DSA4211 Report

Aiman Aminuddin

November 2021

Executive Summary

	LASSO	**Elastic Net**	**PCA**	**PLS**	**XgBoost**
Test MSE	24.25108	24.28296	40.33388	25.82683	25.14059
Test R^2	0.607644	0.6071282	0.347442	0.5821501	0.5932528

Before making any analysis, we realize that one variable: X_{55} with more than 80% of the values missing. We decide to omit X_{55} from our analysis as using only 20% of the values and replacing the NAs with the mean of remaining values is not representative of the distribution of X_{55}. We split the data into a training set to build our models and test set to evaluate the predictive accuracy of said models as well. In order to find the best model, we decide to implement a broad range of approaches to cover all grounds.

The table above shows a summary of the test performance of the different models implemented. It seems that LASSO is the best performing model on test data. This is surprising for several reasons. Theoretically, we would expect Elastic Net to perform better than LASSO as it uses both the l_1 norm and l_2 norm to regularizes the regression model. Having both penalty terms, we have a balance between LASSO and Ridge Regression and perform both variable selection and coefficient reduction (but not exactly 0). On a side note, it is almost as good as LASSO as the difference between test R^2 value is about 10^{-5}. Additionally, XGBoost is a well-regarded as a state-of-the-art algorithm in training models with high accuracy. Many winning entries of Machine Learning Competitions such as Kaggle utilised Xgboost. However here, it is beaten by LASSO.

Additionally, Principal Component Analysis (PCA) is a well-regarded dimension reduction method that takes a large set of predictors and map this set to a smaller set of Principal Components that tries to explain most of the variance in the data set. The idea is that the first M components is sufficient to explain most of the variability in the data set. Here, PCA failed to do so. Using 10-fold Cross-Validation, R suggested that the model with 99 components reduces the MSE the most during training. Clearly, using 99 principal components would lead to over fitting. Thus, it comes to no surprise that the PCA model perform horribly on test data. What worse is that vanilla linear regression model outperforms the PCA model on test data which is surprising. However, this remedied using Partial Least Squares (PLS). Partial Least Squares find directions that help explains both the response and predictors. This is probably why PLS is almost as good as LASSO on test data.

Personally, i find that using LASSO or Elastic Net would be the better strategy in reducing the dimensions of a data set. Not only would they perform moderately well in terms of prediction accuracy on test data, the outputted model is at least interpretable. If the predictors are standardized, we are able to see the relative effect of each predictor on the response. Additionally, the other approaches feels like using a black box approach as we are not able to interpret the relative effect each of the predictor have on a response. Furthermore, having a exact form is nice especially if the data given in a real world setting. Having an exact functional form, would allow people to gain insights the variables that have a greater impact on the response and this could be useful in policy making, industrial process,etc.

DSA4211 Project

Aiman Aminuddin

November 2021

Data Set

Uploading CSV to R

```
Data <- read.csv(file.choose(),header = T)
nrow(Data) # 1000 rows
dim(Data) # 1000 rows and 101 columns
names(Data) # 1 Response Variable Y and 100 predictors
```

There are 1000 observations with 1 response variable and 100 predictors: $X_1, \ldots, X_{100}$.

Data Preprocessing

```
func1 <- function(x){
sum(is.na(x))}
apply(Data,2,func1)  # check if there are NAs in variables

   Y   X1   X2   X3   X4   X5   X6   X7   X8   X9  X10  X11  X12  X13  X14  X15
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 X16  X17  X18  X19  X20  X21  X22  X23  X24  X25  X26  X27  X28  X29  X30  X31
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 X32  X33  X34  X35  X36  X37  X38  X39  X40  X41  X42  X43  X44  X45  X46  X47
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 X48  X49  X50  X51  X52  X53  X54  X55  X56  X57  X58  X59  X60  X61  X62  X63
   0    0    0    0    0    0    0  813    0    0    0    0    0    0    0    0
 X64  X65  X66  X67  X68  X69  X70  X71  X72  X73  X74  X75  X76  X77  X78  X79
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 X80  X81  X82  X83  X84  X85  X86  X87  X88  X89  X90  X91  X92  X93  X94  X95
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 X96  X97  X98  X99 X100
   0    0    0    0    0


Data <- Data[,-56] # Remove X55
```

Using func1 function, we realized that there only one variable with NAs: X_{55}. Additionally, about 81.3% of the values in X_{55} are missing (NAs). We can either remove X_{55} from our analysis or replace NAs with the mean of remaining values. However, only 20% of the data is available for X_{55}. Thus, we decided to remove X_{55} from our analysis. Choosing the latter may result in misrepresentation of the distribution of values of X_{55} as we are only using 20% of the values given.

Currently, we have 99 predictors and 1000 observations. We want to train a regression model from Data to predict values from an independent data set with $m = 10000$. To improve prediction accuracy of our model, we should perform variable selection to reduce the number of predictors used in our model. This is because having many predictors increases the flexibility of our model and result in overfitting. While having a more flexible model improves the fit of model onto the training data. Test RSS will be significantly increase from overfitting. Alternatively, we can implement dimensional reduction methods instead that maps our large set of predictors to a smaller set of predictors and used them in regression to improve prediction accuracy. Doing so, reduces the flexibility of the model as well and reduce the risk of overfitting.

Splitting Data Set into Training and Test Set

```
set.seed(4211) # ensures reproducible results
x <- model.matrix(Y~.,Data)[,-1]
ind <- sample(2,nrow(x),replace = TRUE,prob = c(0.8,0.2))
# Split into training,test set
x.train <- x[ind == 1,]
x.test <- x[ind == 2,]
y.train <- Data$Y[ind == 1]
y.test <- Data$Y[ind == 2]

nrow(x.train)
[1] 799
nrow(x.test)
[1] 201
```

We split Data into a Training Set (x.train,y.train) and Test Set (x.test,y.test). We shall first build a few models from the Training Set and evaluate the performance of these models using the Test Set. The best model will be the one with the lowest MSE and highest R^2 value. Doing so, would give a sensing how well our best model will perform in the independent data set.

LASSO Regression

LASSO performs variable selection by

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \|\beta\|_1$$

By having the penalty term $\lambda\|\beta\|_1$, LASSO will shrink the coefficients estimates by a constant amount. However, unlike Ridge Regression (which uses l_2 penalty), the l_1 penalty forces some of the estimates to be exactly 0. Thus, some of the variables are omitted from model (essentially variable selection) and reduces the flexibility of the model. Additonally, note that OLS estimates tend to have larger variance and small bias. In this setting, we should reduces the variance by increasing bias to reduce the risk of overfitting. LASSO does this and doing so,allows for sparse models to be generated instead. Note that when $\lambda = 0$, we recover the Least Squares Regression.

LASSO Coefficient Plots

```
library(glmnet)
grid <- 10^seq(10,-2,length = 100)
# having a large range covers the full range of possible models generated from null model to
# least squares fit and choosing the best models from these many models
lasso.mod <- glmnet(x.train,y.train,alpha = 1,lambda = grid)
plot(lasso.mod)
```

Note that an alternative representation of the problem that LASSO solves is

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_p x_{ij} \right)^2 \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq s$$

If s is chosen to be small i.e. arbitrarily close to 0, we would expect many coefficient to shrink to exactly 0 as l_1 norm is close to 0. However, by allowing s to increase, there will be lesser restriction on $\|\beta\|_1$ allowing the coefficients of predictor to increase to their least squares estimates.

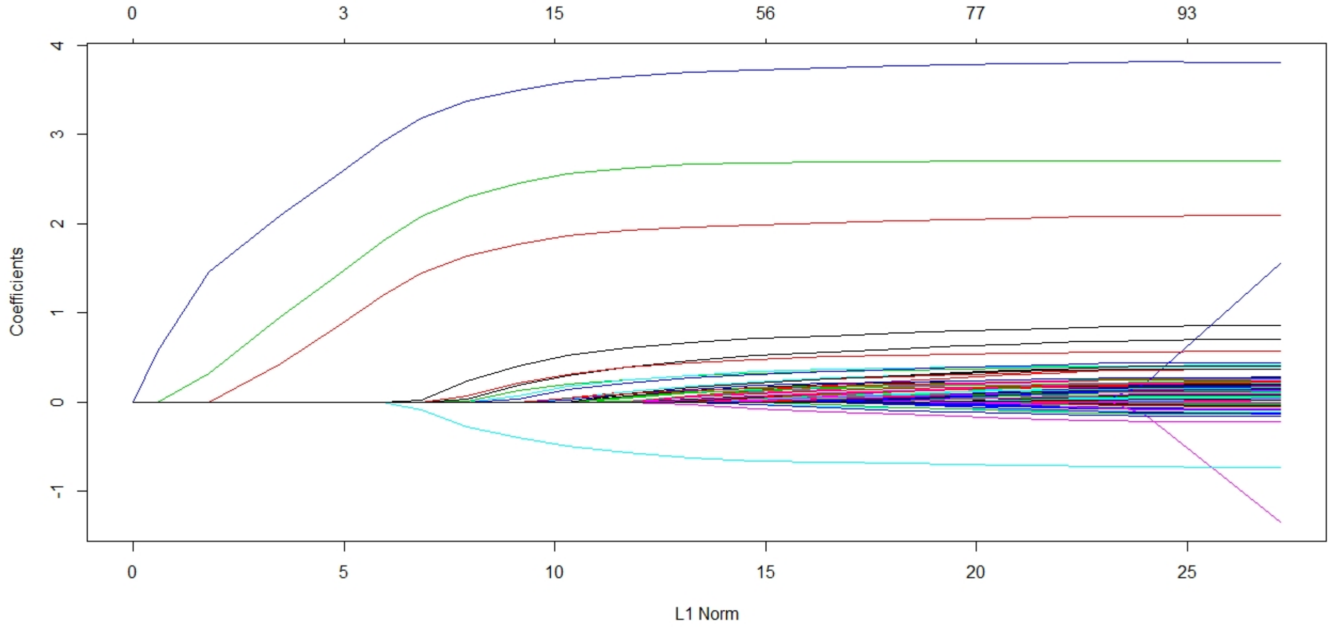

Figure 1: LASSO Coefficient of Plot

From Figure 1, the colours represent the coefficients of the predictors. Depending on the choice of s (which translate to the l_1 norm), there will be predictors whose coefficients will shrink to exactly 0 resulting only a subset of predictors to be included in the model.

Plotting Cross-Validation Error (CV) as a Function of λ

```
cv.out <- cv.glmnet(x.train,y.train,alpha = 1)
plot(cv.out)
```

From Figure 2, we plot CV Error as a function of λ using the grid values of λ. Note that cv.out uses 10-fold cross-validation by default. Additionally, the glmnet() function by default will standardized the variables inputted so that they are on the same scale for ease of analysis. We want to find λ^* that is the global minimizer of CV Error.

Extracting Optimal λ

```
bestlam <- cv.out$lambda.min
bestlam
[1] 0.1885761
```

We can see that $\lambda^* = 0.1885761$.

Calculating Test MSE and R^2 of Best LASSO Model

```
lasso.pred <- predict(lasso.mod,s = bestlam,newx= x.test)
# Calculating test MSE
mean((lasso.pred-y.test)^2)
[1] 24.25108
# Computing R^2
Rsquared <- function(x,y){
1-sum((x-y)^2)/sum((y-mean(y))^2)
}
```

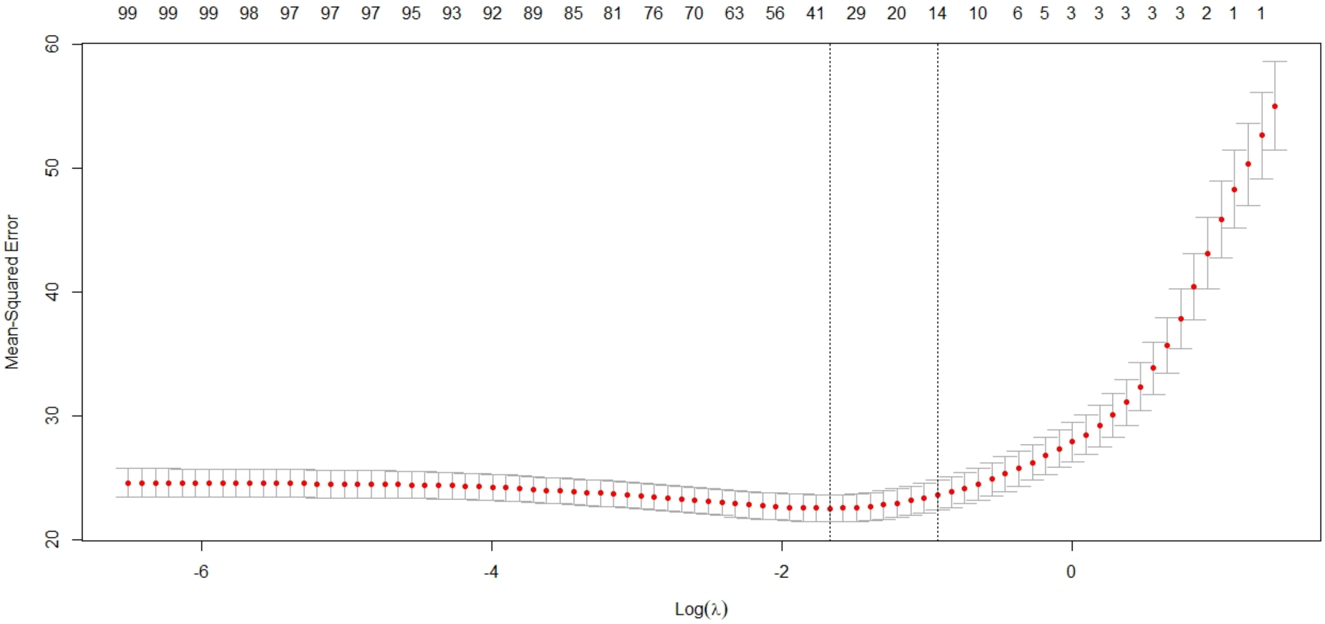

Figure 2: LASSO's CV Error as Function of λ

```
Rsquared(lasso.pred,y.test)
[1] 0.607644
```

On test data, LASSO has a MSE of 24.25108 and R^2 value of 0.607644. We can also compute the test MSE and R^2 of Least Squares Fit model to evaluate LASSO performance on test data.

MSE and R^2 values of Least Squares Fit

```
# Test MSE and R^2 of Least Squares Fit
linear.pred <- predict(lasso.mod,s = 0,newx = x.test)
mean((linear.pred - y.test)^2)
[1] 25.43509
Rsquared(linear.pred,y.test)
[1] 0.5884881
```

It seems that LASSO is slightly better than Least Sqaures in terms of its test MSE and R^2 value.

Extracting the Coefficients of the LASSO Model

```
output <- glmnet(x,Data$Y,alpha = 1,lambda = grid)
lasso.coef <- predict(output,type = "coefficients",s=bestlam)[1:100,]
lasso.coef[lasso.coef!=0]

  (Intercept)            X1            X7            X8            X9 
 0.8096571959  0.0261793966  0.6733397845  1.9377669710  2.6624478746 
          X10           X20           X22           X23           X26 
 3.8599932239  0.1144885555 -0.0005448235  0.0093351488  0.0117606579 
          X27           X28           X31           X33           X34 
 0.0149140655 -0.0032716856  0.0233850438  0.0480531152  0.0656129865 
          X35           X38           X39           X40           X43 
 0.0734976362  0.0276421016  0.1328009169  0.0069146748  0.1990522916 
          X44           X50           X51           X56           X60 
 0.0132704205  0.1000323688  0.0742940045  0.0015127118 -0.0604258734 
          X62           X68           X69           X70           X71 
 0.1063943886  0.2583905989  0.0042553768  0.4305858996  0.5828561665
          X72           X74           X78           X80           X83
-0.0015811229  0.1044664155  0.0331283008 -0.6610099225  0.0282436918
          X84           X85           X86           X87           X88
 0.0155329768  0.1009850157  0.2420389856  0.0023711004  0.1370035955
          X91           X94           X95           X97           X99
 0.3700325776  0.2182178429  0.1348342967  0.0117315379  0.0010943022
```

It seems that 44 predictors are included in the model with 55 variables being omitted i.e. coefficients have been shrunk to exactly 0.

Elastic Net Regression

Similar to LASSO, Elastic Net minimizes the residual sum of squares with added penalty. However, Elastic Net uses both the l_1, l_2 norm. Thus, it uses the penalty terms from LASSO and Ridge Regression to regularize the input model. Mathematically, Elastic Net solves:

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \left[(1-\alpha) \frac{\|\beta\|_2^2}{2} + \alpha \|\beta\|_1 \right]$$

where $0 \leq \alpha \leq 1$. As Elastic Net uses both penalty terms from LASSO and Ridge Regression, it is a compromise between LASSO and Ridge. Doing so, could potentially allow Elastic Net to reap the advantages of both methods. For example, LASSO performs variable selection unlike Ridge Regression creating sparse models by shrinking some β_j to exactly 0. While Ridge Regression does not shrink any β_j to exactly 0 (does not perform variable selection), it is a known fact that the variance from Ridge Regression is slightly lower than LASSO and the minimum MSE of Ridge is smaller than that of LASSO. With such advantages from both methods, Thus, Elastic Net take advantages of these strengths of both regularization methods. Note that setting $\alpha = 0$ or $\alpha = 1$, we recover Ridge Regression and LASSO respectively.

Plotting Coefficient Plots for different values of α

```r
alpha0 <- glmnet(x.train,y.train,alpha = 0,lambda = grid)
alpha1 <- glmnet(x.train,y.train,alpha = 0.1,lambda = grid)
alpha2 <- glmnet(x.train,y.train,alpha = 0.2,lambda = grid)
alpha3 <- glmnet(x.train,y.train,alpha = 0.3,lambda = grid)
alpha4 <- glmnet(x.train,y.train,alpha = 0.4,lambda = grid)
alpha5 <- glmnet(x.train,y.train,alpha = 0.5,lambda = grid)
alpha6 <- glmnet(x.train,y.train,alpha = 0.6,lambda = grid)
alpha7 <- glmnet(x.train,y.train,alpha = 0.7,lambda = grid)
alpha8 <- glmnet(x.train,y.train,alpha = 0.8,lambda = grid)
alpha9 <- glmnet(x.train,y.train,alpha = 0.9,lambda = grid)

par(mfrow = c(2,5))
plot(alpha0,sub = "alpha = 0")
plot(alpha1,sub = "alpha = 0.1")
plot(alpha2,sub = "alpha = 0.2")
plot(alpha3,sub = "alpha = 0.3")
plot(alpha4,sub = "alpha = 0.4")
plot(alpha5,sub = "alpha = 0.5")
plot(alpha6,sub = "alpha = 0.6")
plot(alpha7,sub = "alpha = 0.7")
plot(alpha8,sub = "alpha = 0.8")
plot(alpha9,sub = "alpha = 0.9")
```

From Figure 3, it is easy to see that Elastic Net like LASSO does variable selection for different values of α.

Plotting CV as a Function of λ for each α

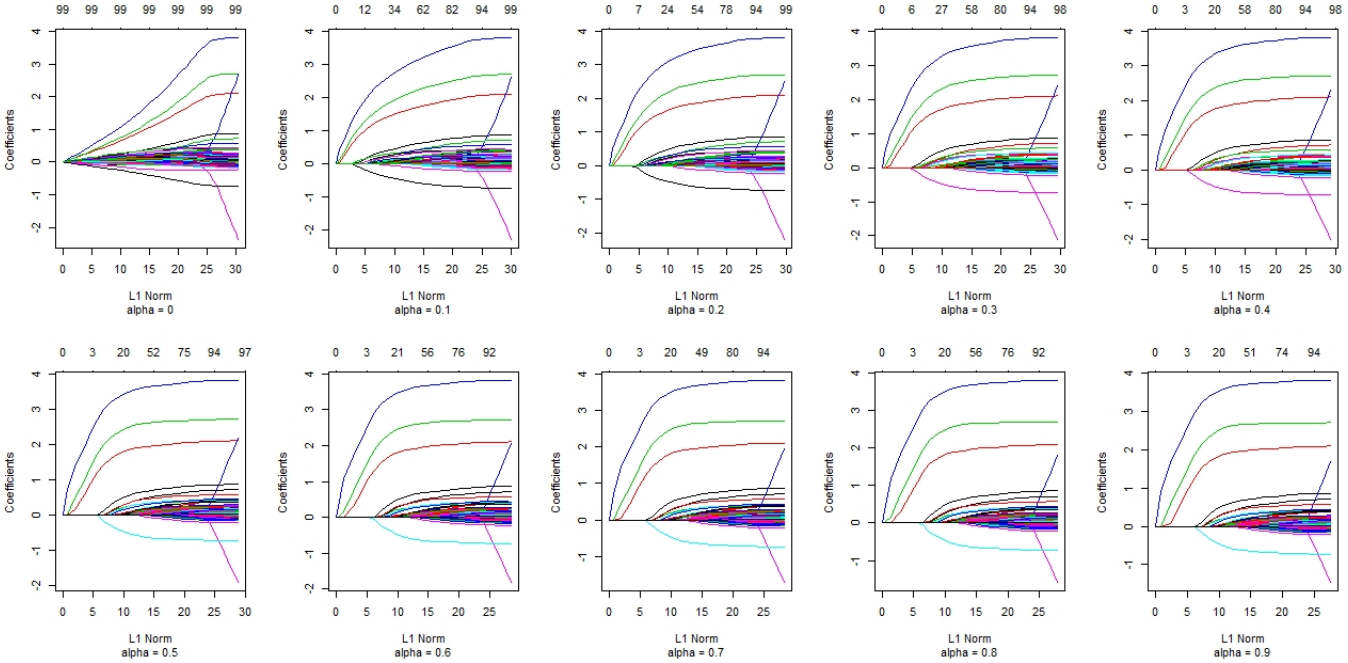

Figure 3: Elastic Net Coefficient Plots for different values of α

```
cv.out0 <- cv.glmnet(x.train,y.train,alpha = 0)
cv.out1 <- cv.glmnet(x.train,y.train,alpha = 0.1)
cv.out2 <- cv.glmnet(x.train,y.train,alpha = 0.2)
cv.out3 <- cv.glmnet(x.train,y.train,alpha = 0.3)
cv.out4 <- cv.glmnet(x.train,y.train,alpha = 0.4)
cv.out5 <- cv.glmnet(x.train,y.train,alpha = 0.5)
cv.out6 <- cv.glmnet(x.train,y.train,alpha = 0.6)
cv.out7 <- cv.glmnet(x.train,y.train,alpha = 0.7)
cv.out8 <- cv.glmnet(x.train,y.train,alpha = 0.8)
cv.out9 <- cv.glmnet(x.train,y.train,alpha = 0.9)
par(mfrow=c(2,5))
plot(cv.out0,sub = "alpha = 0")
plot(cv.out1,sub = "alpha = 0.1")
plot(cv.out2,sub = "alpha = 0.2")
plot(cv.out3,sub = "alpha = 0.3")
plot(cv.out4,sub = "alpha = 0.4")
plot(cv.out5,sub = "alpha = 0.5")
plot(cv.out6,sub = "alpha = 0.6")
plot(cv.out7,sub = "alpha = 0.7")
plot(cv.out8,sub = "alpha = 0.8")
plot(cv.out9,sub = "alpha = 0.9")
```

From Figure 4, it easy to see that for each alpha there exists a λ that is a global minimizer of CV Error. We then extract these λ values.

Extracting optimal λ^* values for each α

```
result <- data.frame(alpha = 0,lambda = cv.out0$lambda.min)
result <- rbind(result,data.frame(alpha = 0.1,lambda = cv.out1$lambda.min))
result <- rbind(result,data.frame(alpha = 0.2,lambda = cv.out2$lambda.min))
result <- rbind(result,data.frame(alpha = 0.3,lambda = cv.out3$lambda.min))
result <- rbind(result,data.frame(alpha = 0.4,lambda = cv.out4$lambda.min))
result <- rbind(result,data.frame(alpha = 0.5,lambda = cv.out5$lambda.min))
result <- rbind(result,data.frame(alpha = 0.6,lambda = cv.out6$lambda.min))
result <- rbind(result,data.frame(alpha = 0.7,lambda = cv.out7$lambda.min))
```

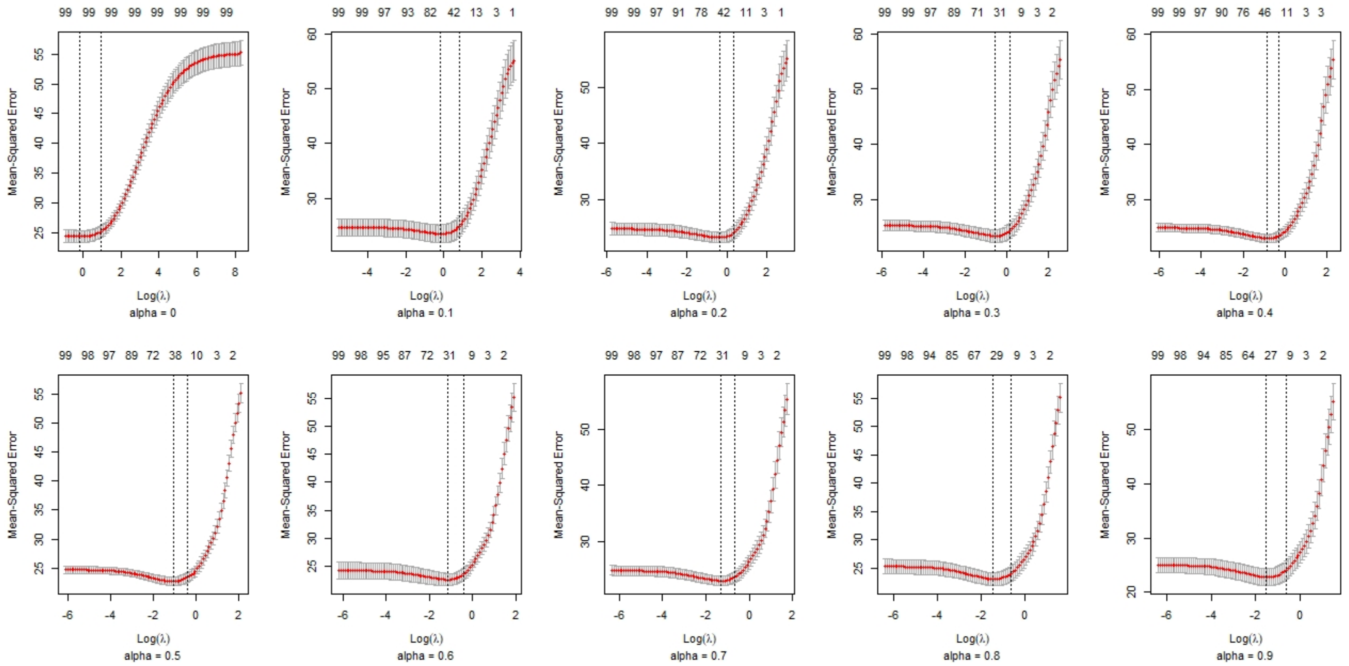

Figure 4: CV Error as a function of λ for each alpha value

```
result <- rbind(result,data.frame(alpha = 0.8,lambda = cv.out8$lambda.min))
result <- rbind(result,data.frame(alpha = 0.9,lambda = cv.out9$lambda.min))
result

   alpha    lambda
1    0.0 0.8551698
2    0.1 0.8163010
3    0.2 0.7132546
4    0.3 0.5727451
5    0.4 0.4295588
6    0.5 0.3436471
7    0.6 0.3142935
8    0.7 0.2693945
9    0.8 0.2357201
10   0.9 0.2095290
```

It seems as α increases in value, λ^* decreases in value.

Evaluating Test MSE and R^2 value for different α values

```
Elastic.pred0 <- predict(alpha0,s = cv.out0$lambda.min,newx = x.test)
Elastic.pred1 <- predict(alpha1,s = cv.out1$lambda.min,newx = x.test)
Elastic.pred2 <- predict(alpha2,s = cv.out2$lambda.min,newx = x.test)
Elastic.pred3 <- predict(alpha3,s = cv.out3$lambda.min,newx = x.test)
Elastic.pred4 <- predict(alpha4,s = cv.out4$lambda.min,newx = x.test)
Elastic.pred5 <- predict(alpha5,s = cv.out5$lambda.min,newx = x.test)
Elastic.pred6 <- predict(alpha6,s = cv.out6$lambda.min,newx = x.test)
Elastic.pred7 <- predict(alpha7,s = cv.out7$lambda.min,newx = x.test)
Elastic.pred8 <- predict(alpha8,s = cv.out8$lambda.min,newx = x.test)
Elastic.pred9 <- predict(alpha9,s = cv.out9$lambda.min,newx = x.test)

# Calculating Test MSE

test0 <- mean((Elastic.pred0-y.test)^2)
test1 <- mean((Elastic.pred1-y.test)^2)
```

```
test2 <- mean((Elastic.pred2-y.test)^2)
test3 <- mean((Elastic.pred3-y.test)^2)
test4 <- mean((Elastic.pred4-y.test)^2)
test5 <- mean((Elastic.pred5-y.test)^2)
test6 <- mean((Elastic.pred6-y.test)^2)
test7 <- mean((Elastic.pred7-y.test)^2)
test8 <- mean((Elastic.pred8-y.test)^2)
test9 <- mean((Elastic.pred9-y.test)^2)


# Calculating test R^2

R0 <- Rsquared(Elastic.pred0,y.test)
R1 <- Rsquared(Elastic.pred1,y.test)
R2 <- Rsquared(Elastic.pred2,y.test)
R3 <- Rsquared(Elastic.pred3,y.test)
R4 <- Rsquared(Elastic.pred4,y.test)
R5 <- Rsquared(Elastic.pred5,y.test)
R6 <- Rsquared(Elastic.pred6,y.test)
R7 <- Rsquared(Elastic.pred7,y.test)
R8 <- Rsquared(Elastic.pred8,y.test)
R9 <- Rsquared(Elastic.pred9,y.test)

# Summarizing results
test.results <- data.frame(alpha = c(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9),
test.MSE = c(test0,test1,test2,test3,test4,test5,test6,test7,test8,test9),
test.Rsquared = c(R0,R1,R2,R3,R4,R5,R6,R7,R8,R9))
test.results

   alpha test.MSE test.Rsquared
1    0.0 25.50343     0.5873824
2    0.1 24.86598     0.5976957
3    0.2 24.71573     0.6001266
4    0.3 24.61770     0.6017126
5    0.4 24.46697     0.6041512
6    0.5 24.38123     0.6055383
7    0.6 24.36941     0.6057295
8    0.7 24.34304     0.6061563
9    0.8 24.29137     0.6069921
10   0.9 24.28296     0.6071282
```

From test.results, it seems that the best performing model is the model with $\alpha = 0.9$ as it has the smallest test MSE and highest R^2 value.

Extracting the Coefficient of Best Elastic Net Model

```
output <- glmnet(x,Data$Y,alpha = 0.9,lambda = grid)
Elastic.coef <- predict(output,type = "coefficients",s=cv.out9$lambda.min)[1:100,]
Elastic.coef[Elastic.coef!=0]

  (Intercept)            X1            X5            X7            X8
 0.8089618769  0.0234154893 -0.0000759793  0.6717599407  1.9338277756
           X9           X10           X13           X20           X22
 2.6561162337  3.8504479545 -0.0009454334  0.1139382128 -0.0022086818
          X23           X26           X27           X28           X31
 0.0034834545  0.0043891085  0.0097817275 -0.0020971076  0.0242015186
          X33           X34           X35           X38           X39
 0.0488922351  0.0657789610  0.0753240806  0.0286286455  0.1348755124
          X40           X43           X44           X50           X51
 0.0032046096  0.1999437594  0.0111271243  0.0998221465  0.0747285516
          X56           X57           X60           X62           X65
 0.0021199463  0.0015740962 -0.0605132441  0.1047029632  0.0012700738
          X68           X69           X70           X71           X72
 0.2570226166  0.0028234159  0.4293314364  0.5807641570 -0.0018782354
          X74           X78           X80           X83           X84
 0.1053521466  0.0326001853 -0.6602357090  0.0255166394  0.0107869717
          X85           X86           X87           X88           X91
```

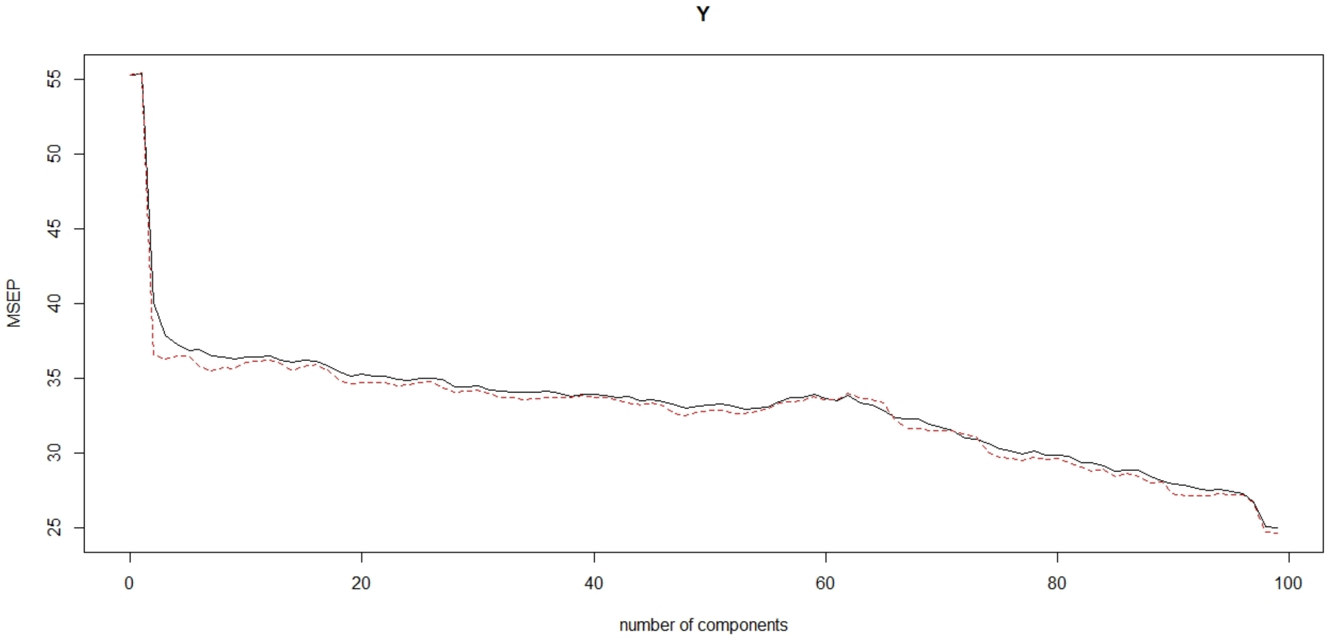

Figure 5: Plot of CV Error as a function of M components

```
0.1014589560  0.2426668174  0.0021958460  0.1359940807  0.3711351162
         X94           X95           X97           X99
0.2184382665  0.1353811381  0.0046800546  0.0019230803
```

Only 48 predictors are included in the model with 51 predictors omitted from it.

Principal Component Analysis

Principal Component Analysis (PCA) is a popular approach to map a large set of predictors to a low dimensional set of features (Principal components). Using PCA, we want to construct the first M principal components to be used as predictors for least squares regression model. The idea is to use a small number of components that is able to explain most of the variability in Data. Doing so, reduces the risk of overfitting.

Plotting CV Error as a function of M

```
library(pls)
train <- which(ind  == 1)
pcr.fit <- pcr(Y˜.,data = Data,subset = train,scale = TRUE,validation = "CV")
validationplot(pcr.fit,val.type = "MSEP")
```

Similar to the glmnet() function, pcr will standardize the predictors inputted for ease of analysis. Furthermore, validation is set to ”CV” would mean that 10 fold cross-validation is carried out to determine the optimal M value.

From Figure 5, we see that as M increases, the training MSE decreases. We now extract the optimal value of M.

Extracting the Optimal value of M

```
cverr <- RMSEP(pcr.fit)$val[1,,]
imin <- which.min(cverr) - 1 # Find the optimal number of PC to use
imin

99 comps
      99
```

It turns out that to minimize training MSE, we need to use all 99 components. We can alsao extract the minimium training MSE using the summary function.

Summary of pcr.fit

```
summary(pcr.fit)


Data:   X dimension: 799 99
        Y dimension: 799 1
Fit method: svdpc
Number of components considered: 99

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           7.436    7.441    6.328    6.154    6.107    6.071    6.077
adjCV        7.436    7.444    6.046    6.024    6.039    6.041    5.979
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       6.043    6.033     6.02     6.037     6.035     6.042     6.014
adjCV    5.957    5.974     5.97     6.002     6.011     6.015     5.999
       14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
CV        6.004     6.019     6.013     5.986     5.952     5.927     5.936
adjCV     5.956     5.982     5.995     5.964     5.905     5.886     5.889
       21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
CV        5.928     5.929     5.911     5.905     5.913     5.913     5.907
adjCV     5.892     5.890     5.874     5.879     5.892     5.895     5.862
       28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
CV        5.869     5.869     5.873      5.85     5.842     5.838     5.837
adjCV     5.836     5.839     5.849      5.83     5.802     5.804     5.790
       35 comps  36 comps  37 comps  38 comps  39 comps  40 comps  41 comps
CV        5.837     5.840     5.831     5.810     5.821     5.824     5.819
adjCV     5.800     5.803     5.807     5.803     5.817     5.805     5.805
       42 comps  43 comps  44 comps  45 comps  46 comps  47 comps  48 comps
CV        5.805     5.809     5.788     5.789     5.782     5.761     5.744
adjCV     5.794     5.774     5.761     5.771     5.757     5.713     5.702
       49 comps  50 comps  51 comps  52 comps  53 comps  54 comps  55 comps
CV        5.756     5.759     5.765     5.752     5.734     5.742     5.751
adjCV     5.721     5.731     5.729     5.714     5.708     5.722     5.735
       56 comps  57 comps  58 comps  59 comps  60 comps  61 comps  62 comps
CV        5.779     5.802     5.802     5.825     5.798     5.784     5.817
adjCV     5.772     5.778     5.784     5.810     5.795     5.791     5.830
       63 comps  64 comps  65 comps  66 comps  67 comps  68 comps  69 comps
CV        5.774     5.762     5.730     5.688     5.682     5.682     5.646
adjCV     5.803     5.793     5.772     5.672     5.622     5.626     5.611
       70 comps  71 comps  72 comps  73 comps  74 comps  75 comps  76 comps
CV        5.631     5.608     5.563     5.560     5.531     5.504     5.486
adjCV     5.610     5.607     5.590     5.575     5.482     5.447     5.445
       77 comps  78 comps  79 comps  80 comps  81 comps  82 comps  83 comps
CV        5.468     5.486     5.463     5.459     5.456     5.417     5.414
adjCV     5.426     5.450     5.435     5.439     5.414     5.391     5.364
       84 comps  85 comps  86 comps  87 comps  88 comps  89 comps  90 comps
CV        5.398     5.363     5.372     5.371     5.329     5.305     5.282
adjCV     5.378     5.327     5.347     5.328     5.289     5.292     5.223
       91 comps  92 comps  93 comps  94 comps  95 comps  96 comps  97 comps
CV        5.275     5.254     5.243     5.245     5.237     5.224     5.166
adjCV     5.205     5.210     5.207     5.222     5.215     5.211     5.158
       98 comps  99 comps
CV        5.005     4.994
adjCV     4.969     4.958
```

```

TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X   2.2139    4.081    5.882     7.63    9.289    10.93    12.55    14.16
Y   0.9103   36.083   36.163    36.16   36.165    38.48    38.72    38.89
   9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X    15.74     17.30     18.83     20.32     21.80     23.27     24.72
Y    38.91     38.97     39.14     39.45     39.54     40.64     40.65
   16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
X     26.16     27.57     28.96     30.34     31.69     33.04     34.37
Y     40.66     41.14     42.14     42.50     42.71     42.72     42.90
   23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
X     35.69     36.98     38.26     39.54     40.80     42.04     43.28
Y     43.14     43.14     43.16     43.22     44.12     44.43     44.43
   30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
X     44.51     45.71     46.89     48.06     49.21     50.34     51.46
Y     44.80     45.08     45.61     45.62     46.13     46.13     46.52
   37 comps  38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
X     52.56     53.66     54.74     55.83     56.89     57.94     58.99
Y     46.53     46.55     46.62     47.32     47.38     47.46     48.18
   44 comps  45 comps  46 comps  47 comps  48 comps  49 comps  50 comps
X     60.03     61.06     62.08     63.08     64.08     65.06     66.03
Y     48.58     48.67     48.99     49.73     49.91     49.93     50.05
   51 comps  52 comps  53 comps  54 comps  55 comps  56 comps  57 comps
X     66.99     67.92     68.85     69.77     70.67     71.56     72.44
Y     50.35     50.53     50.56     50.59     50.66     50.67     51.18
   58 comps  59 comps  60 comps  61 comps  62 comps  63 comps  64 comps
X     73.31     74.18     75.03     75.89     76.73     77.55     78.37
Y     51.18     51.23     51.23     51.23     51.31     51.37     51.59
   65 comps  66 comps  67 comps  68 comps  69 comps  70 comps  71 comps
X     79.18     79.98     80.76     81.53     82.30     83.06     83.81
Y     51.78     53.64     54.69     54.89     54.91     54.94     54.99
   72 comps  73 comps  74 comps  75 comps  76 comps  77 comps  78 comps
X     84.55     85.27     85.99     86.70     87.40     88.08     88.76
Y     55.03     55.51     57.12     57.47     57.59     57.80     57.82
   79 comps  80 comps  81 comps  82 comps  83 comps  84 comps  85 comps
X     89.43     90.09     90.73     91.36     91.98     92.60     93.20
Y     58.04     58.08     58.64     58.64     59.22     59.22     60.02
   86 comps  87 comps  88 comps  89 comps  90 comps  91 comps  92 comps
X     93.80     94.39     94.96     95.52     96.07     96.61     97.13
Y     60.13     60.75     60.95     60.96     62.15     62.71     62.73
   93 comps  94 comps  95 comps  96 comps  97 comps  98 comps  99 comps
X     97.65     98.16     98.65     99.12     99.56     99.99    100.00
Y     62.73     62.75     62.91     62.99     63.54     65.86     66.11

train.mse <- 4.994 ** 2
train.mse
[1] 24.94004
```

We have that the training MSE is 24.94004. However, since 99 components are used, there no dimension reduction and we suspect that there is overfitting involved. We now calculate the test MSE and R^2 value of PCR model.

Evaluating the Test MSE and R^2 of PCR Model

```
pcr.pred <- predict(pcr.fit,x.test,ncomp=9)
mean((pcr.pred-y.test)^2)
[1] 40.33388
pcr.Rsquared <- Rsquared(pcr.pred,y.test)
pcr.Rsquared
[1] 0.347442
```

We see as that as compared to Elastic Net and LASSO, PCR perform poorly on test data (even as worse than Least Squares Fit model as well) This could be as a result of overfitting as it uses all the components to fit the training data.

Partial Least Squares

An alternative dimension reduction method is Partial Least Squares (PLS). Similar to PCR, it finds a new set of features: $Z_1, \ldots, Z_m$ which are linear combinations of the original predictors. Unlike PCR, PLS utilises the response variable to construct the new features.

Training PLS Model

```
pls.fit <- plsr(Y~.,data = Data,subset = train,scale = TRUE,validation = "CV")
summary(pls.fit)

Data:    X dimension: 799 99
         Y dimension: 799 1
Fit method: kernelpls
Number of components considered: 99

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV           7.436    5.210    5.050    5.021    5.010    5.002    5.001
adjCV        7.436    5.179    5.009    4.984    4.974    4.967    4.966
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV       5.003    4.998    5.000     4.999     5.002     5.001     5.000
adjCV    4.968    4.963    4.962     4.962     4.966     4.965     4.965
       14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       35 comps  36 comps  37 comps  38 comps  39 comps  40 comps  41 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       42 comps  43 comps  44 comps  45 comps  46 comps  47 comps  48 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       49 comps  50 comps  51 comps  52 comps  53 comps  54 comps  55 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       56 comps  57 comps  58 comps  59 comps  60 comps  61 comps  62 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       63 comps  64 comps  65 comps  66 comps  67 comps  68 comps  69 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       70 comps  71 comps  72 comps  73 comps  74 comps  75 comps  76 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       77 comps  78 comps  79 comps  80 comps  81 comps  82 comps  83 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       84 comps  85 comps  86 comps  87 comps  88 comps  89 comps  90 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       91 comps  92 comps  93 comps  94 comps  95 comps  96 comps  97 comps
CV        5.000     5.000     5.000     5.000     5.000     5.000     5.000
adjCV     4.965     4.965     4.965     4.965     4.965     4.965     4.965
       98 comps  99 comps
CV        5.000     5.000
adjCV     4.965     4.965

TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X    1.751    2.815    4.219    5.697    6.826    7.921    8.993    9.909
```

```
Y   58.438   64.853   65.659   65.826   65.864   65.873   65.880   65.916
   9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X    10.52     11.37     12.63     13.75     14.86     15.86     16.87
Y    66.04     66.10     66.11     66.11     66.11     66.11     66.11
   16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
X     17.94     18.96     20.04     21.22     22.29     23.42     24.55
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
X     25.56     26.64     27.73     28.78     29.81     30.88     31.98
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
X     33.05     34.16     35.27     36.32     37.37     38.42     39.48
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   37 comps  38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
X     40.58     41.56     42.63     43.71     44.76     45.81     46.80
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   44 comps  45 comps  46 comps  47 comps  48 comps  49 comps  50 comps
X     47.78     48.82     49.82     50.86     51.80     52.79     53.77
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   51 comps  52 comps  53 comps  54 comps  55 comps  56 comps  57 comps
X     54.83     55.93     56.84     57.78     58.79     59.88     60.87
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   58 comps  59 comps  60 comps  61 comps  62 comps  63 comps  64 comps
X     61.82     62.79     63.71     64.65     65.59     66.52     67.46
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   65 comps  66 comps  67 comps  68 comps  69 comps  70 comps  71 comps
X     68.39     69.32     70.25     71.19     72.12     73.06     73.99
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   72 comps  73 comps  74 comps  75 comps  76 comps  77 comps  78 comps
X     74.92     75.86     76.79     77.72     78.66     79.59     80.53
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   79 comps  80 comps  81 comps  82 comps  83 comps  84 comps  85 comps
X     81.46     82.39     83.33     84.26     85.20     86.13     87.07
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   86 comps  87 comps  88 comps  89 comps  90 comps  91 comps  92 comps
X     88.00     88.93     89.87     90.80     91.74     92.67     93.60
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
   93 comps  94 comps  95 comps  96 comps  97 comps  98 comps  99 comps
X     94.54     95.47     96.41     97.34     98.27     99.21    100.14
Y     66.11     66.11     66.11     66.11     66.11     66.11     66.11
```

From inspecting the output of the summary function, it seems that we do not need to use all 99 components in our model to minimize CV Error. Note that by default, plsr() uses 10 fold cross-validation as well. For easier visualisation let plot CV Error as a function of PLS Components.

Plotting CV Error as a Function of PLS Components

```
validationplot(pls.fit,val.type="MSEP")
```

From Figure 6, it seems that we do need to use all 99 components to minimize CV Error. We now extract the optimal number of PLS components from pls.fit.

Extracting Optimal number of PLS Components

```
cverr <- RMSEP(pls.fit)$val[1,,]
imin <- which.min(cverr) - 1
imin

8 comps
      8
```

Thus, we use 8 PLS Components to minimize CV Error.

Evaluating the Test MSE and R^2

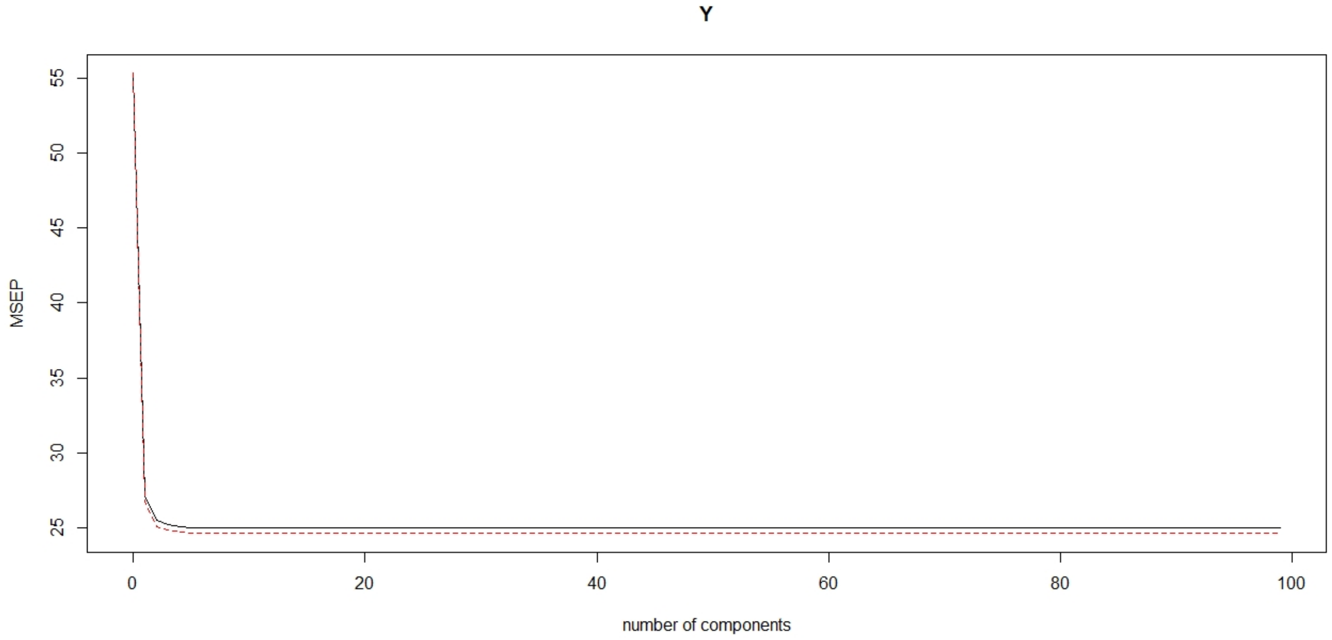

Figure 6: CV Error as a function of PLS Components

```
pls.pred <- predict(pls.fit,x.test,ncomp = 8)
mean((pls.pred-y.test)^2)
[1] 25.82683
pls.Rsquared <- Rsquared(pls.pred,y.test)
pls.Rsquared
[1] 0.5821501
```

PLS does perform better than PCR on test data. However, Elastic Net and LASSO still performs better than PLS.

XGBoost

Extreme Gradient Boosting (XGBoost) is one of the popular machine learning algorithms and has been known to high prediction accuracy. It is considered state-of-the-art machine learning algorithms. XGBoost is essentially a boosting algorithm that can be used to train models for both classification and regression problems. It is in fact an example of Gradient Boosted Decision Tree Algorithm which builds a model from an ensemble of weaker models (gives poor test MSE and R^2 values) such as decision trees.

A generic Gradient Boosted Algorithm involves a training set $\{(x_i, y_i)\}_{i=1}^{n}$, differentiable Loss Function $L(y, F(x))$ and M number of iterations. Then,

1. Initialise model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. For $m = 1$ to M,

(a) Compute pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, i = 1, \ldots, n$$

(b) Fit a base learner (weak learner eg. tree) closed under $h_m(x)$ to pseudo-residual i.e. train it using training set

(c) Derive γ_m by solving

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

(d) Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

3. Output $F_M(x)$ (Friedman, J. H. (February 1999))

Intuitively, Gradient Boosting uses the gradient of the Loss function to minimize the Objective Value. In each iteration, Gradient Boosting uses the gradient of the loss function to find the direction the model parameter vector should take in order to minimize the residual squared error. This is similar to the Gradient Descent algorithm. However, XGBoost uses the Hessian of the Loss function similar to Newton's method to do. This gives better approximated values and thus better gives better directions to minimize residual squared error. Additionally, it utilizes regularization (l_1, l_2 norm) for the same method. (Rblogger)

Note that the parameters used in XGBoost are:

1. nround: maximum number of iteration used by xgboost to train model
2. γ: Regularization parameter. Larger values reduces the risk of overfitting
3. eta: learning rate of XGBoost (Default rate is 0.3, value usually lies between 0.01 and 0.3)
4. maxdepth: controls the depth of the tree (smaller trees reduce the risk of overfitting)
5. subsample: controls the number of samples given to each tree (learner)
6. colsampletree: is the proportion of predictors to be used
7. minimum child weight: Parameters is used to minimized possible predictor interaction to reduce risk of overfitting
8. max delta step: refer to the maximum weight given to each decision tree (hackerearth)

Finding optimal parameters for XGBoost Model using 10 fold Cross-Validation

```r
library(xgboost)
d.train <- xgb.DMatrix(x.train, label = y.train)
d.test <- xgb.DMatrix(x.test, label = y.test)

# Iterate through 200 times to find best parameters
# to minimize Training MSE


optimal.parameters <- list() # store the optimal parameters after for loop
optimal.seed <- 4211
optimal.rmse <- Inf # store minimum root MSE
optimal.idx <- 0 # Locate the smallest minimum root MSE

for (iter in 1:200){
  parameters <- list(objective = "reg:linear",
        eval_metric = "rmse",
        max_depth = sample(2:10,1),
        eta = runif(1,0.01,0.3),
        subsample = runif(1,0.6,0.9),
        colsample_bytree = runif(1,0.5,0.8),
        min_child_weight = sample(1:40,1),
        max_delta_step = sample(1:10,1))

  # At each iteration make use of Randomisation to minimize
  # training MSE

  no.rounds <- 1000
  cv.folds <- 10 # Performing 10 fold Cross-Validation
  current.seed <- sample.int(10000,1) # find the best seed that minimizes CV
  set.seed(current.seed)

  model <- xgb.cv(data = d.train,params = parameters,nfold = cv.folds,
        nrounds = no.rounds,verbose = F,
        early_stopping_rounds = 8,maximise = FALSE)

  current.idx <- model$best_iteration
  current.rmse <- model$evaluation_log[current.idx]$test_rmse_mean
  # update the optimal rmse
  if (current.rmse < optimal.rmse){
    optimal.parameters <- parameters
    optimal.idx <- current.idx
    optimal.seed <- current.seed
    optimal.rmse <- current.rmse}

}

# Source: https://yangliuresearch.blogspot.com/2018/07/extreme-gradient-boosting-xgboost.html
```

Optimal Parameters obtained from Random Search

```
set.seed(optimal.seed)
xg_mod <- xgboost(data = d.train,params = optimal.parameters,nround = optimal.idx,verbose = F)

# Best Tuning Parameters
result <- data.frame(optimal.parameters,optimal.idx,optimal.rmse,optimal.seed)
result

  objective eval_metric max_depth        eta subsample colsample_bytree
1 reg:linear        rmse         2 0.02979434 0.7835935        0.7236754
  min_child_weight max_delta_step optimal.idx optimal.rmse optimal.seed
1               19              3         370     4.837736          575

```

Feature Importance

```
 importance_matrix <- xgb.importance(colnames(x.train), model = xg_mod)
library(Ckmeans.1d.dp) # for xgb.ggplot.importance
xgb.ggplot.importance(importance_matrix, top_n = 10,
                      measure = "Gain")
# Source: https://yangliuresearch.blogspot.com/2018/07/extreme-gradient-boosting-xgboost.html

```

From Figure 7, it seems that X10 is by far the most important feature followed by X9 and 8 (by a significant margin).

Plotting the first few Trees of the XGBoost Model

```
library(DiagrammeR)
```

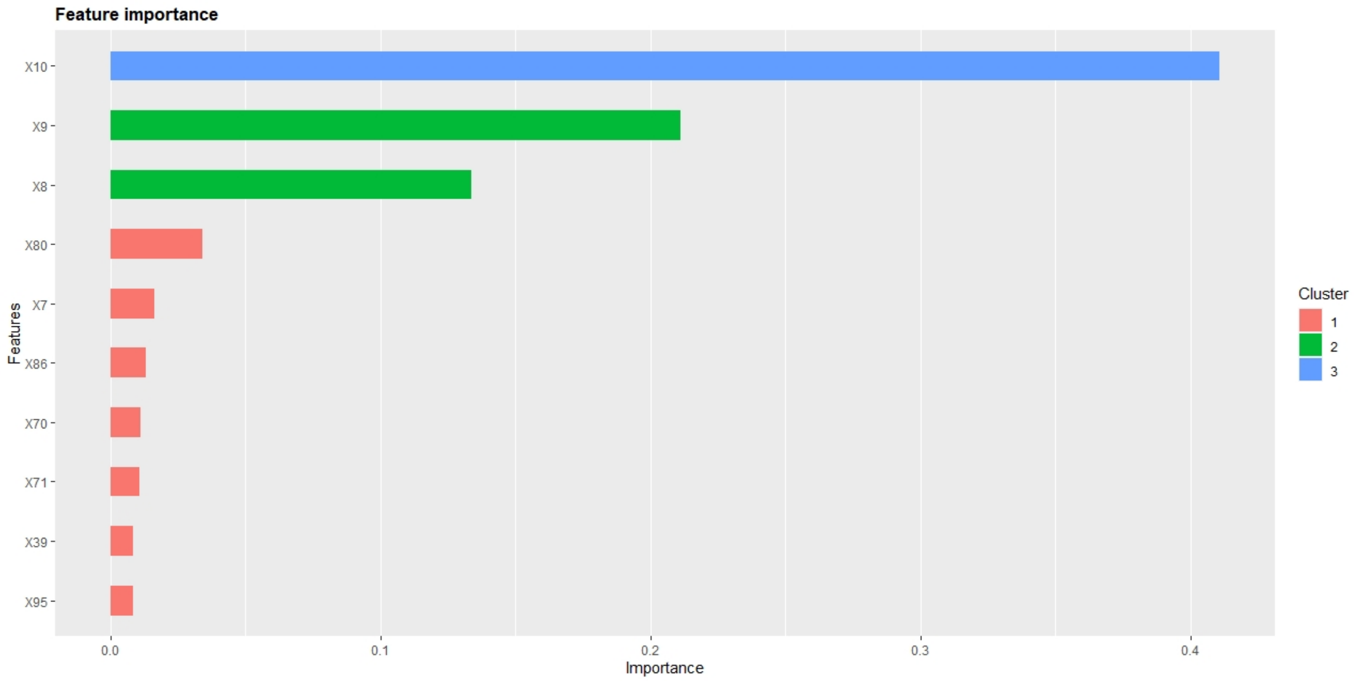

Figure 7: XGBoost Feature Importance

```
xgb.plot.tree(model = xg_mod,trees = 1)
xgb.plot.tree(model = xg_mod,trees = 2)
xgb.plot.tree(model = xg_mod,trees = 3)
```

As mentioned previously, Xgboost is an ensemble of weaker model. We will plot the first few decision trees of the Xgboost model. From Figure 8,9,10, we have the first 3 trees of the Xgboost model.

Test MSE and R^2 value

```
# Calculate test MSE and R^2
yhat_xg <- predict(xg_mod,d.test)
test.mse <- mean((yhat_xg-y.test)^2)
test.mse
[1] 25.14059
Rsquared(yhat_xg,y.test)
[1] 0.5932528
```

It seems that XGBoost is almost as good as LASSO but LASSO is still the best performing model on test data.

Predicting Unseen Data

```
# LASSO is the best performing model on test data
# we shall use LASSO to predict values of y from an independent set
# Upload Data Set to R
test <- read.csv(file.choose(),header = T)
test <- test[,-55] # remove X55
edited <- data.frame(Y = rep(1,nrow(test)),test)
final.test <- model.matrix(Y~.,edited)[,-1]
Y <- predict(lasso.mod,s = bestlam,newx = final.test)
values <- data.frame(Y = Y)
library(readr)
setwd("C:\\Users\\Aiman\\Documents\\DSA4211")
write_csv(values,"A0187108Bfinal.csv")
```

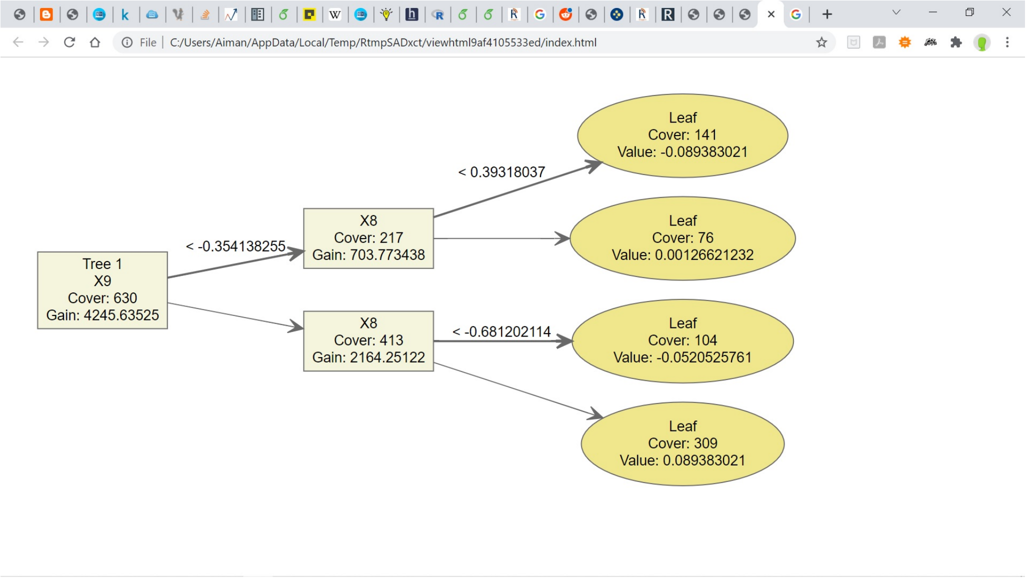

Figure 8: First Tree of the Xgboost model

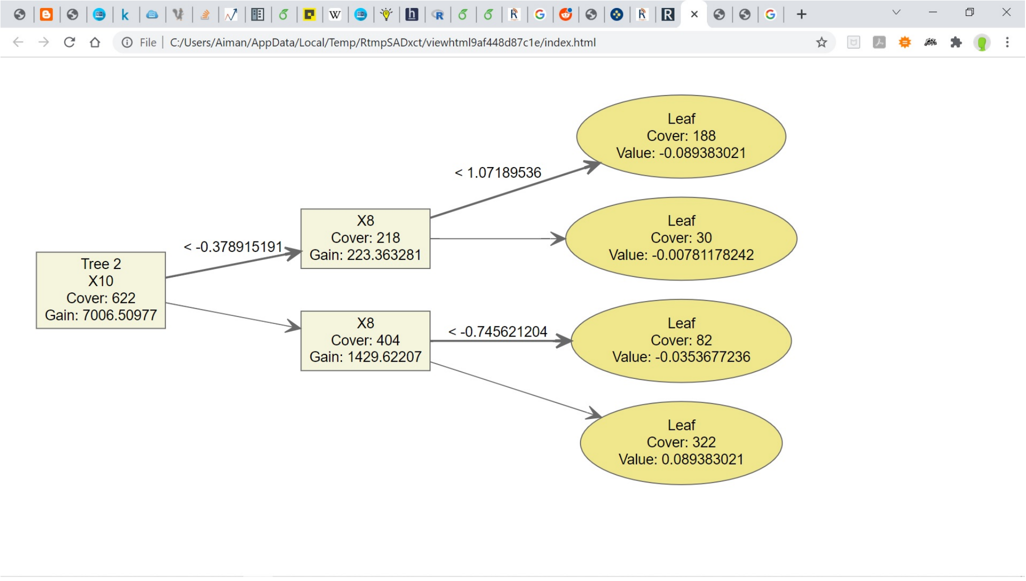

Figure 9: Second Tree of the Xgboost model

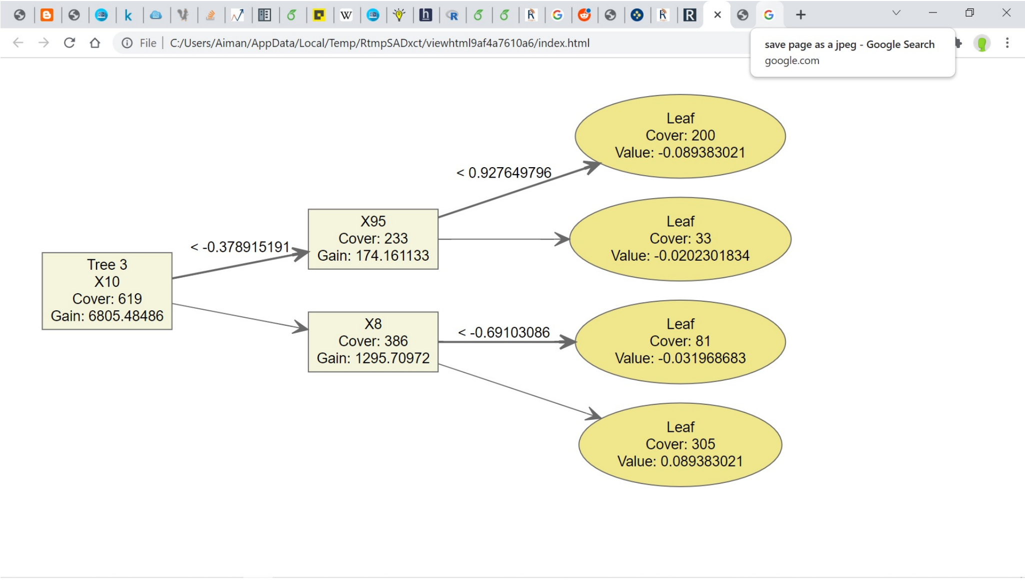

Figure 10: Third Tree of Xgboost Model

References

1. Friedman, J. H. (February 1999). ”Greedy Function Approximation: A Gradient Boosting Machine”
2. Beginners tutorial on XGBoost and parameter tuning in R tutorials amp; notes: Machine learning. HackerEarth. (n.d.). Retrieved November 8, 2021, from https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/.
3. Glander, D. S. (2018, November 29). Machine learning basics – gradient boosting amp; xgboost: R-bloggers. R. Retrieved November 8, 2021, from https://www.r-bloggers.com/2018/11/machine-learning-basics-gradient-boosting-xgboost/.
4. Liu, Y. (2018, July 10). Extreme gradient boosting (XGBoost): Better than random forest or gradient boosting. eXtreme Gradient Boosting (XGBoost): Better than random forest or gradient boosting. Retrieved November 8, 2021, from https://yangliuresearch.blogspot.com/2018/07/extreme-gradient-boosting-xgboost.html.