
  6   1   6  21  24  25  19
```

After reading the "CCHS_HA_Derived_variables" document provided with the dataset we observed that each category had derived variables that summarize several variables in that particular category. For example, ("ADL_04A", "ADL_06A","ADLDTOI") are summarized by "**ADLDCLS**" in ADL category, (CAGFPAS, CAG_07C, CAG_07D) are summarized by "**CAGDFAP**" in CAG category, etc. So, we decided to keep only the derived variables.

To extract all the derived variables using the special property that all derived variables have i.e. they have '**D**' as the fourth character in all derived variable names. Filtering features using this criterion, we have 41 derived variables in feature set from 591.

Additionally, there are several categories of variables that can be summarized by a consolidated summary score and do not have derived variables. For example, applying MCA on 27 CIH features, converts it to 4 dimensions and 8 CGE features to 2 dimensions which explains about 50% of the total variance for the both cases.



Explained Variance (%) vs Dimensions for CIH(Left) and CGE(Right) variables

After adding variables that where important and not covered under derived variables and removing duplicate derived variables which are more or less similar to already existing DV. We are finally left with 40 features in total which are listed below.

```
 [1] "GEND08"    "GEND09"    "GENDHDI"   "GENDMHI"   "CCCDCPD"   "HUPDPAD"   "SLSDCLS"   "PA2DSCR"   "NURDHNR"
[10] "SMKDSTY"   "ALCDTTM"   "FALDFOF"   "FALDSTA"   "ADLDCLS"   "CR2DTHC"   "CR2DFAR"   "SSADAFF"   "SSADEMO"
[19] "SSADSOC"   "SSADTNG"   "CAGDFAP"   "DPSDSF"    "LONDSCR"   "LBFDMJS"   "RETDRS"    "HUIDHSI"   "CCCF1"
[28] "FALG02"    "HC2FCOP"   "HWTGBMI"   "IN2GHH"    "MEDF1"     "SLP_02"    "SPAFPAR"   "CIH.Dim.1" "CIH.Dim.2"
[37] "CIH.Dim.3" "CIH.Dim.4" "CGE.Dim.1" "CGE.Dim.2"
```

Scaling and Centering HStrain training and test Data

Scaling and Centering both train and test data as some statistical methods depends on whether the features are scaled or not. Also results would be more comparable if features are scaled. Splitting Data into train and test, to compare our models we'll divide our 10000 observations in ratio = 70:30.

3.2 Prediction Models (Part 1)

For the first part of the project I have applied classification methods namely: GBM, SVM, Random Forests (Bagging), Random Forest without Bagging. I decided not to choose decision tree to predict HUIDCOG because they are the building blocks of Random Forests and we know trees [ISLR pg, 318] by themselves generally don't have same level of accuracy as highly sophisticated classification models like SVM, GBM do.

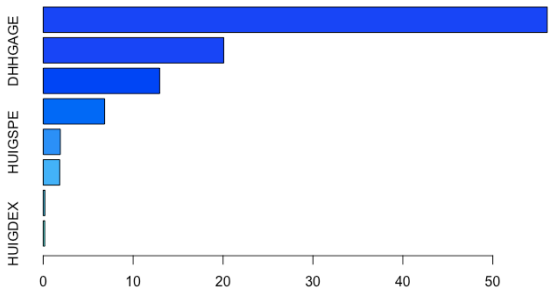
3.2.1 Gradient Boosting Model

GBM is an ensemble learning model (similar to random forest) which relies on a committee of weak classifiers (usually decision trees) for prediction. Each weak classifier C[i] is constructed sequentially i.e C[i] depends on C[i-1] where C[i] are constructed to address problems which are "hard" for C[i-1]. It can be used to solve both classification and regression problems. For classification the distribution is "multinomial" and regression its "gaussian"

Important tuning parameters in GBM:

N.trees (number of trees to fit or number of basis functions in additive expansion), I tried the multiple values between 100 and 3000 and the best value with highest accuracy I found was 1000. **Interaction.depth** (number of splits in each tree which controls the complexity of the model), I tried values between 1 and 6 and got the best results with depth=5. **Shrinkage** (a small positive number that controls the rate at which boosting learns), very small λ can require using a very large value of n.trees in order to achieve good performance. I tried typical values suggested by [Islr pg.322] values like 0.01 or 0.001. Best results were obtained using λ = 0.0001.

GBM Summary



var <fctr>	rel.inf <dbl>
HUIDEMO	56.0537855
DHHGAGE	20.0698242
HUIGMOB	12.9489229
HUIGHER	6.8318153
HUIGSPE	1.9034747
HUIGVIS	1.8350156
DHH_SEX	0.1870227
HUIGDEX	0.1701390

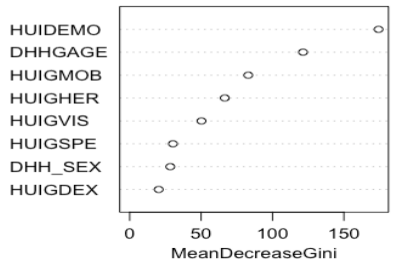
It can be interpreted from the model summary that DDH_SEX and HUIDEX have the lowest relative influence with values 0.18 and 0.17. Removing these features, I saw an improvement in validation accuracy from 0.6971146 to 0.6972853. Thus, resulting in a better and simpler model.

3.2.2 Support Vector Machines

General idea behind SVM is to define a decision boundary which maximizes the margin between the decision boundary and the classes in the feature space, where such boundary is defined by some influential data points which are called support vectors. We explored an SVM where an observation can be on the wrong side of the margin by at most cost c. We explored different kernels: linear, polynomial and radial. We find the best parameters using tune.svm(), this broke my laptop and is not worth trying. However, the best parameters obtained where kernel=”radial”,cost=100 and gamma=0.01.

3.2.3 Random Forests

Random Forests are similar to a famous ensemble technique called Bagging but have a different tweak in it. In Random Forests the idea is to decorrelate several trees which are generated by different bootstrapped samples from training data. And then we simply reduce the Variance in the Trees by averaging them. If we set **mtry** equal to the number of predictors in our case 6 we implicitly tell that bagging should be done. i.e while splitting at a node consider all the independent variables.



	MeanDecreaseAccuracy	MeanDecreaseGini
DHHGAGE	11.7688248	121.36705
DHH_SEX	3.8619633	28.42992
HUIGDEX	12.4545655	20.29583
HUIDEMO	27.4030273	174.25304
HUIGHER	16.0109329	66.58472
HUIGMOB	17.5639926	83.00369
HUIGSPE	30.7876491	30.29565
HUIGVIS	0.1058956	50.25809

RF varImpPlot also suggests that both DHH_SEX and HUIGDEX are not significant in predicting HUIDCOG. As both of these features have least mean decrease gini values, I decided to **not** select them in the feature set for prediction.

3.3 Prediction Models (Part 2)

In this part of the project we solve a regression problem to predict HUIDCOG score. Models I used are as follows:

3.3.1 Stepwise Subset Selection

- Backward Elimination: We start with the largest model and try to drop terms, down to obtain a reduced model.
- Forward Stepwise Selection: We start with smallest model or null model add terms up to some largest model.
- Inference: The MSE reported by backward elimination is 0.03513711 lesser than forward stepwise selection is 0.04188759.

3.3.2 Shrinkage Methods

This approach involves fitting a model involving all p predictors. However, the estimated coefficients are shrunken towards zero relative to the least square's estimates. This shrinkage (also known as regularization) has the effect of reducing variance. Depending on what type of shrinkage is performed, some of the coefficients may be estimated to be exactly zero. Hence, shrinkage methods can be used to perform variable selection.

- Ridge: Ridge regression is very similar to least squares, except that the coefficients in this method are estimated by minimizing a slightly different quantity.
- Lasso: Lasso is an alternative to ridge regression, the key difference here is that lasso can shrink the coefficient estimates equal to zero, while ridge can only minimize it. Therefore, we can say this lasso regression model can perform variable selection.
- Inference: We see ridge MSE=0.01492584 and lasso MSE = 0.01352717 regression. The sparse model generated by lasso is much easier to interpret than that given by ridge.

3.3.3 Random forests

As random forest was also formally introduced in the earlier section, we skip it here. We now use this model to solve a regression problem.

- Using features selected by Lasso
    - Validation MSE: 0.0149067
- Using all 40 features:
    - Validation MSE: 0.0149067
- Inference: We see ridge MSE=0.01492584 and lasso MSE = 0.01352717 regression reports almost the same MSE

3.3.4 Gradient Boosting Model

As GBM was also formally introduced in the earlier section, we skip it here. The only parameter we change is distribution, I use gaussian as this is regression problem we are solving. The best parameters achieved by fine tuning the model were **n.trees**=200, **interaction.depth** = 2. We use the same approach to tune the model parameters. There are 11 dummy variables with high influence with value greater than 1, HUPDPADPAIN. ATT..LEV.5 being the highest and all the remaining have negligible influence.

4 Results

4.1 Results (Part 1)

Method	Description	Validation Accuracy
GBM	HUIDCOG~., n.trees=1000, interaction.depth = 5, shrinkage=0.0001	69.72%
Bagging	HUIDCOG ~ ., mtry=6,importance =TRUE	69.18%
RF	HUIDCOG ~ ., mtry=sqrt(6) importance =TRUE	69.72%
SVM	HUIDCOG ~ ., kernel="radial",cost=100,gamma=0.01	69.72%
Majority (GBM,Bagg.,RF,SVM)	Ensemble {SVM,RF,Bagging,GBM}	**69.74 %**

Table 1. Accuracy on validation set

Test/Holdout Accuracy by winner model (GBM): **70.11039**

4.2 Results (Part2)

Methods	Validation Mean Squared Error
Linear Regression using Forward Stepwise Selection	0.04188759
Linear Regression using Backward Elimination	0.03513711
Ridge Regression	0.01492584
Lasso Regression	0.01352717
Random Forest with Lasso	0.01388354
Random Forest with all features	0.01490675
Gradient Boosting Machine with all features	**0.01336019**

Table 2. MSE on validation set

Test/Holdout Accuracy by winner model (GBM): **0.01130872**

Conclusions and Discussions

Various classifiers were tuned and implemented to achieve simplest and highest validation accuracy. In part one the validation accuracy obtained by SVM and GBM after fine tuning are same. But I choose the winner model which is easier to interpret and equally flexible, namely GBM [ISLR pg, 25, Figure 2.7]. In terms of future work, we could convert this to multiclass classification problem to binary classification and perform model diagnostics to check if a better model can be obtained. In part two I did the same but applied models solved a regression problem, we chose the winner model which had the least validation MSE i.e GBM again. In terms of future work, we can perform model diagnostics and apply advanced methods like xgBoost and see if the MSE degrades further. Working with various models, I experienced that as the flexibility of the model increases, we trade its interpretability.

Appendix for the project

Software Version: All analysis on this project was done using R Studio version: R version 3.5.1 OS: Mac OS v 10.14.1

Part I: Predicting HUIDCOG

Dataset Summary

```
library(dplyr)
library(forcats)
library(gbm)
library(randomForest)
library(e1071)
library(functional)
library(tidyverse)
library(dplyr)
library(FactoMineR)
library(factoextra)
library(leaps)
library(glmnet)

hui <- read.csv("hui.csv")
HUItest <- read.csv("HUItest.csv")
summary(hui)
dim(hui)
```

From the summary we see that there are no “NA” or missing values. However there are records with “NOT STATED” values.Dropping rows with “NOT STATED” Values and converting all rows to factor.

Removing DHH_SEX and HUIGDEX features as these not significant in predicting HUIDCOG after interpreting GBM and RF results with full model.

```
dataclean <- function(dat) {
    dat[dat == 'NOT STATED'] <- NA
    dat <- na.omit(dat)
    for(i in 1:ncol(dat)){
      dat[,i] <- fct_drop(dat[,i])
    }
    dat<-select(dat,-DHH_SEX,-HUIGDEX) ## added this line after interpretting
GBM and RF results
    return(dat)
}
hui<-dataclean(hui)
summary(hui)
dim(hui)

HUItest<-dataclean(HUItest)
summary(HUItest)
```

Splitting data into train and test in ratio 70:30

```
set.seed(1)
train_size = floor(0.70 * nrow(hui))
train = 1:train_size
hui.train = hui[train,]
hui.test = hui[-train,]
```

Since this is a multiclass classification problem we cannot apply logistic regression directly, also mutlinomial family is not supported by glm(). Other methods taught in class like GBM,SVM,RF,Bagging are the goto classifiers for this problem.

GBM

```
set.seed(1)
hui.boost = gbm(HUIDCOG ~ ., data=hui.train, n.trees=1000, interaction.depth
= 5, shrinkage=0.0001)
summary(hui.boost)
``

#GBM Validation accuracy

#Validation
boost.probablities = predict(hui.boost, hui.test, n.trees=1000, type="respons
e")
boost.prediction = apply(boost.probablities,1,which.max)
mean(colnames(boost.probablities)[boost.prediction] == hui.test$HUIDCOG)
```

Random Forest (bagging)

```
set.seed(1)
#bagging
randForest.bag=randomForest(factor(hui.train$HUIDCOG) ~ .,data=hui.train,subs
et=train,mtry=6,importance =TRUE)
```

Random Forest (bagging) Validation accuracy

```
randForest.bag.pred = predict(randForest.bag, hui.test)
mean(factor(randForest.bag.pred) == factor(hui.test$HUIDCOG))
```

Random Forest without bagging

```
set.seed(1)
randForest=randomForest(factor(hui.train$HUIDCOG) ~ .,data=hui.train,subset=t
rain,importance =TRUE)
varImpPlot(randForest)
importance(randForest)
```

Random Forest without bagging Validation accuracy

```
randForest.pred = predict(randForest, hui.test)
mean(randForest.pred == factor(hui.test$HUIDCOG))
```

SVM

```
set.seed(1)
#tune.out = tune(svm, HUIDCOG~., data=hui.train, kernel="radial", ranges=list
(cost=c(0.1, 1, 5, 10,100), gamma=c(0.01, 0.1, 1, 5, 10, 100)))
#summary(tune.out)
svc.fit = svm(HUIDCOG~., data=hui.train, kernel="radial",cost=100,gamma=0.01)
```

SVM Validation accuracy

```
#Validation
svm.pred = predict(svc.fit, hui.test)
mean(svm.pred == hui.test$HUIDCOG)
```

Majority voting using GBM,RandomForest(Bagging),Random Forest, SVM

```
temp<- data.frame(gbm=colnames(boost.probablities)[boost.prediction],rfbag=fa
ctor(randForest.bag.pred),rf=randForest.pred,svm=svm.pred)
ens.pred = apply(temp, 1, Compose(table,function(i) i==max(i),which.max,names
))
mean(ens.pred == hui.test$HUIDCOG)
```

Conclusion

The validation by SVM and GBM after fine tuning them are same. But I choose the winner model which is easier to interpret and equally flexible, namely GBM

Winner Model(GBM) Test accuracy

```
boost.probablities = predict(hui.boost, HUItest, n.trees=1000, type="response
")
boost.fprediction = apply(boost.probablities,1,which.max)
mean(colnames(boost.probablities)[boost.fprediction] == HUItest$HUIDCOG)
```

The winner model Test/Holdout accuracy is 0.7011039

Part II: Predicting a HUI Score with Other Variables

```
#read train and test data
hs = read.csv("HStrain.csv")
hsTest = read.csv("HStest.csv")
cn <- colnames(hs)

# We remove ADM as they are to do something with administering the survey
hs <- select(hs,-starts_with("ADM"))
hsTest<-select(hsTest,-starts_with("ADM"))

cn <- colnames(hs)
table(substr(cn,start=1,stop=3))
```

Extracting all derived variables with letter "D" at position 4 in feature name

```
cn1<- cn[substr(cn,start=4,stop=4) =='D']
length(cn1)
```

Adding variables that are important and not covered by DV's

```
cn2 <- c('CCCF1','FALG02','HC2FCOP','HWTGBMI','IN2GHH','MEDF1','NURDHNR','PA2DSCR','SLP_02','SPAFPAR')
cn <- c(cn1,cn2)
```

Removing duplicate variables which are similar to already existing features or I dont consider them to be important

```
cn <- setdiff(cn, c('CCCD901','SMKDYCS','ADLDTOI','CR1DTRE','CR2DIAR','CAGDIAR','DPSDMT','DPSDPP','DPSDWK','EDUDR04','EDUDH04','IN2DRPR','IN2DRCA','RETDWAR','LBFDPFT'))
cn
```

My choices

```
table(substr(cn,start=1,stop=3))
```

Multiple Correspondence Analysis

Creating our own summary score using PC's for CIH variables

```
res.mca <- MCA(select(hs,starts_with("CIH")),graph=FALSE)
CIHPCs <- res.mca$ind$coord[,1:4] # first 4 dimensions explains 50 % of variance
colnames(CIHPCs) <- paste("CIH",colnames(CIHPCs))

#Preparing test dataset
testRes.mca <- MCA(select(hsTest,starts_with("CIH")),graph=FALSE)
TestCIHPCs <- testRes.mca$ind$coord[,1:4]
colnames(TestCIHPCs) <- paste("CIH",colnames(TestCIHPCs))

fviz_screeplot(res.mca, addlabels = TRUE, ylim = c(0, 45))
```

Creating our own summary score using PC's for CGE variables

```
res.mca <- MCA(select(hs,starts_with("CGE")),graph=FALSE)
CGEPCs <- res.mca$ind$coord[,1:2]   # first 2 dimensions explains more than 50% of variance
colnames(CGEPCs) <- paste("CGE",colnames(CGEPCs))

#Preparing test dataset
testRes.mca <- MCA(select(hsTest,starts_with("CGE")),graph=FALSE)
TestCGEPCs <- testRes.mca$ind$coord[,1:2]
colnames(TestCGEPCs) <- paste("CGE",colnames(TestCGEPCs))
fviz_screeplot(res.mca, addlabels = TRUE, ylim = c(0, 45))
```

We select top 2 dimensions which explains about 52% of variance.

Creating Training and testing dataframe.

```
hs <- select(hs,cn)
hsTest<-select(hsTest,cn)
hsred <- data.frame(hs,CIHPCs,CGEPCs)

#print final choices of Features i.e 40 features
colnames(hsred)

#Preparing test dataset
hsTest<-data.frame(hsTest,TestCIHPCs,TestCGEPCs)
```

Scale and center both train and test data as some statistical methods depend on if the features are scaled or not. Also results would be more comparable if features are scaled.

```
#Creating dummy variables out of factors using model.matrix() and leaving out
intercept
temp <- model.matrix(HUIDHSI ~ .,data=hsred)[,-1]
X <- as.data.frame(scale(temp))
Y <- hsred$HUIDHSI

temp <- model.matrix(HUIDHSI ~ .,data=hsTest)[,-1]
XTest <- as.data.frame(scale(temp))
YTest <- hsTest$HUIDHSI
```

Splitting Data into train and test

To compare our models we'll divide our 10000 observations in ratio = 70:30

```
set.seed(123)
n.train <- 7000
n.test <- nrow(hsTest)

train <- sample(1:nrow(hs),replace=FALSE,size=n.train)
test <- sample(1:n.test,replace=FALSE,size=n.test)

#returns indexes of sampled 7000 rows.
X.train <- X[train,]; Y.train <- Y[train]
X.test <- X[-train,]; Y.test <- Y[-train]
X.hstest <- XTest[test,]
Y.hstest <- YTest[test]
```

Stepwise Backward Elimination

```
mod.bwd <- regsubsets(X.train,Y.train,method="backward")
best_subet <- summary(mod.bwd)

#best_subet$which
```

```r
#plot(mod.fwd,scale ='Cp')
best_cp <- which.min(best_subet$cp)
best_cp

best_bic_bwd <-which.min(best_subet$bic)
best_bic_bwd

new_mod <- coefficients(mod.bwd, id=9)[-1]
```

Calculating MSE for features reported by Backward elimination

```r
cols<- best_subet$which[best_bic_bwd,-1]
Xred <- as.matrix(X.test[,cols])
pred.test <- cbind(1,Xred) %*%  coef(mod.bwd,id=9)  #multiplying weights/coef with features(X)
mean((Y.test - pred.test)^2)
```

Stepwise Forward Subset selection

```r
mod.fwd <- regsubsets(X.train,Y.train,nvmax=40,method="forward")
best_subet <- summary(mod.fwd)

#best_subet$which

#plot(mod.fwd,scale ='Cp')
best_cp_fwd <- which.min(best_subet$cp)
best_cp_fwd

best_bic_fwd <-which.min(best_subet$bic)
best_bic_fwd

#since we know BIC penalizes model complexity more heavily than Mellow CP we will choose coefficients reported by BIC
coefficients(mod.fwd, id=28)[-1]
```

Calculating MSE for features reported by Forward stepwise selection.

```r
cols<- best_subet$which[best_bic_fwd,-1]
Xred <- as.matrix(X.test[,cols])
pred.test <- cbind(1,Xred) %*%  coef(mod.fwd,id=28)  #multiplying weights/coef with features(X)
mean((Y.test - pred.test)^2)
```

Inference: We see the MSE reported by Backward elimination is lesser than Forward subset selection

Ridge Regression

```r
set.seed(123)
lambdas <- 10^{seq(from=-3,to=5,length=100)}
cv.lafit <- cv.glmnet(as.matrix(X.train),Y.train,alpha=0,lambda=lambdas)
```

```
plot(cv.lafit)
la.best.lam <- cv.lafit$lambda.1se

rr <- glmnet(as.matrix(X.train),Y.train,alpha=0,lambda=la.best.lam)
```

Calculate MSE for Ridge Regression

```
pred.test <- predict(rr,as.matrix(X.test))
mean((Y.test-pred.test)^2)
```

Lasso Regression

```
set.seed(123)
lambdas <- 10^{seq(from=-3,to=5,length=100)}
cv.lafit <- cv.glmnet(as.matrix(X.train),Y.train,alpha=1,lambda=lambdas)

plot(cv.lafit)
la.best.lam <- cv.lafit$lambda.1se

lr <- glmnet(as.matrix(X.train),Y.train,alpha=1,lambda=la.best.lam)
```

Calculate MSE for Lasso Regression

```
pred.test <- predict(lr,as.matrix(X.test))
mean((Y.test-pred.test)^2)
```

Inference: We see ridge MSE=0.01492584 and lasso MSE = 0.01352717 regression reports almost the same MSE. I choose features selected by lasso considering the lower MSE

```
nonz <- (as.numeric(coef(lr))!=0)[-1]
hsred2.train <- data.frame(HUIDHSI=Y.train,X.train[,nonz])
```

RandomForest using features selected by Lasso

```
set.seed(123)
rf<-randomForest(X.train[,nonz],y=Y.train,xtest=X.test[,nonz],ytest=Y.test,nt
ree=200,mtry=sqrt(ncol(X.train[,nonz])),importance=TRUE )

pred.test<-rf$test$predicted
mean((Y.test - pred.test)^2)
```

RandomForest using all 40 features selected after dimensionality reduction

```
set.seed(123)
bb <- randomForest(X.train,y=Y.train,xtest=X.test,ytest = Y.test,ntree=200,mt
ry=sqrt(ncol(X.train)),importance=TRUE)
varImpPlot(bb,type=1)

pred.test<-bb$test$predicted
mean((Y.test - pred.test)^2)
```

Gradient Boosting using all 40 features selected initially

```
set.seed(123)
hstrain<- data.frame(HUIDHSI=Y.train,X.train)
boost<- gbm(HUIDHSI ~.,data=hstrain,n.trees=200,interaction.depth = 2,distrib
ution = "gaussian")
summary(boost)

hstest<- data.frame(HUIDHSI=Y.test,X.test)
pred.test<- predict(boost,newdata=hstest,n.trees=200,type="response")
mean((Y.test-pred.test)^2)
```

Conclusion

The winner model of all is GBM with lowest validation MSE = 0.0133

```
hsTest<- data.frame(HUIDHSI=Y.hstest,X.hstest)
pred.test<- predict(boost,newdata=hsTest,n.trees=200,type="response")
mean((Y.hstest-pred.test)^2)
```

The test MSE obtained using the GBM is 0.01130872