Deploying Several Machine Learning Models and Candidate the Best Model "data2"

Samer Hisham Ismail

Cohort A20

Date

1st of April 2021

Course title

Advanced Statistics and Machine Learning

Pr.Christine Malot

Overview

This document is to report the results of building several Machine Learning models with R, with “data2”, explaining the Grade response variable, with several independent variables.

The key to this exercise is to assess the performance of the models and candidate the best one.

Main Modeling settings:

- The problem considered as multivariate supervised machine learning problem. Since we are going to explain continuous variable, the models will be in regression techniques.

- We will build the model and assess it using ‘Caret’ R package, as it makes the process of training, tuning, and evaluation consistent, because it uses the Cross - Validation technique.

- Models performance will be evaluated with RSME metric.

▪ Project Steps:

1. **Choosing the Packages libraries:**
   As mentioned before, the package Caret will be used to Train, Tune and *Predict()* function for testing the models.

2. **Algorithm Selection:**
   Algorithms used in this project are, Full Linear Regression, Stepwise with AIC, Forward and Backward with AIC, Ridge, Lasso, Principal Component Regression, Partial Least Square, Random Forest.

3. **Data Exploration:**
   Exploring the dataset and variables type and distribution using plots, assessing, and testing the Gaussianity (Normality) of the residuals using KSsmirnov Test, in addition of correlation analysis between variables using correlation plot and VIF (Variable Inflation Factor).

4. **Data Split:**
   Splitting the data into two parts, Training and Testing using *createDataPartition()* function.

5. **Setting Train Control:**
   Train control is a function within 'Caret' Package that allows to specify the method of training whether it is *Leave One Out* or *Repeated Cross Validation* or *K- Folds Cross validation* which will be used in this project with 5 folds, also it allows to do the grid search during the training process to identify the best hyperparameters that generates the best model, in addition of its ability to store the models generated during Cross Validation process to be viewed and evaluated later.
   The reason we picked *K folds CV* and not *Leave one out* method, because we wanted the training to be consistent without bias, which might be the case with LOOCV, although its good for small datasets like we have here in our project.

6. **Building Models:**
   In this step we will be using *train()* function within the 'Caret' Package, to train the models using prespecified *trainControl()*, with Root Mean Squared Error (RSME) as performance metric.

7. **Predictions:**
   Again, with '*Caret'* Package, the function used to predict is *predict(),* using the unseen data 'Testing' dataset, although 'Repeated Cross Validation' method generates decent results, but it is a way to assess models with unseen data to avoid any surprises. The performance metric to be used in prediction is 'RMSE' predicted.

8. **Summarize Training and Prediction models:**
   Using *resamples()* function, and data frame where the results will be stored in, also we will use RSME and RMSE_Predicted to assess the models and candidate the best one.

▪ Importing data2 and needed Packages:

```
### Libraries Import & Installation ###

library(olsrr)                    # Linear Regression utility Package
library(e1071)                    # Utilities Package
library(PerformanceAnalytics)     # Utilities Package for Linear Regression
library(corrplot)                 # Plot Correlation Package
library(pls)                      # Package for PCA and PLS
library(caret)                    # Main Package that we will be working on
library(foba)                     # Package for Ridge
library(elasticnet)               # Package for lasso
library(remotes)                  # Download specific version of packages form the web
```

a) Data Import and preparation:

```
### Data Import ###

    setwd("C:/Users/Samer/Desktop/ASML_RAW")
    data2  <- read.csv('data2.csv',header = TRUE, sep= ";")
    str(data2)
```

```
'data.frame':  16 obs. of  6 variables:
 $ Product: int  1 2 3 4 5 6 7 8 9 10 ...
 $ Sugar  : num  6.21 7.75 7.21 8.33 4.87 5.09 6.04 6.09 6.08 6.17 ...
 $ Acid   : num  7.08 3.29 4.38 2.79 7.71 7.5 6.58 5.13 5.5 5.58 ...
 $ Bitter : num  2 1.54 1.79 1.63 1.96 2.13 2.04 2 2.09 2.13 ...
 $ Pulpy  : num  2.54 2.26 2.58 2.71 1.7 2.42 2.04 2.42 2.46 2.48 ...
 $ Grade  : num  4.97 6.98 4.58 6.45 4.33 4.26 6.16 6.26 5.83 5.74 ...
```

After importing and eyeballing the data, I found out that the Independent variables and Response variable are numeric except 'Product' as its an ID variable and its just there for identification and does not have any impact on the response since it have 16 levels, therefore it should not be included in the models.

Dropping the 'Product' independent variable:

```
data2<-data2[,2:6]
str(data2)
```

```
'data.frame':  16 obs. of  5 variables:
 $ Sugar : num  6.21 7.75 7.21 8.33 4.87 5.09 6.04 6.09 6.08 6.17 ...
 $ Acid  : num  7.08 3.29 4.38 2.79 7.71 7.5 6.58 5.13 5.5 5.58 ...
 $ Bitter: num  2 1.54 1.79 1.63 1.96 2.13 2.04 2 2.09 2.13 ...
 $ Pulpy : num  2.54 2.26 2.58 2.71 1.7 2.42 2.04 2.42 2.46 2.48 ...
 $ Grade : num  4.97 6.98 4.58 6.45 4.33 4.26 6.16 6.26 5.83 5.74 ...
```

Now, since we dropped the 'Product' variable, we have the numeric response and independent variables as shown in the Above table.

Data Split:

The next step is to split the data 65% into Train (obs=12), Test (obs=4)

```
set.seed(1234)
trainingIndex <- createDataPartition(data2$Grade, p=0.65, list=FALSE)
training <- data2[trainingIndex,]
testing <- data2[-trainingIndex,]

training
testing
```

Training

Sugar	Acid	Bitter	Pulpy	Grade	
1	6.21	7.08	2.00	2.54	4.97
2	7.75	3.29	1.54	2.26	6.98
4	8.33	2.79	1.63	2.71	6.45
6	5.09	7.50	2.13	2.42	4.26
7	6.04	6.58	2.04	2.04	6.16
9	6.08	5.50	2.09	2.46	5.83
10	6.17	5.58	2.13	2.48	5.74
11	7.17	3.54	1.52	2.33	6.80
12	7.08	3.25	1.46	2.17	6.12
13	4.30	8.33	2.29	1.58	4.02
15	4.52	8.33	2.48	1.74	3.53
16	4.70	6.39	2.17	1.79	5.10

Testing

Sugar	Acid	Bitter	Pulpy	Grade	
3	7.21	4.38	1.79	2.58	4.58
5	4.87	7.71	1.96	1.70	4.33
8	6.09	5.13	2.00	2.42	6.26
14	5.61	5.13	2.13	1.81	6.05

▪ Data Exploration:

At first, we will start with general descriptive statistics analysis to review data and to assess its skewness or whether if data needs to be standardized or normalized.

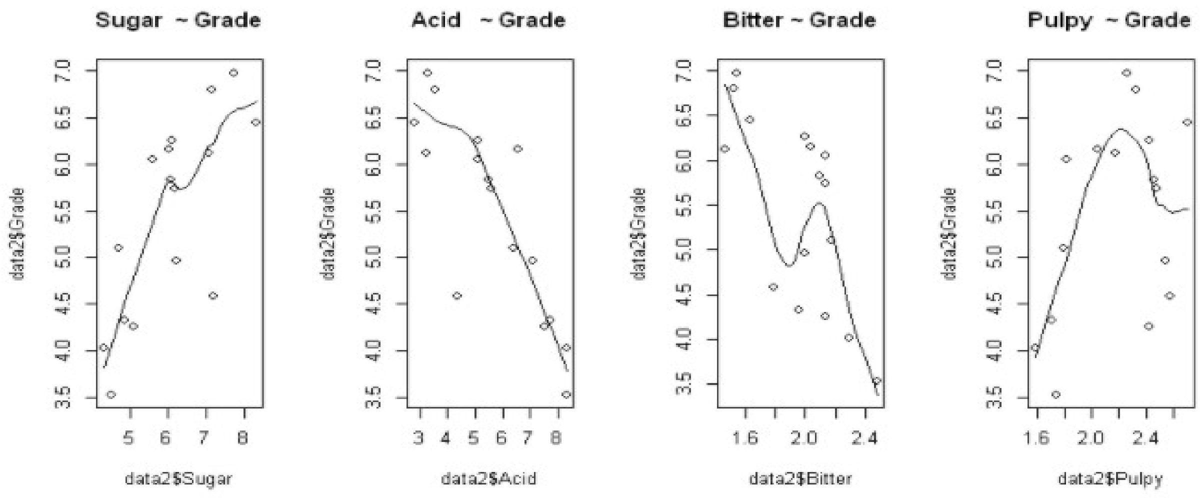
```
### Data Exploration ###
summary(data2)
```

```
     Sugar            Acid            Bitter          Pulpy           Grade
 Min.   :4.300   Min.   :2.790   Min.   :1.46   Min.   :1.580   Min.   :3.530
 1st Qu.:5.035   1st Qu.:4.170   1st Qu.:1.75   1st Qu.:1.805   1st Qu.:4.518
 Median :6.085   Median :5.540   Median :2.02   Median :2.295   Median :5.785
 Mean   :6.076   Mean   :5.657   Mean   :1.96   Mean   :2.189   Mean   :5.449
 3rd Qu.:7.103   3rd Qu.:7.185   3rd Qu.:2.13   3rd Qu.:2.465   3rd Qu.:6.185
 Max.   :8.330   Max.   :8.330   Max.   :2.48   Max.   :2.710   Max.   :6.980
```

The data is almost consistent, mean and median are somehow close, but in Grade, which is the Response variable, its median is a bit higher than the mean which is an indicator of slight skewness to the left, so the distribution of this variable not perfectly normal, however, there will be no such a huge impact on the models since it is a Regression problem.
As for variables minimum and maximums, they seem to be almost symmetric.

Scatter Plot:

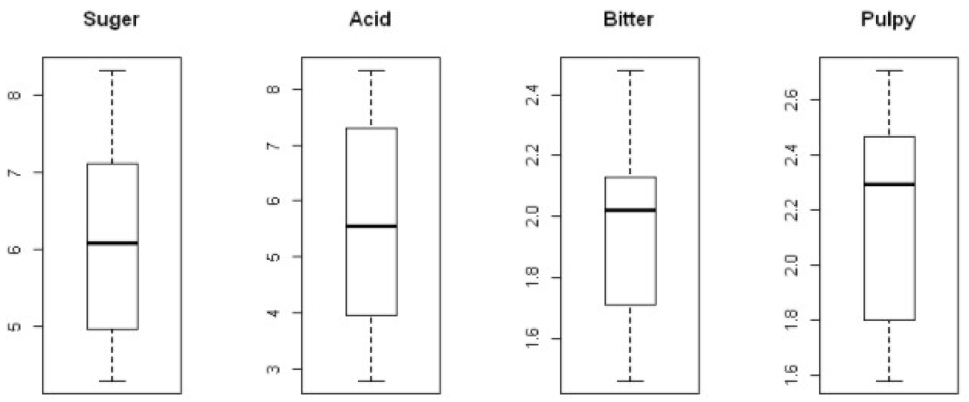
```
par(mfrow=c(2, 4)) #  chart area size
scatter.smooth(x=data2$Sugar,  y=data2$Grade, main="Sugar  ~ Grade")
scatter.smooth(x=data2$Acid,   y=data2$Grade, main="Acid   ~ Grade")
scatter.smooth(x=data2$Bitter, y=data2$Grade, main="Bitter ~ Grade")
scatter.smooth(x=data2$Pulpy,  y=data2$Grade, main="Pulpy  ~ Grade")
```



This scatterplot shows a weak, positive, linear association between (Sugar-Grade) and (Publy-Grade). With moderate negative linear association between (Acid – Grade), in addition of weak negative linear association between (Bitter – Grade).
Also, there might existence of some outliers in data, we will investigate into that next using Boxplot.

Box Plots:

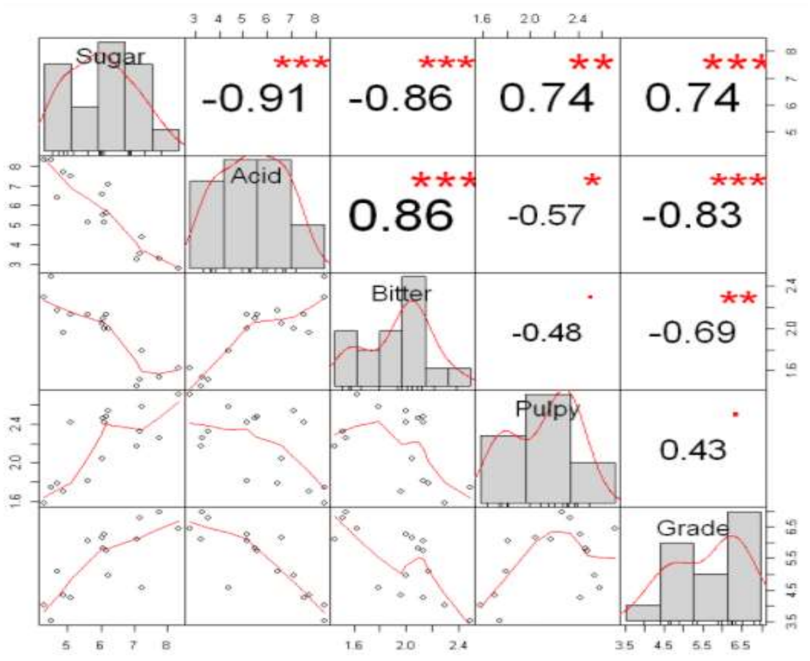
```
#Boxplots
par(mfrow=c(2, 4))
boxplot(data2$Sugar,  main="Suger",  sub=paste("Outlier rows: ",
boxplot.stats(data2$Sugar)$out))
boxplot(data2$Acid,   main="Acid",   sub=paste("Outlier rows: ",
boxplot.stats(data2$Acid)$out))
boxplot(data2$Bitter, main="Bitter", sub=paste("Outlier rows: ",
boxplot.stats(data2$Bitter)$out))
boxplot(data2$Pulpy,  main="Pulpy",  sub=paste("Outlier rows: ",
boxplot.stats(data2$Pulpy)$out))
```



According to the boxplot, the data is relatively consistent, with no outliers, so we can proceed our modeling process.

Variables Correlation:

```
#Correlation Matrix
chart.Correlation(data2, histogram=TRUE, pch=30)
```



- As we observe form the above correlation matrix, there is high correlation between several variables, The strongest correlation is between (Acid - Sugar), (Bitter - Acid) and (Bitter – sugar). That lead us to suspect Multicollinearity issue between these variables.

- To confirm Multicollinearity, we will use *VIF()* Variance Inflation Factor, using olsrr package.
- In this function we will input full Linear Regression model as shown below:

```
# Double Check for the Multicollinearity:
#Variance Inflation Factor (VIF)
ols_vif_tol(lm(Grade ~ ., data = data2))
```

Variables	Tolerance	VIF
Sugar	0.07113012	14.058741
Acid	0.13448096	7.435997
Bitter	0.19840911	5.040091
Pulpy	0.33224353	3.009840

The rule of thumb for interpreting VIF:

VIF = 1:	not correlated.
1< VIF <= 5:	moderately correlated.
VIF > 5:	highly correlated

As a result, in our case we have two highly correlated variables, which are Sugar and Acid.
And two moderately correlated variable, which are Bitter and Publy.

trainControl() Function Settings:

This function is part of the train function, where we specify the training settings that we want to use in our models, in our models, we will use K-Fold CV with 5 folds, in addition of *savePredictions()* function that allow us to keep the final best model based on RMSE.

Defining training control for cross validation:

```
set.seed(1234)
train.control <- trainControl(number = 5, method="cv",savePredictions = "final")
```

- ## Building Models:

As mentioned earlier, we will be using ‘*Caret*’ package to train and *predict()* to test models, in addition we might use another packages like *ols_best_subset*, and also we may have to build the models in a simple classical way, because sometimes I notice that ‘*Caret*’ hides some details underneath its code.

Full Linear Regression Model

```
## Full Linear Regression Model ###
set.seed(1234)
lm_full   <- train(Grade ~., data = training, method = "lm",trControl =
train.control,metric="RMSE")

#Model Results
summary(lm_full)
```

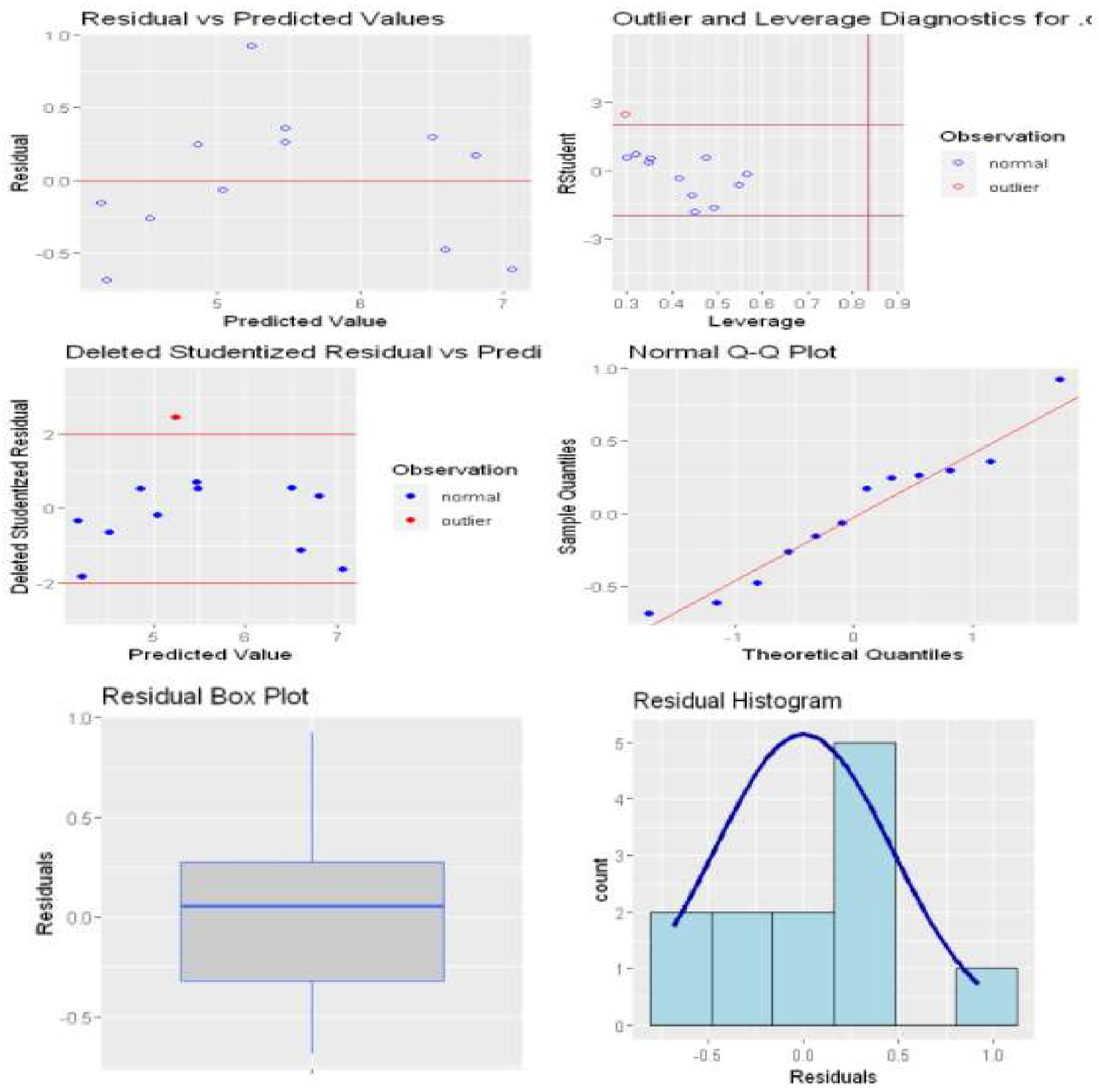
```
Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-0.68384 -0.31627  0.05307  0.27159  0.92298

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.50514    4.40431   1.250    0.251
Sugar        0.36821    0.49755   0.740    0.483
Acid        -0.29679    0.28773  -1.031    0.337
Bitter      -0.03163    1.48200  -0.021    0.984
Pulpy       -0.23269    0.80688  -0.288    0.781

Residual standard error: 0.5848 on 7 degrees of freedom
Multiple R-squared:  0.8268, Adjusted R-squared:  0.7278
F-statistic: 8.354 on 4 and 7 DF,  p-value: 0.00842
```

- We notice some symmetry as first impression, the F- statistics seems to be significant, adjusted R-Squared is relatively good, we will try several models to see if it could be enhanced somehow.

- The p-value of the predictors is not significant as all values are way above 5%

- Have symmetry is not enough, because we are not sure about the Gaussianity of the residuals, hence we will use q-q plot and make several tests including KS test to confirm.

```
# Plot
par(mfrow=c(2, 4))
ols_plot_diagnostics(lm_full$finalModel)

# Confirming Gausianity of the residulas
 ols_test_normality(lm(Grade ~ ., data = training))
```



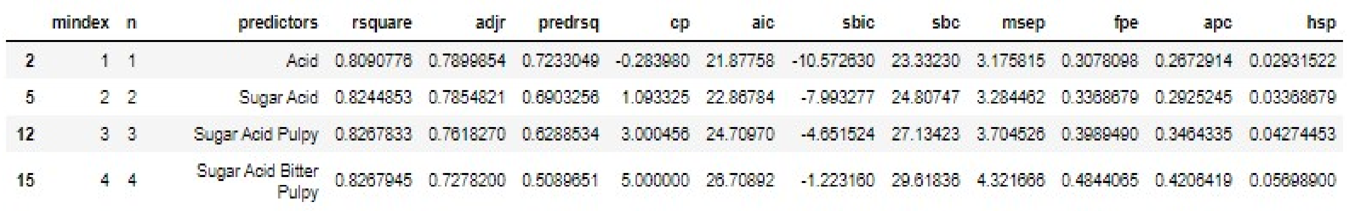
```
-----------------------------------------------
       Test             Statistic       pvalue
-----------------------------------------------
Shapiro-Wilk              0.9539         0.6950
Kolmogorov-Smirnov        0.144          0.9347
Cramer-von Mises          1.497           1e-04
Anderson-Darling          0.2741         0.5958
-----------------------------------------------
```

By looking at the above plots, we can spot the normality of the residuals, but to be surer and to double check, we used KS test, Shaprio Test, Cramer Test.
Since we have high p-value we accept the Null hypothesis, which means that we accept the Gaussianity of the Residuals.

Best Subset Regression:

in this model, we will use *ols_step_best_subset* function, that will choose the best subset of the predictors generates highest R squared and lowest AIC, so it could be used also for variable selection.

```
### Best Subset Regression  ###
ols_step_best_subset(lm(Grade ~ ., data = training))
```



	mindex	n	predictors	rsquare	adjr	predrsq	cp	aic	sbic	sbc	msep	fpe	apc	hsp
2	1	1	Acid	0.8090776	0.7899854	0.7233049	-0.283980	21.87758	-10.572630	23.33230	3.175815	0.3078098	0.2672914	0.02931522
5	2	2	Sugar Acid	0.8244853	0.7854821	0.6903256	1.093325	22.86784	-7.993277	24.80747	3.284462	0.3368679	0.2925245	0.03368679
12	3	3	Sugar Acid Pulpy	0.8267833	0.7618270	0.6288534	3.000456	24.70970	-4.651524	27.13423	3.704526	0.3989490	0.3464335	0.04274453
15	4	4	Sugar Acid Bitter Pulpy	0.8267945	0.7278200	0.5089651	5.000000	26.70892	-1.223160	29.61836	4.321666	0.4844065	0.4206419	0.05698900

The best subset selected by the model is the first model with only one predictor (Acid), which have the highest Adj R- squared, and lowest AIC.

Stepwise Regression with AIC

Stepwise regression is a special Linear Regression model, but the main difference is that the stepwise model inserts and removes the explanatory variables based on AIC.

```
### Stepwise with AIC ###
set.seed(1234)
step_aic  <- train(Grade ~., data = training, method = "lmStepAIC", trControl =
train.control, trace=FALSE,metric="RMSE")
#Model Results
summary(step_aic)
print(step_aic $results)
# Plot
ols_plot_diagnostics(step_aic$finalModel)
```

```
Call:
lm(formula = .outcome ~ Acid, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.64901 -0.36970  0.06291  0.24041  1.11084 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  8.32093    0.45849   18.15 5.53e-09 ***
Acid        -0.49723    0.07638   -6.51 6.81e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5137 on 10 degrees of freedom
Multiple R-squared:  0.8091,	Adjusted R-squared:   0.79 
F-statistic: 42.38 on 1 and 10 DF,  p-value: 6.81e-05

parameter      RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
1      none 0.7285636 0.8376783 0.675919 0.4013398  0.2353388 0.3771335
```

- The stopping rule is applied, and the only variable was selected is ACID, stopping rule when we see the "none", so the best model is the model with intercept and Acid only.
  Note: "*none*" is not available for *Caret*
- we notice here that it is the same model selected with the best subset regression with the same Adjusted R-Squared and same F- statistics.

Backward Regression with AIC:

This model works by starting with Full Linear model with all predictors, then removing variables one by one until finding the best models that explain the data.

```
### Backward with AIC ###
set.seed(1234)
back_aic  <- train(Grade ~., data = training, method = "leapBackward",trControl =
train.control, trace=FALSE,metric="RMSE")

#Model Results
summary(back_aic)
print(back_aic $results)
```

```
Subset selection object
4 Variables  (and intercept)
       Forced in Forced out
Sugar      FALSE      FALSE
Acid       FALSE      FALSE
Bitter     FALSE      FALSE
Pulpy      FALSE      FALSE
1 subsets of each size up to 2
Selection Algorithm: backward
         Sugar Acid Bitter Pulpy
1  ( 1 ) " "   "*"  " "    " "
2  ( 1 ) "*"   "*"  " "    " "


Linear Regression with Backwards Selection

12 samples
 4 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 9, 10, 9, 10, 10
Resampling results across tuning parameters:

  nvmax  RMSE       Rsquared   MAE
  2      0.7643586  0.8451694  0.7238967
  3      0.8310737  0.8723055  0.7828474
  4      0.8117601  0.8853155  0.7664424

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was nvmax = 2.

nvmax      RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
1     2 0.7643586 0.8451694 0.7238967 0.3759825  0.2305501 0.3542619
2     3 0.8310737 0.8723055 0.7828474 0.3957911  0.1770776 0.3584400
3     4 0.8117601 0.8853155 0.7664424 0.3754842  0.1570772 0.3374706
```

The best model was selected using cross validation is the model with the lowest RSME, which is the model with one predictor (ACID)

Forward Regression with AIC:

The exact opposite of the Backward model, this model starts with the minimum Linear Model with one variable, and then gradually expand the model by adding one variable every time.

```
### Forward with AIC ###
          set.seed(1234)
          fwd_aic   <- train(Grade ~., data = training, method = "leapForward"
,trControl = train.control, trace=FALSE,metric="RMSE")

#Model Results
summary(fwd_aic)
print(fwd_aic$results)
```

```
Subset selection object
4 Variables  (and intercept)
       Forced in Forced out
Sugar      FALSE      FALSE
Acid       FALSE      FALSE
Bitter     FALSE      FALSE
Pulpy      FALSE      FALSE
1 subsets of each size up to 2
Selection Algorithm: forward
         Sugar Acid Bitter Pulpy
1  ( 1 ) " "   "*"  " "    " "
2  ( 1 ) "*"   "*"  " "    " "


Linear Regression with Forward Selection

12 samples
 4 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 9, 10, 9, 10, 10
Resampling results across tuning parameters:

  nvmax  RMSE       Rsquared   MAE
  2      0.7226769  0.8370325  0.6912900
  3      0.8081900  0.8359608  0.7683330
  4      0.8117601  0.8853155  0.7664424

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was nvmax = 2.
```

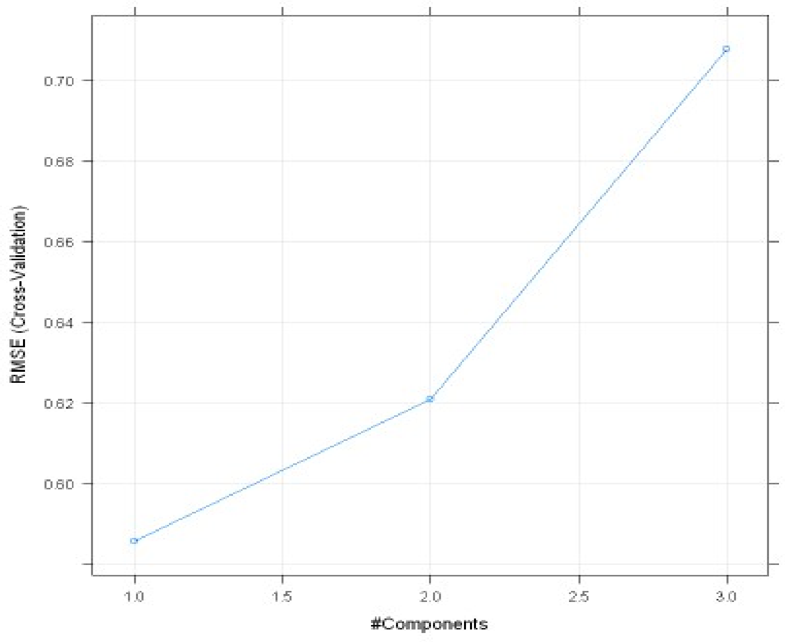
Until now, the models are choosing the best models with the variable (Acid)

Principal Component Regression:

Its good idea to do the PCR model, because it can do some dimension reduction and deal with highly correlated data, which is exactly our case, as it summarizes the explanatory variables into a smaller number of variables.

```
### Principle Component Regression ###

set.seed(1234)
#pca <- pcr(Grade ~ ., data = training, validation = “CV”)
pca <- train(Grade ~., data = training, method = “pcr”,trControl =
train.control,metric=”RMSE”)

#Model Results
Print(pca$results)

# Plot
plot(pca)
validationplot(pca, legendpos = “topright”)
```

```
Data:   X dimension: 12 4
        Y dimension: 12 1
Fit method: svdpc
Number of components considered: 1
TRAINING: % variance explained
          1 comps
X           95.45
.outcome    82.35



Principal Component Analysis


12 samples

 4 predictor


No pre-processing

Resampling: Cross-Validated (5 fold)

Summary of sample sizes: 9, 10, 9, 10, 10

Resampling results across tuning parameters:


  ncomp  RMSE       Rsquared   MAE

  3      0.5858988  0.8958715  0.5641196

  2      0.6209459  0.8789823  0.6013179

  3      0.7076292  0.7833972  0.6865434


RMSE was used to select the optimal model using the smallest value.

The final value used for the model was ncomp = 1.
```



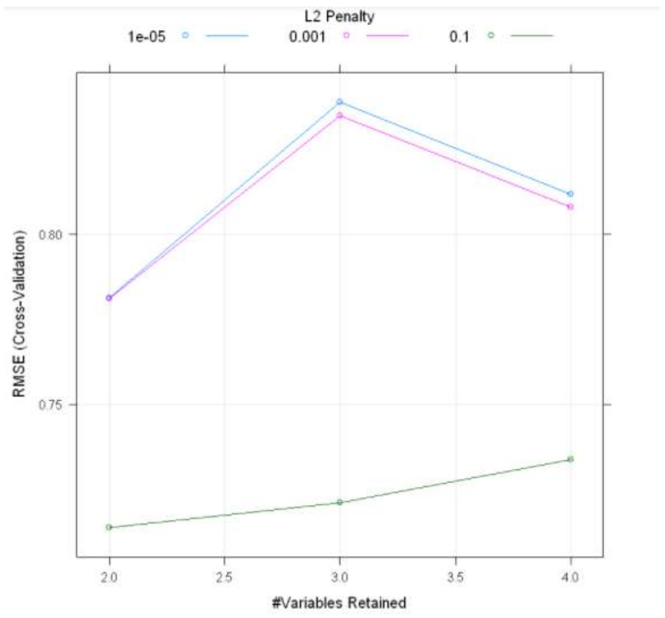
The selected model is only with one component, at the lowest RMSE.

Regression with Ridge:

- This model was chosen because it is very good when modeling data that have Multicollinearity issue, as the Least Squares will be unbiased, and it performs L2 regularization.
- The penalty term to be used is Lambda which will be selected with CV.

```
### Linear Regression with Ridge (foba in caret Package) ###
set.seed(1234)
lm_ridge  <- train(Grade ~., data = training, method = "foba",trControl =
train.control,metric="RMSE")

#Model Results
lm_ridge
print(lm_ridge$results)
# Plot
plot(lm_ridge)
```

```
Ridge Regression with Variable Selection

12 samples
 4 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 9, 10, 9, 10, 10
Resampling results across tuning parameters:

  lambda  k  RMSE       Rsquared   MAE
  1e-05   2  0.7811744  0.8451710  0.7369248
  1e-05   3  0.8387717  0.8723170  0.7860222
  1e-05   4  0.8117191  0.8853286  0.7664152
  1e-03   2  0.7810823  0.8453225  0.7369906
  1e-03   3  0.8347256  0.8733583  0.7831843
  1e-03   4  0.8078189  0.8865369  0.7638156
  1e-01   2  0.7137816  0.8392435  0.6859359
  1e-01   3  0.7210107  0.8328661  0.6942320
  1e-01   4  0.7336794  0.8328661  0.7076939

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were k = 2 and lambda = 0.1.
```



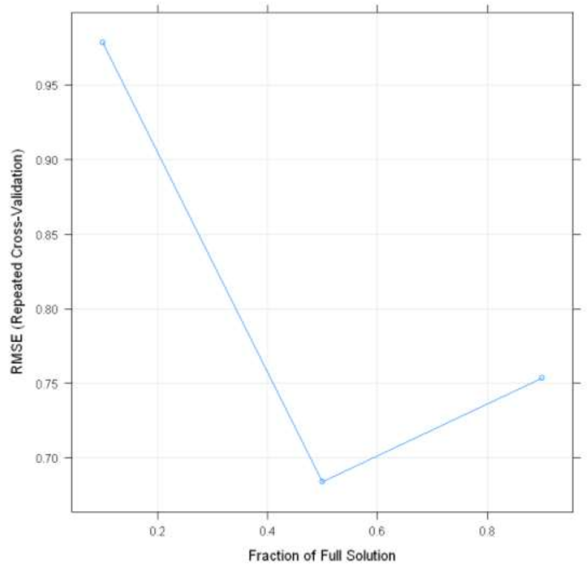
The best hyperparameters with lowest RMSE, chosen by cross validation are K=2 with Lambda=0.1

Regression with Lasso:

It is also a regularization technique, called Absolute Shrinkage and Selection Operator, this model uses shrinkage that shrinks the coefficient values of predictors to zero, and it helps to reduce over-fitting in the model and helps for feature selection sometimes.
The hyper-parameter tuned by caret is the fraction of the maximum L1 norm.

```
### Linear Regression with Lasso ###

set.seed(1234)
lm_lasso  <- train(Grade ~., data = training, method = "lasso",trControl =
train.control,metric="RMSE")

#Model Results
lm_lasso
print(lm_lasso$results)

# Plot
plot(lm_lasso)
```

```
12 samples
 4 predictor

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 9, 10, 9, 10, 10, 10, ...
Resampling results across tuning parameters:

  fraction  RMSE       Rsquared   MAE
  0.1       0.9781487  0.8916050  0.8590996
  0.5       0.6842426  0.8888315  0.6078015
  0.9       0.7535245  0.8931547  0.6826628

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was fraction = 0.5.
```



The best hyperparameter with lowest RMSE, chosen by cross validation are L1 norm fraction= 0.5

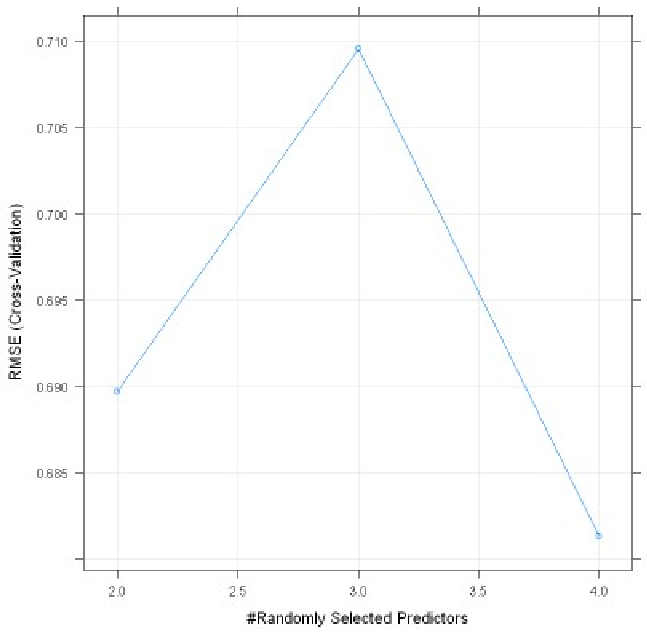
Random Forest:

This model uses bagging technique, it works by constructing multiple decision trees. The tuning parameter to be chosen using CV is *mtry* which is the number of variables randomly sampled as candidates at each split.

```
### Random Forest ###
set.seed(1234)
rf    <- train(Grade ~., data = training, method = "rf",tuneLength = 15, trControl =
train.control, metric="RMSE")

#Model Results
print(rf)
print(rf$results)
```

```
note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .

Random Forest

12 samples
 4 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 9, 10, 9, 10, 10
Resampling results across tuning parameters:

  mtry  RMSE       Rsquared   MAE
  2     0.6897125  0.8681549  0.6297687
  3     0.7095357  0.8574068  0.6291041
  4     0.6813445  0.8857311  0.5954092

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 4.
  mtry      RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
1    2 0.6897125 0.8681549 0.6297687 0.5078359  0.1949366 0.5151210
2    3 0.7095357 0.8574068 0.6291041 0.4911151  0.2027258 0.5029080
3    4 0.6813445 0.8857311 0.5954092 0.4873341  0.1882133 0.4977881
```

The best mtry with lowest RMSE, chosen by cross validation is mtry=4



Predict models on Test Dataset:

Here we are predicting the trained models using unseen data (testing), but frankly we are doing this step just out of curiosity, because our data is very small in general, especially for testing, as its not representative in my opinion, so we are not confident about the accuracy, but it is always good idea to assess the models and see the results!

```
### PREDICTIONS ###

pred_lm     = predict.train(lm_full,   newdata = testing, metric="RMSE")
pred_step   = predict.train(step_aic,  newdata = testing, metric="RMSE")
pred_back   = predict.train(back_aic,  newdata = testing, metric="RMSE")
pred_fwd    = predict.train(fwd_aic,   newdata = testing, metric="RMSE")
pred_pca    = predict.train(pca,       newdata = testing, metric="RMSE")
pred_ridge  = predict.train(lm_ridge,  newdata = testing, metric="RMSE")
pred_lasso  = predict.train(lm_lasso,  newdata = testing, metric="RMSE")
pred_rf     = predict.train(rf,        newdata = testing, metric="RMSE")
```

Summarizing metrics from Training and Testing for Model Comparison:

Metrics for Training:

Building the results table to include RMSE and Rsquared:

```
metrics <- function(results)
{
  row = c(results$RMSE,
          results$Rsquared)
}
```

Gathering all final best models results:

```
Full_LM       = metrics(lm_full$results)
Step_AIC      = metrics(step_aic$results)
Back_AIC      = metrics(back_aic$results[1,])
FWD_AIC       = metrics(fwd_aic$results[1,])
PRC           = metrics(pca$results[1,])
PLS           = metrics(pls$results[1,])
Ridge         = metrics(lm_ridge$results[3,])
Lasso         = metrics(lm_lasso$results[2,])
Random_Forest = metrics(rf$results[3,])
```

Combine all results into table to evaluate:

```
model_results = rbind(Full_LM,
                      Step_AIC,
                      Back_AIC,
                      FWD_AIC,
                      PRC,
                      PLS,
                      Ridge,
                      Lasso,
                      Random_Forest)

colnames(model_results) = c("RMSE", "Rsquared")
model_results
```

Using *resamples()* function to plot the RMSE for models with resamples to detect the variance across folds :

```
train_results <- resamples(list( "Full_LM"        =lm_full,
                                 "Step_AIC"       =step_aic,
                                 "Back_AIC"       =back_aic,
                                 "FWD_AIC"        =fwd_aic,
                                 "PRC"            =pca,
                                 "PLS"            =pls,
                                 "Ridge"          =lm_ridge,
                                 "Lasso"          =lm_lasso,
                                 "Random_Forest" =rf))
#dev.off()
dotplot(train_results, metric = "RMSE")
```

Metrics for Testing:

```
 lm_fullp   =   RMSE(pred_lm,    testing$Grade)
 step_aicp  =   RMSE(pred_step,  testing$Grade)
 back_aicp  =   RMSE(pred_back,  testing$Grade)
 fwd_aicp   =   RMSE(pred_fwd,   testing$Grade)
 pcap       =   RMSE(pred_pca,   testing$Grade)
 lm_ridgep  =   RMSE(pred_ridge, testing$Grade)
 lm_lassop  =   RMSE(pred_lasso, testing$Grade)
 rfp        =   RMSE(pred_rf,    testing$Grade)
```

```
pred_RMSE <- list(lm_fullp   =   RMSE(pred_lm,    testing$Grade),
                  step_aicp  =   RMSE(pred_step,  testing$Grade),
                  back_aicp  =   RMSE(pred_back,  testing$Grade),
                  fwd_aicp   =   RMSE(pred_fwd,   testing$Grade),
                  pcap       =   RMSE(pred_pca,   testing$Grade),
                  lm_ridgep  =   RMSE(pred_ridge, testing$Grade),
                  lm_lassop  =   RMSE(pred_lasso, testing$Grade),
                  rfp        =   RMSE(pred_rf,    testing$Grade))

pred_RMSE
```

```
$lm_fullp   0.916721357122357
$step_aicp  0.834594513658442
$back_aicp  0.917202128921969
$fwd_aicp   0.917202128921969
$pcap       0.906846509902378
$lm_ridgep  0.923777820587183
$lm_lassop  0.827429261616814
$rfp        0.972887768381985
```

▪ Conclusion:

Method	RMSE	RMSE_Predicted (RMSEP)
Full_LM	0.81176	0.916721357
Step_AIC	0.728564	0.834594514
Back_AIC	0.764359	0.917202129
FWD_AIC	0.722677	0.917202129
PRC	0.585899	0.90684651
Ridge	0.713782	0.923777821
Lasso	0.701673	0.827429262
Random_Forest	0.681345	0.972887768

- As per implemented strategy that adopted trying to explain response variable (Grade), we have constructed multiple models with K-folds CV and chosen the best one based on lowest RMSE.

- The most important variable that explains the response variable and generate the best model, is the predictor (Acid), as it was nominated by *ols_best_subset*() function and Backward, forward as well.

- The best Training model with lowest RMSE is Principle Component Analysis (RMSE = 0.585899) with (RMSEP = 0.90684651), which is an indicator of overfitting, as a result will be ignored.

- The best Predicted model with lowest RMSEP is Regression with Lasso (RMSEP = 0.827429262) with (RMSE = 0.701673), which we can consider it somehow good fit. And we can choose it a sour hero model.

- It is hard to decide which model is the best because the number of observations in the dataset is small to build good robust models with good predictions, even though we used K-fold cross validation, which is a reliable technique, but the data scarcity makes it hard to candidate the best one without sacrificing R squared or prediction accuracy for example.