Banknote Authentication System using Machine Learning

By: Sherif Allam

Contents

1 Introduction

Banknotes are one of the most important assets of a country. Some criminals introduce fake notes which bear a resemblance to original note to create discrepancies of the money in the financial market. It is difficult for humans to tell true and fake banknotes apart especially because they have a lot of similar features. Fake notes are created with precision, hence there is need for an efficient algorithm which accurately predicts whether a banknote is genuine or not.[1]

This document aims to explain building a Banknote Authentication System using Machine Learning algorithms.

Before Begin, let us have a look what is the business problem. Despite a decrease in the use of currency due to the recent growth in the use of electronic transactions, cash transactions remain very important in the global market. Banknotes are used to carry out financial activities. To continue with smooth cash transactions, entry of forged banknotes in circulation should be preserved. There has been a drastic increase in the rate of fake notes in the market. Fake money is an imitation of the genuine notes and is created illegally for various motives. These fake notes are created in all denominations which brings the financial market of the country to a low level. The various advancements in the field of scanners and copy machines have led the miscreants to create copies of banknotes. It is difficult for human-eye to recognize a fake note because they are created with great accuracy to look alike a genuine note. Security aspects of banknotes have to be considered and security features are to be introduced to mitigate fake currency. Hence, there is a dire need in banks and ATM machines to implement a system that classifies a note as genuine or fake.[1]

To drop the light on the business problem, let us take an example on the volume of fake notes in US market. There's more than $147 million in fake U.S. currency circulating globally these days. CNBC reported in 2015.[2]

To explain how Machine Learning algorithms could be used in Banknote Authentication Process. First of all, it is required to have an applied case with historical transactional data.

1.1 Goal of the Project

The Project goal is to indicate how machine learning could be very valuable and accurate solution in detecting fake Banknote and to show how machine learning algorithms are capable to differentiate between genuine banknote and fake banknote.

For our project purpose, Banknote Authentication dataset provided by University of California, Irvine is selected as a case study dataset. The dataset is also available on OpenML.

1.2 Dataset overview

As per the dataset author of the dataset, Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images.

Wavelet Transform is used to transform images to distribution of wavelets. Details of the Wavelet transform can be found here and a sample application for Image Wavelet Transform tool can be downloaded from here

Before start training the selected Machine Learning algorithms on the Banknote dataset. It is helpful to describe our data and discover any patterns that could help in customizing Machine Learning algorithms to fit our data correctly.

As we have now Banknote dataset downloaded, let us start by loading dataset into our environment and exploring the dataset columns:

```
# # Explore dataset structure

# Load Data from CSV file

raw_data <- read.csv("data_banknote_authentication.csv", header = FALSE)
```

```r
# Set Proper Columns names

colnames <- c("Var_Wave_Trans", "Skw_Wave_Trans", "Cur_Wave_Trans", "Entro_Image",
    "Class")
colnames(raw_data) <- colnames

# Check Columns Data structure

str(raw_data, 4, 4.4)
```

```
## 'data.frame':    1372 obs. of  5 variables:
##  $ Var_Wave_Trans: num  3.6216 4.5459 3.866 3.4566 0.3292 4.3684 ...
##  $ Skw_Wave_Trans: num  8.666 8.167 -2.638 9.523 -4.455 9.672 ...
##  $ Cur_Wave_Trans: num  -2.807 -2.459 1.924 -4.011 4.572 -3.961 ...
##  $ Entro_Image   : num  -0.447 -1.4621 0.1065 -3.5944 -0.9888 -3.1625 ...
##  $ Class         : int  0 0 0 0 0 0 0 0 0 0 0 ...
```

The Banknote dataset has 1372 observations and five columns as the following:

- Variance of Wavelet Transformed image (continuous)

Variance finds how each pixel varies from the neighboring pixels and classifies them into different regions.

- Skewness of Wavelet Transformed image (continuous)

Skewness is the measure of the lack of symmetry.

- Curtosis of Wavelet Transformed image (continuous)

Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.

- Entropy of image (continuous)

Image entropy is a quantity which is used to describe the amount of information which must be coded for, by a compression algorithm.

- Class (integer)

Class contains two values 0 representing genuine note and 1 representing fake note.

We can see sample Data as the following:

```r
# # Explore dataset rows

kable(head(raw_data),"latex",booktabs = T) %>% kable_styling(position = "center")
```

Var_Wave_Trans	Skw_Wave_Trans	Cur_Wave_Trans	Entro_Image	Class
3.62160	8.6661	-2.8073	-0.44699	0
4.54590	8.1674	-2.4586	-1.46210	0
3.86600	-2.6383	1.9242	0.10645	0
3.45660	9.5228	-4.0112	-3.59440	0
0.32924	-4.4552	4.5718	-0.98880	0
4.36840	9.6718	-3.9606	-3.16250	0

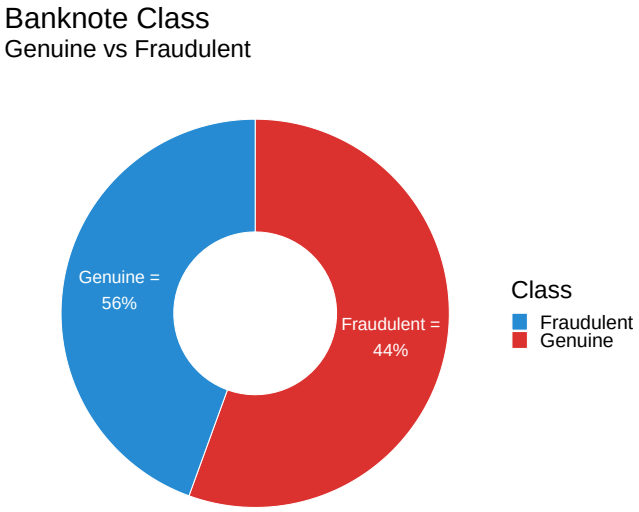
Each row represents a distribution properties for each image starting by Variance, Skewness and Curtosis while the entropy is the average information of an image can be determined approximately from the histogram of the image[3].

Next, Let us see how many genuine and how many fraudulent our dataset has. To do so let us convert 0 and 1 in Class variable to a descriptive values.

```
# Map 0 values to Genuine and 1 to Fraudulent

raw_data$Class <- ifelse(raw_data$Class==0,"Genuine", "Fraudulent")
```

We can use Donut Chart to visualize the percentage between Genuine and Fraudulent banknotes.

```
# First let us group by Class

Percent_table <-  raw_data %>% group_by(Class) %>% summarise(Count=n())

# Plot using Pie Chart and make a hole in the middle to convert it to Dount

Percent_table %>% ggplot( aes(x = 2, y = Count, fill = Class)) +
  geom_bar(stat = "identity", color = "white") +
  coord_polar(theta = "y", start = 0)+
  geom_text(aes(label = Count),position = position_stack(vjust = 0.5),
  label = paste0(Percent_table$Class, " =\n",
  round(Percent_table$Count/nrow(raw_data),2)*100, '%'),
  color= 'white',size=6.5) +
  theme_void()+
  theme(text = element_text(size=25))+ xlim(0.9, 2.5)+
  ggtitle("Banknote Class")+
  labs(subtitle ="Genuine vs Fraudulent")
```



We have almost a balanced dataset.

As our features are all in numbers, Hence missing values will be a challenge. Let us check whether our data has any missing values or not.

```
# Check for Missing Values

kable(apply(raw_data, 2, function(x) sum(is.na(x))),"latex",booktabs = T) %>%
 kable_styling(position = "center")
```

	x
Var_Wave_Trans	0
Skw_Wave_Trans	0
Cur_Wave_Trans	0
Entro_Image	0
Class	0

That is good news, as we do not have any NA values.

As we don't have missing values, let us check if all features affecting the classification of the banknote or not.

To check the relation between each feature and banknote class. Let us do a quick visualization to the each feature value ranges and the corresponding banknote class.

The most powerful graph to demonstrate that relation is the Boxplot.

To link between each feature value ranges and the corresponding banknote class, we need to gather our data by features, hence we will have table with three columns as the following:

- Banknote Class
- Feature Name
- Feature Value

Let us gather our Data.

```
# Gather Dataset to have all Features as rows

gathered_raw_data <- raw_data %>% gather(Feature, Value, -Class)
```

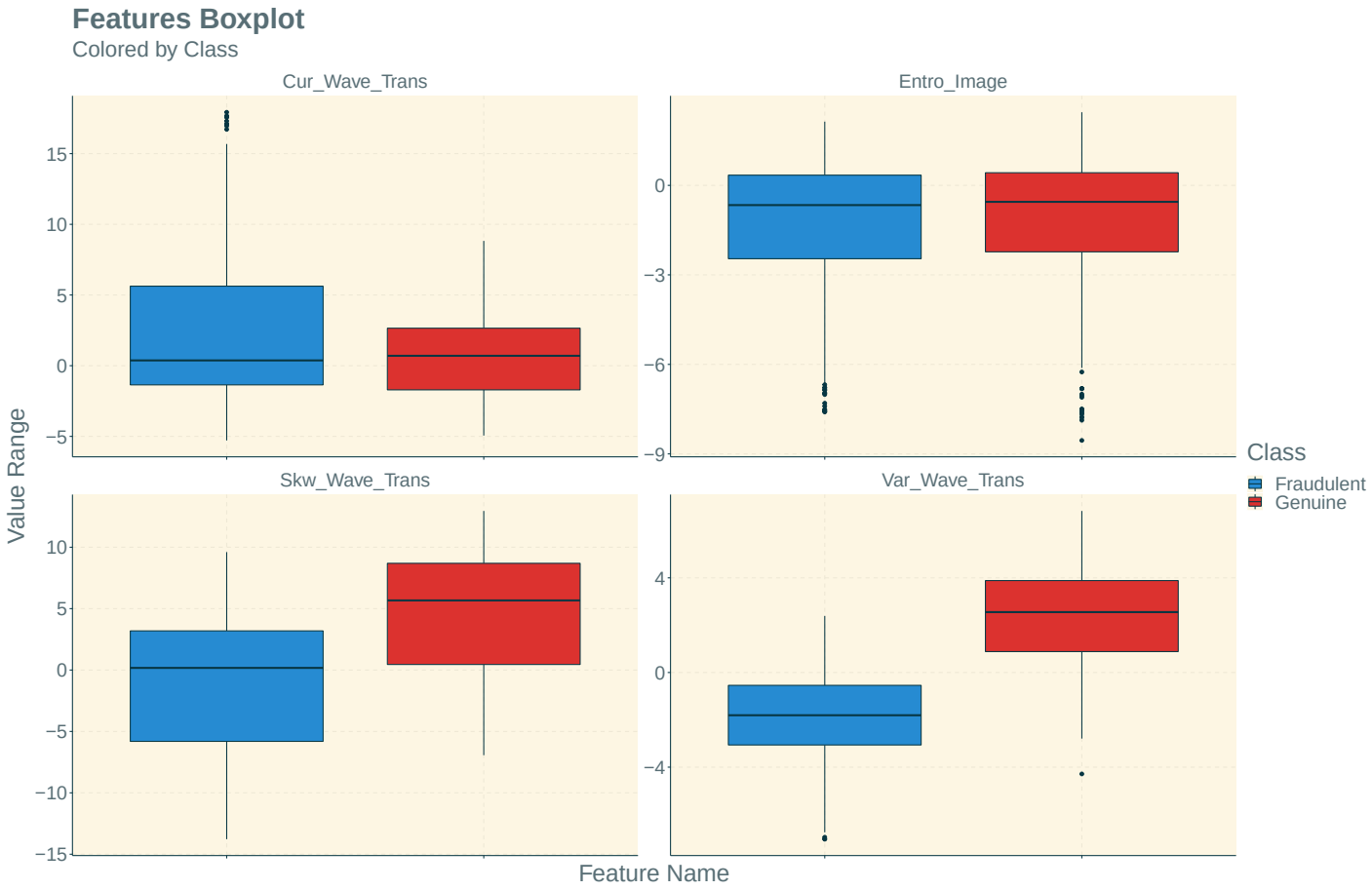
Now, let have a look on how our data look after gathering.

```
# Explore Data set after gathering Features

kable(head(gathered_raw_data),"latex",booktabs = T) %>%
 kable_styling(position = "center")
```

Class	Feature	Value
Genuine	Var_Wave_Trans	3.62160
Genuine	Var_Wave_Trans	4.54590
Genuine	Var_Wave_Trans	3.86600
Genuine	Var_Wave_Trans	3.45660
Genuine	Var_Wave_Trans	0.32924
Genuine	Var_Wave_Trans	4.36840

Now, It is time to Plot the Boxplot.

```
# Plot Features Ranges colored by Banknote Class

gathered_raw_data %>% ggplot(aes(Class, Value, fill = Class)) +
geom_boxplot() +
facet_wrap(~Feature, scales = "free") +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=25))+
ggtitle("Features Boxplot")+
labs(subtitle ="Colored by Class", x="Feature Name" ,
y="Value Range")
```



Now, we have a more clear idea about our features distribution and the relation between each feature and banknote Class.

Form the previous simple Boxplot graph, we can build an intuition that:

- Var_Wave_Trans is the most important feature, as more than 2/3 of its values which are over Zero identify Genuine Banknote Class. On the other side the 2/3 of its values which are over Zero identify Fraudulent Banknote Class.
- Skw_Wave_Trans is the second important feature as the range common values between Genuine and Fraudulent is not so big.
- While Cur_Wave_Trans and Entro_Image have almost all values common between Genuine and Fraudulent.

1.3 Summarize/ Key Steps

First, we will do a detailed analysis for the dataset to examine each variables statistically. Second, we will check the relation between variables and how it is related to determine the Banknote Class. Next, we will use Machine Learning algorithms to predict Banknote Class.

The following Machine Learning algorithms will be used.

- Supervised Machine learning
  - Logistic Regression
  - K-nearest neighbors

- Support Vector Machine
- Random Forest
- Neural networks

Finally, we will measure each algorithm performance. Considering that we have two classes. Hence, we will measure each algorithm accuracy by looking at different measures like Sensitivity, Specificity, ROC and F1 score.

2 Methods/Analysis

As we discussed in the Introduction section, there are three main assumptions or intuitions that we have assumed based on preliminary exploration of the dataset. Let us now take a deep dive to the data to validate our intuitions and assumptions.

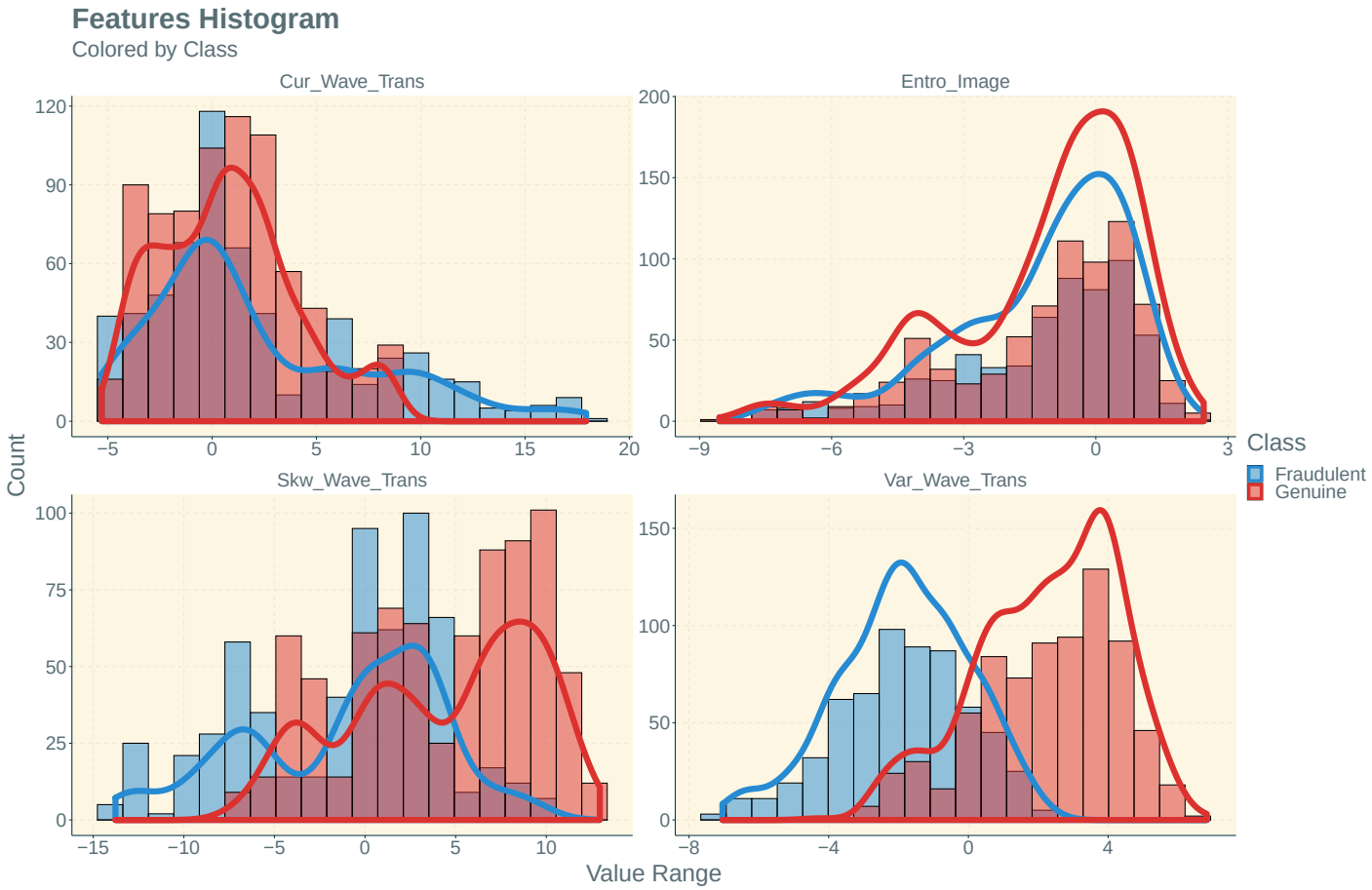
2.1 Data Exploration/ Insights

Now let us get deep inside our data and check each Feature in detail and the relations between our Features.

2.1.1 Univariate Analysis

For more clarification about data characteristics let us see the distribution of the features.

```
# Plot Histogram and Density Plots for all Features

gathered_raw_data %>% ggplot(aes(Value,y = ..count.., fill = Class, color=Class)) +
  geom_histogram(bins = 20, aes(color = Class, fill = Class),
  position = "identity", color = "black",alpha = 0.5 )+
  geom_density(alpha = 0,aes(color = Class), size =3)+
  facet_wrap(~Feature, scales = "free") +
  theme(text = element_text(size=25))+
  ggtitle("Features Histogram")+
  labs(subtitle ="Colored by Class", x="Value Range" ,
  y="Count")
```



It is obvious now, that our initial assumption that Var_Wave_Trans with range more than Zero is most of the time classified as Genuine where the values less than Zero are considered as Fraudulent.

Let us now have a more clear view on our features values through scattering all values.

```
# Replicate Index Sequence for the each feature

x= rep(seq(1,nrow(raw_data)),4)

# Randaomize our data, as orginal data was sorted is sorted by class

gathered_raw_data_random <- gathered_raw_data[sample(nrow(gathered_raw_data)),]

# Plot Scattered and Density Plots for all Features

gathered_raw_data_random %>% ggplot(aes(x=x, y=Value, fill = Class, color=Class)) +
geom_point()+
facet_wrap(~Feature, scales = "free") +
theme(text = element_text(size=25))+
ggtitle("Features Values")+
labs(subtitle ="Colored by Class", x="Value Index" ,
y="Value")
```

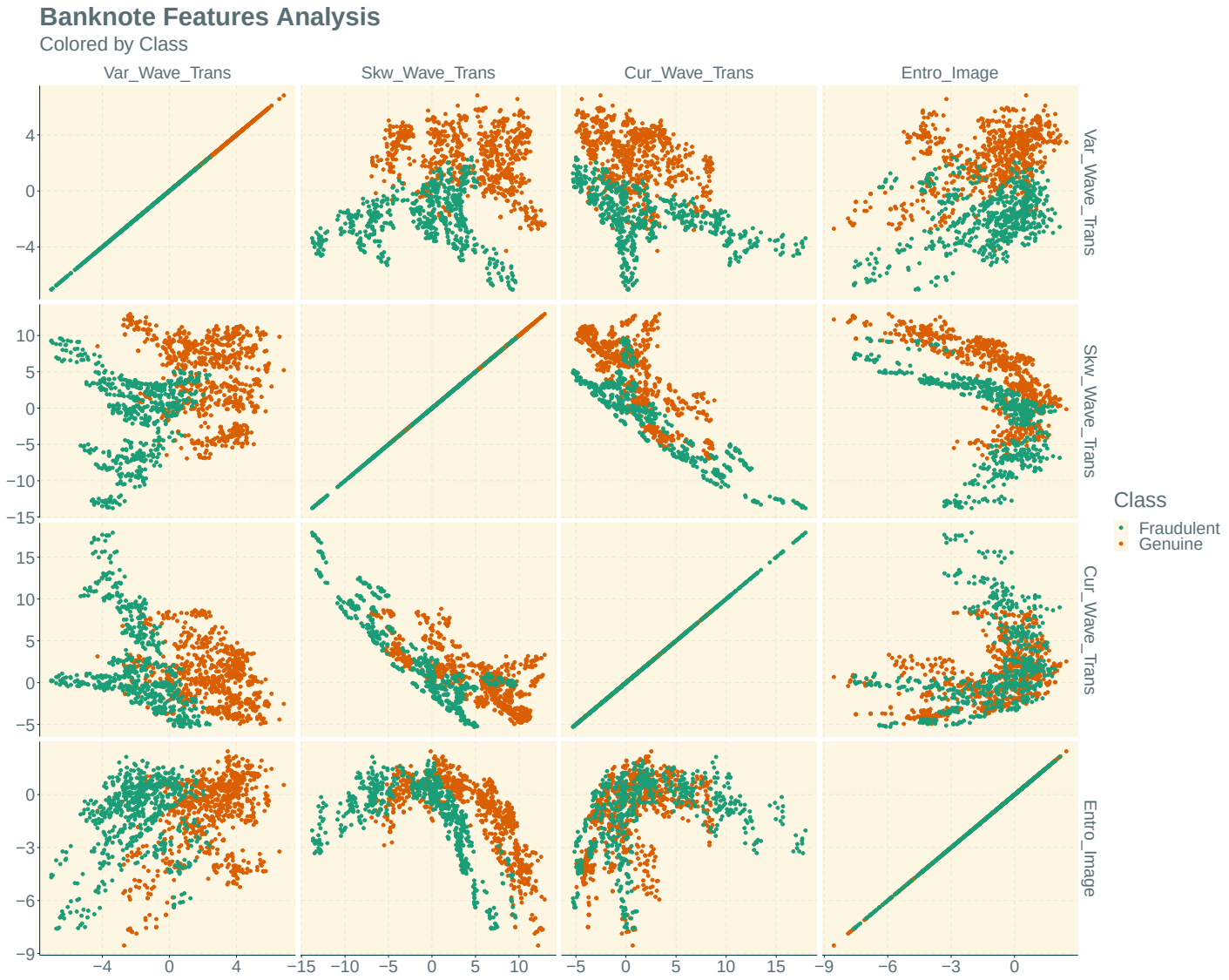
From here we can draw one more conclusion about Car_Wave_Trans, It is clear that values bigger 10 are all Fraudulent. For Skw_Wave_Trans, values less than-10 are all Fraudulent.

2.1.2 Multivariate Analysis

Here in that section, we will try to analyze Features from two perspectives First, the relation between each two features and the Banknote Class. Second, we will check correlations between features and check whether there any Feature combination that can help us identifying Banknote Class.

Hence, let us start by plotting each Feature against the other Features grouped by Banknote Class.

```
# Plot Relation Between all Features

PairPlot(raw_data,
         colnames(raw_data)[1:4],
         "Banknote Features Analysis",
         group_var = "Class")+
theme(text = element_text(size=25))+
labs(subtitle ="Colored by Class")
```

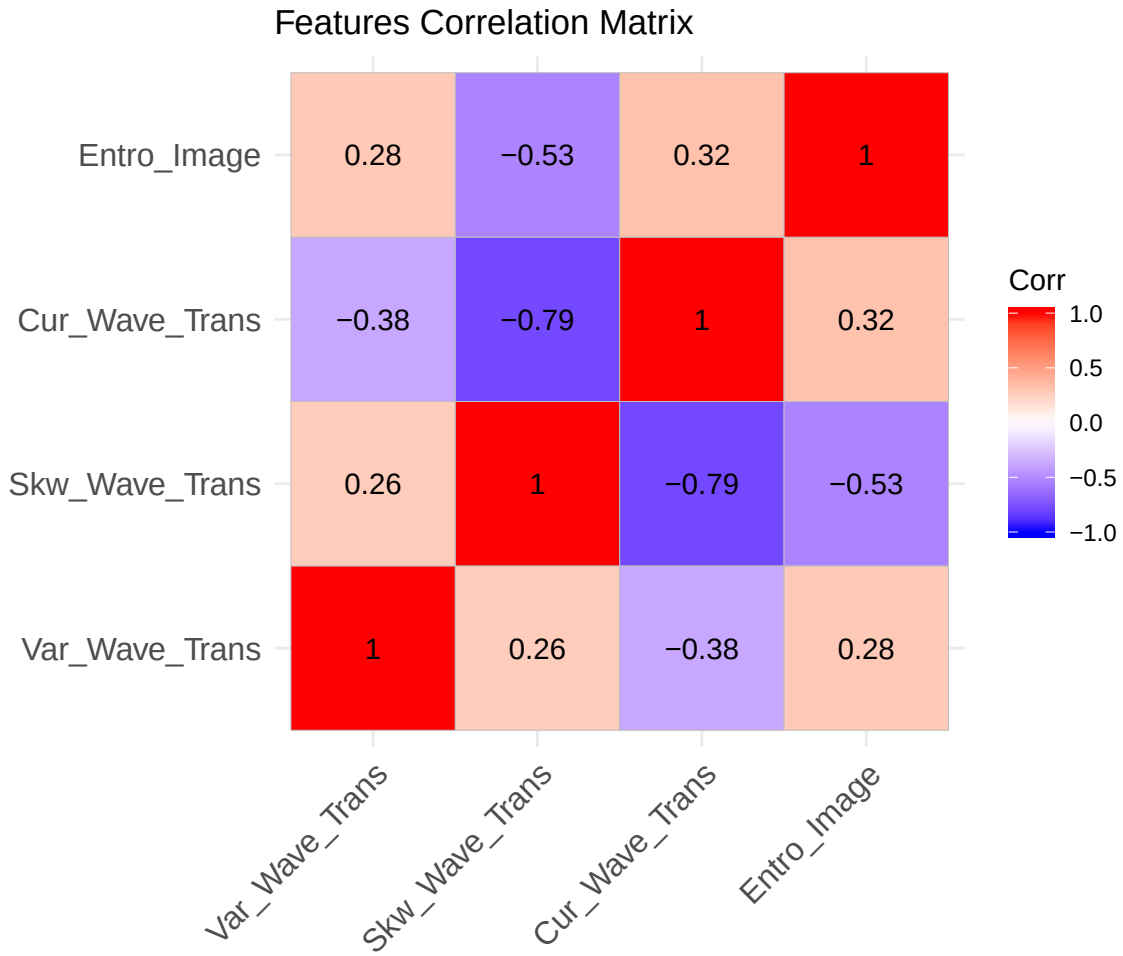


We have good news here, as some features combination separate two Banknote classes.

- In First Row and Second Column, Var_Wave_Trans and Skw_Wave_Trans plot indicates that we can draw a simple curve to separate most of “Genuine” and “Fraudulent” Classes.
- In First Row and Third Column, Var_Wave_Trans and Cur_Wave_Trans plot indicates that we can draw a simple slop line to separate most of “Genuine” and “Fraudulent” Classes.
- In First Row and Fourth Column, Var_Wave_Trans and Entro_Image plot indicates that we can draw a simple slop line to separate most of “Genuine” and “Fraudulent” Classes.
- In Second row and Third Column, Skw_Wave_Trans and Cur_Wave_Trans plot, it is somehow hard to separate “Genuine” and “Fraudulent” Classes.
- In Second row and Fourth Column, Skw_Wave_Trans and Entro_Image plot, it is somehow hard to separate “Genuine” and “Fraudulent” Classes.
- In Third row and Fourth Column, Cur_Wave_Trans and Entro_Image plot, it is too how hard to separate “Genuine” and “Fraudulent” Classes.

Let us now check correlations between Features, the easiest way is to plot correlations.

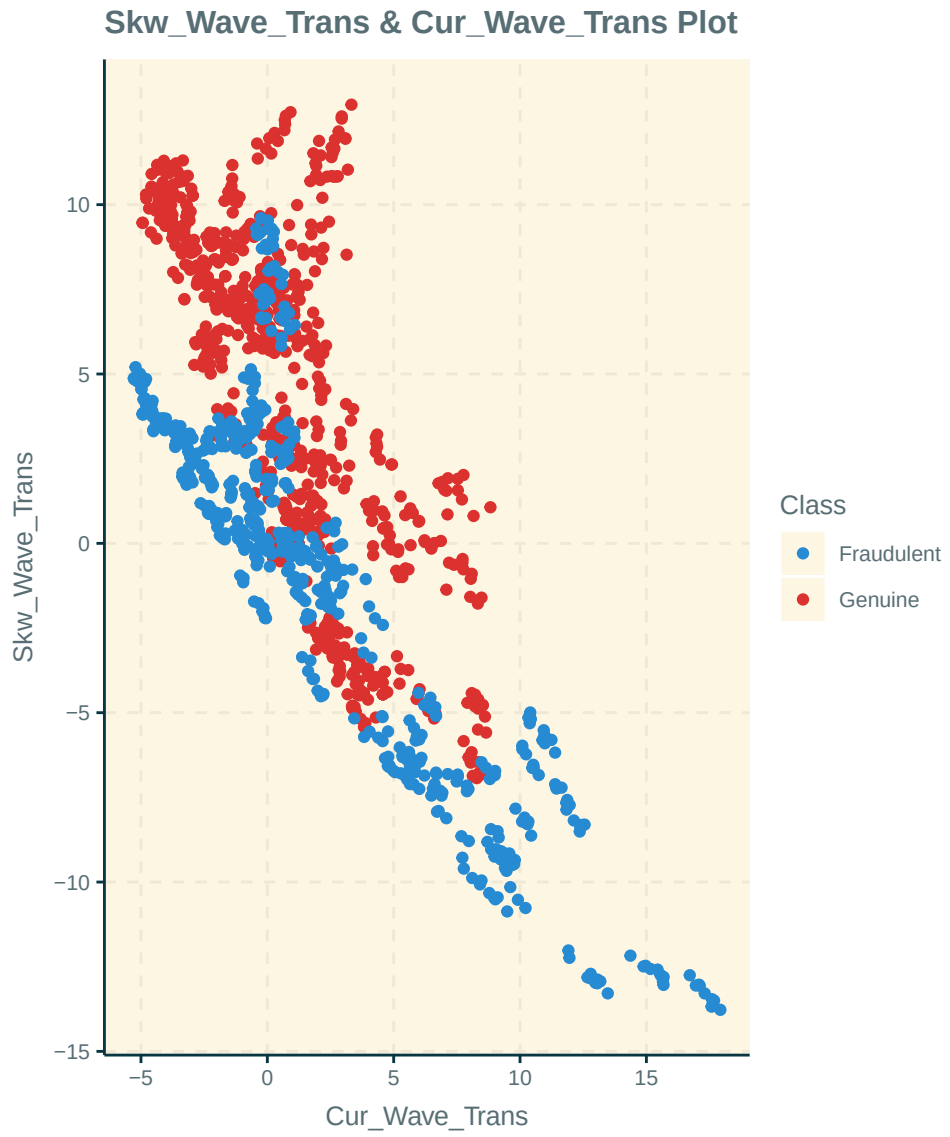
```
# Calculate Correlation between all Features

cor_data <- round(cor(raw_data[,1:4]),2)

# Plot Correlation as Matrix

ggcorrplot(cor_data,lab = TRUE)+
ggtitle("Features Correlation Matrix")
```



Now We can see that strongest correlation is between Skw_Wave_Trans and Cur_Wave_Trans which it was expected also from previous scattered plot.

```
# Plot Skw_Wave_Trans and Cur_Wave_Trans against each other
```

```
raw_data %>% ggplot(aes(x=Cur_Wave_Trans, y=Skw_Wave_Trans, fill = Class, color=Class)) +
geom_point()+
theme(text = element_text(size=10))+
ggtitle("Skw_Wave_Trans & Cur_Wave_Trans Plot")
```

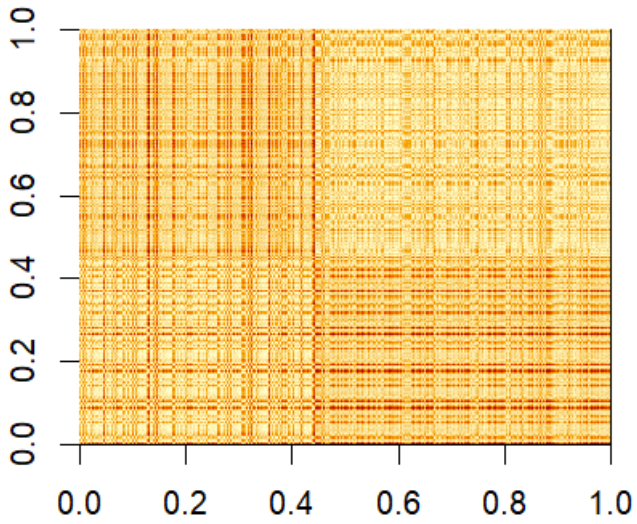


Hence, we can say If we neglect one of the correlated Features on Prediction of Banknote Class, we don't expect to loose much.

Let us validate that assumption by plotting distance using all features and then plotting distance using three features after neglecting Skw_Wave_Trans.

```
# Get Features all Features without Banknote Class

x <- raw_data[,1:4] %>% as.matrix()

# Calculate distance between all Observations
```

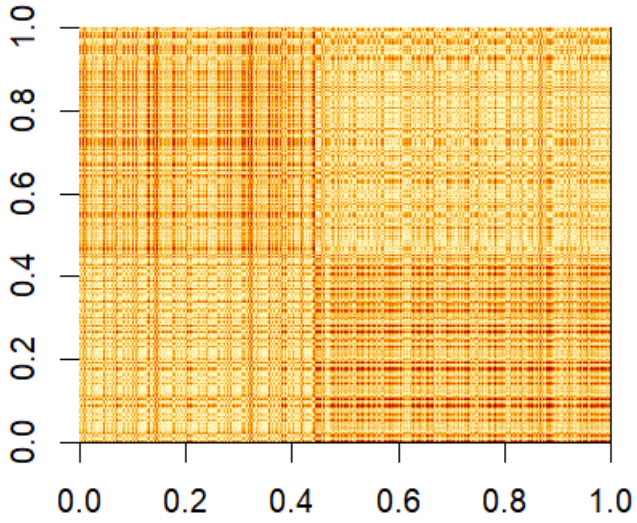
```
d <- dist(x)

# Plot Distance as Matrix  Image

image(as.matrix(d)[order(raw_data$Class), order(raw_data$Class)])
```



From the previous image, even if we don't know that we have two classes, we can conclude that. However, we can also say that the difference between the two classes is not that much as the coloring density is not so obvious.

Now, let us do the same task again after dropping Cur_Wave_Trans.

```
# Get Features all Features Except Cur_Wave_Trans and without Banknote Class

x <- raw_data[,c(1:2,4)] %>% as.matrix()

# Calculate distance between all Observations

d <- dist(x)

# Plot Distance as Matrix  Image

image(as.matrix(d)[order(raw_data$Class), order(raw_data$Class)])
```



We can see that we didn't lose the Classification as it is clear that we have two Classes.

But fortunately, the difference between the Classes is more clear now. As we can see the difference between two Classes color density is more obvious now.

2.1.3 Principal Component & Unsupervised Clustering Analysis

To get deep inside our dataset features and whether the data is really clustered by it self or not, let us use some data analysis techniques without using the known class from the original data. This analysis will help us to understand our data points or observations in a more comprehensive way.

Let us first prepare our data for the analysis by doing the following:

- Name our observations based on their class
- Rearrange our data rows in a random way
- Extract features without Class in a Matrix format
- Center and Scale features
- Name Matrix rows

```
# Give each raw a name based on raw Class

uns_raw_data <- raw_data %>%
  mutate(raw_name= ifelse(Class=="Genuine",
 paste0("G",row_number()),paste0("F",row_number()) ))

# Rearange our data rows in a random way
set.seed(1)
uns_raw_data <- uns_raw_data[sample(nrow(uns_raw_data)),]
```

```
# Get raw names
row_names <- uns_raw_data [,6]

# Convert data to a matrix for analysis purpose
uns_mtx_data <- uns_raw_data[,1:4] %>% as.matrix()

# Name matrix rows
rownames(uns_mtx_data) <- row_names

#Center and Scale features
scaled_mtx_data <- scale(uns_mtx_data, center = TRUE, scale = TRUE)
```

2.1.3.1 Principal Component Analysis

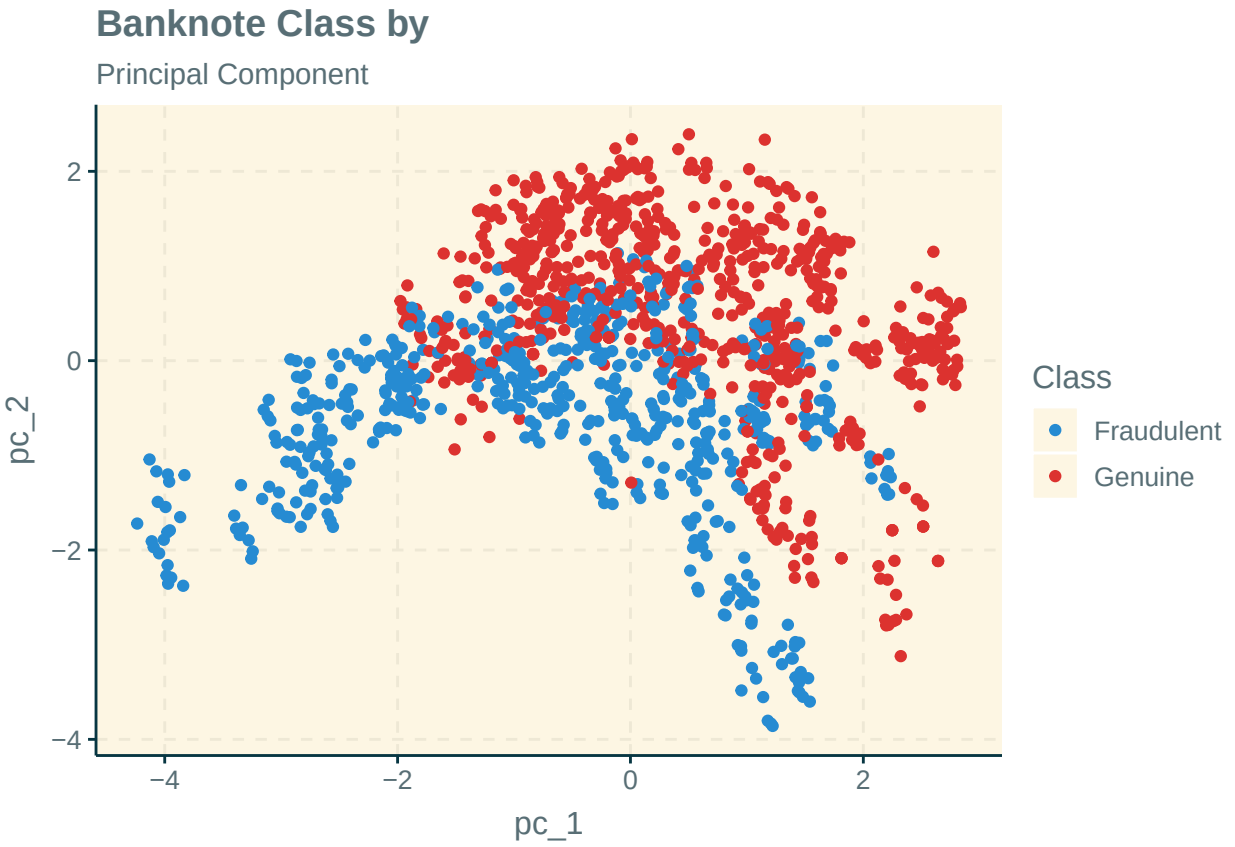
After preparing our data, we are ready to compute Principal Component as the following:

```
# Get Principal Component

pca <- prcomp(scaled_mtx_data,scale=FALSE,center = FALSE)
summary(pca)
```

```
## Importance of components:
##                           PC1   PC2    PC3    PC4
## Standard deviation     1.476 1.137 0.5928 0.4190
## Proportion of Variance 0.545 0.323 0.0878 0.0439
## Cumulative Proportion  0.545 0.868 0.9561 1.0000
```

we can see that our data is having a pattern of classification by itself, as it is clear from Principal Component Analysis that by 2 Principal Components only PC1 and PC2, we can achieve almost 92% of variability of our data. We can visualize that as well.

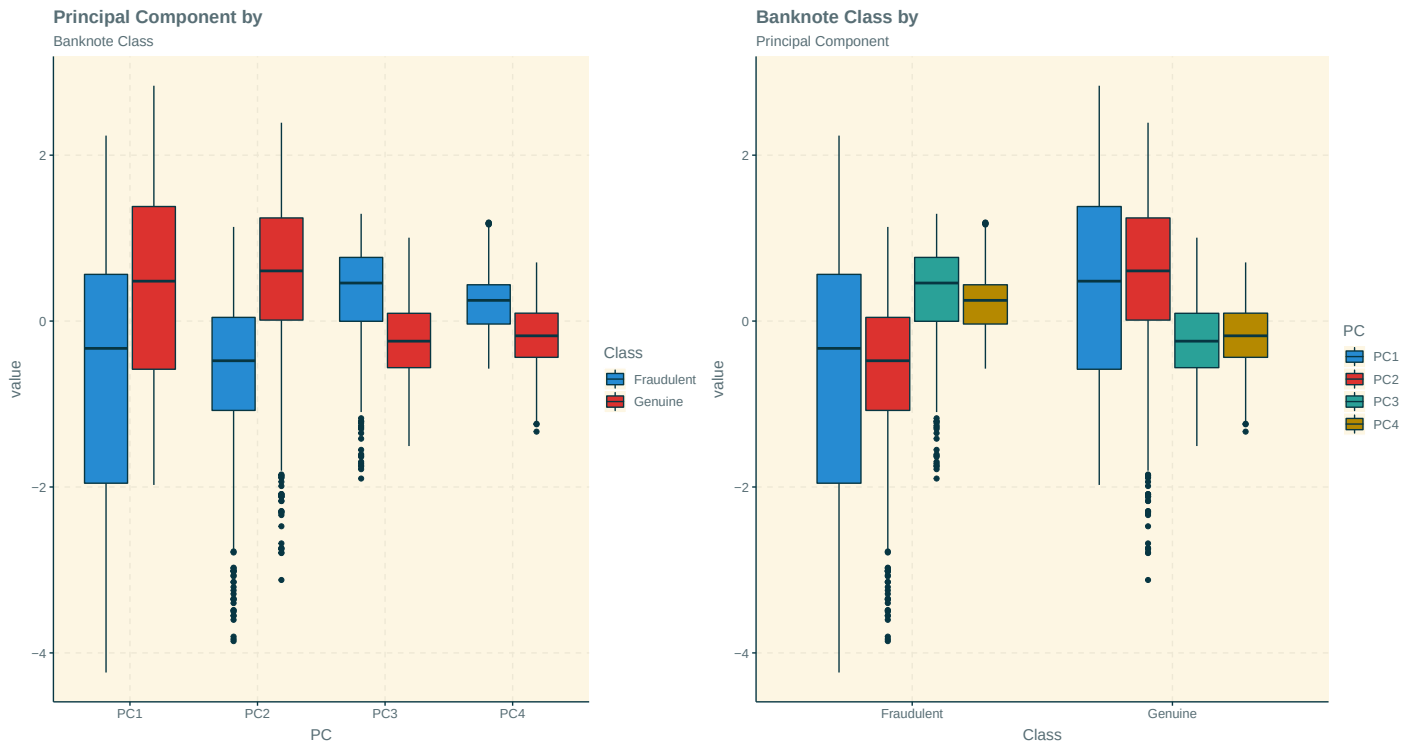
```
data.frame(pc_1 = pca$x[,1], pc_2 = pca$x[,2], Class = uns_raw_data$Class) %>%
  ggplot(aes(pc_1, pc_2, color = Class)) +
  geom_point()+
  theme(text = element_text(size=12))+
  ggtitle("Banknote Class by")+
  labs(subtitle ="Principal Component")
```



From the above graph, we can see that by applying some simple rules for PC1 and PC2, we can determine whether the point is Genuine or Fraudulent.

To have get more inside Principal Component, we can plot each Principal Component categorized by Class.

```
plot1 <- data.frame(Class = uns_raw_data$Class, pca$x[,1:4]) %>%
    gather(key = "PC", value = "value", -Class) %>%
    ggplot(aes(PC, value, fill = Class)) +
    geom_boxplot()+
    theme(text = element_text(size=12))+
    ggtitle("Principal Component by")+
    labs(subtitle ="Banknote Class")

plot2 <- data.frame(Class = uns_raw_data$Class, pca$x[,1:4]) %>%
    gather(key = "PC", value = "value", -Class) %>%
    ggplot(aes(Class, value, fill = PC)) +
    geom_boxplot()+
    theme(text = element_text(size=12))+
    ggtitle("Banknote Class by")+
    labs(subtitle ="Principal Component")

  grid.arrange(plot1, plot2, ncol=2)
```

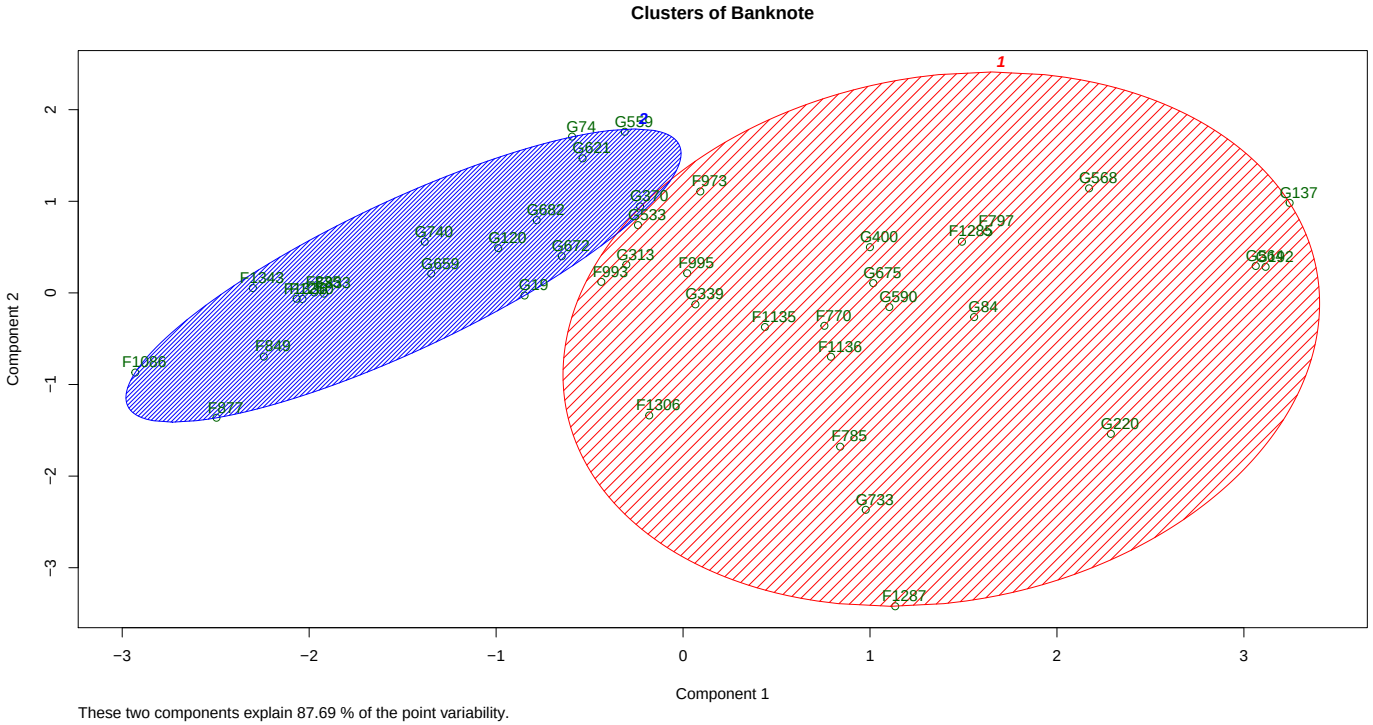


Now, It is more clear that Principal Component Number 2 and Number 3 can play a big rule in our data classification.

2.1.3.2 Hierarchical clustering Analysis

let us see whether our data could be clustered by itself or not.

```
set.seed(12)

# Get smallportion of the data

clust_index <- createDataPartition(y = uns_raw_data$Class, times = 1,
p = 0.03, list = FALSE)
clus_mtx_data <- scaled_mtx_data[clust_index,]

# Substrate raw mean
clus_mtx_data <- sweep(clus_mtx_data, 1, rowMeans(clus_mtx_data, na.rm = TRUE))

# Calculate Distance
d <- dist(clus_mtx_data)
# Get Hierarchical clustering
h <- hclust(d)
# Get Data as 2 Groups
groups <- cutree(h, k = 2)
# Plot cluster
clusplot(clus_mtx_data,groups,lines = 0, shade = TRUE, color = TRUE, labels= 2,
         plotchar = FALSE, span = TRUE, main = paste('Clusters of Banknote'))
```



We can see that our data has a pattern based on Hierarchical clustering, although that some points are miss classified.

2.1.3.3 k-means Analysis

In that section, we will apply k-means clustering algorithm to our data and compare the output to the real classes.

```
x <- sweep(uns_mtx_data, 2, colMeans(uns_mtx_data))
x <- sweep(uns_mtx_data, 1, rowMeans(uns_mtx_data))


k <- kmeans(x, centers = 2, nstart = 50)


table(k$cluster, uns_raw_data$Class)
```

```
##
##     Fraudulent Genuine
##   1        346     565
##   2        264     197
```

It seems based on k-means clustering that our data has a pattern as discovered group “1” represent Genuine class. On the other hand, group “2” represent Fraudulent class. although the accuracy is not good.

Now, we have a good insights about our data and we can proceed with Protection.

2.2 Building Training & Cross Validation and Test datasets

The main pain for all Machine Learning algorithms is overfitting, it happened when algorithm works with good performance on training set. However, when algorithm is tested on data that were not seen before, the performance got decreased heavily.

To avoid that, we will use cross validation technique and we will keep part of our data not used at all until prediction (test dataset).

Before proceeding with building training, validation and test datasets. Let us remember that, our original data has 1372 rows. Hence, we will use 80% of the data for training and validation and 20% for testing.

The 80% of the data, we will call it the training dataset. However, while implementing Machine Learning algorithms, we will use cross validation to split training set to training and cross validation datasets by 80% for training and 20% for validation.

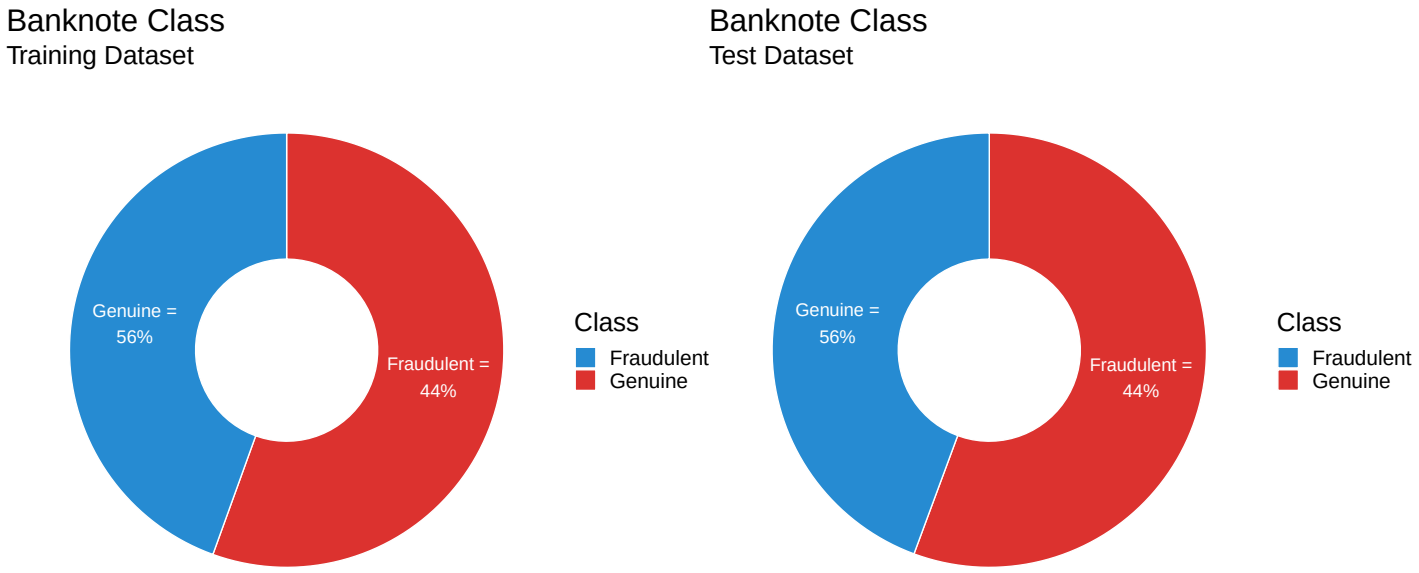
Now let us, separate our original data to training and testing datasets.

```
# Set seed to make sure getting the same result on next run
set.seed(1)

# Separate Data using Banknote Class as reference

test_index <- createDataPartition(y = raw_data$Class, times = 1,
p = 0.2, list = FALSE)

# Get Test and Training datasets

train_set <- raw_data[-test_index,]
test_set <- raw_data[test_index,]
```

Before going forward, let us make sure that Training and Testing datasets keep the balance between Genuine and Fraudulent Classes.

```
# First let us group by Class

train_Percent_table <-  train_set %>% group_by(Class) %>% summarise(Count=n())
test_Percent_table <-  test_set %>% group_by(Class) %>% summarise(Count=n())

# Plot using Pie Chart and make a hole in the middle to convert it to Dount


plot1 <- train_Percent_table %>% ggplot( aes(x = 2, y = Count, fill = Class)) +
  geom_bar(stat = "identity", color = "white") +
  coord_polar(theta = "y", start = 0)+
  geom_text(aes(label = Count),position = position_stack(vjust = 0.5),
  label = paste0(Percent_table$Class, " =\n",
  round(Percent_table$Count/nrow(raw_data),2)*100, '%'),
  color= 'white',size=5) +
  theme_void()+
  theme(text = element_text(size=20))+ xlim(0.9, 2.5)+
  ggtitle("Banknote Class")+
  labs(subtitle ="Training Dataset")

plot2 <- test_Percent_table %>% ggplot( aes(x = 2, y = Count, fill = Class)) +
  geom_bar(stat = "identity", color = "white") +
  coord_polar(theta = "y", start = 0)+
  geom_text(aes(label = Count),position = position_stack(vjust = 0.5),
  label = paste0(Percent_table$Class, " =\n",
  round(Percent_table$Count/nrow(raw_data),2)*100, '%'),
  color= 'white',size=5) +
```

```
  theme_void()+
  theme(text = element_text(size=20))+ xlim(0.9, 2.5)+
  ggtitle("Banknote Class")+
  labs(subtitle ="Test Dataset")

  grid.arrange(plot1, plot2, ncol=2)
```



2.3 Logistic Regression

Now, we can start with our first algorithm Logistic Regression.

2.3.1 Logistic Regression Training

Let us start by training using all Features. While Training, we will inform the Logistic Regression to use 80% of the data to Train and 20% to Validate and to repeat Picking process randomly 5 times.

```
# createFolds will return a list of 5 lists
# Each list will have 80% of randomly selected Indices from train_set


folds = createFolds(train_set$Class, 5, returnTrain = TRUE)

# Data of Selected 80% Indices will be used for Training our ML algorithm
# Training will be repeated for 5 times as we have 5 Folds
# Each Corresponding Remaining 20% Indices will be used for Validation
# Prediction will be done on Corresponding Validation set
```

```
# Algorithm will Return Class Probability
# Save Prediction on each Validation dataset will be used later

Control <-
  trainControl(
  index = folds,
  classProbs = T,
  savePredictions = T,
  summaryFunction = twoClassSummary
)

# Train Logistic model

glm_model <- train(Class ~ . , method = "glm", data= train_set,
   trControl = Control, family = "binomial")
```

2.3.2 Logistic Regression Tuning

In that section, we will check the following:

- Multicollinearity between Predictors
- Outliers in Predictors
- Linear relationship between outcome logit and Predictors
- Predictors/ Variables Importance
- Best Threshold or Cut-off

First, let us check whether our predictors suffer from Multicollinearity which is an important issue for Logistic Regression and should be fixed by excluding the correlated features.

```
# Multicollinearity Check

car::vif(glm_model$finalModel)
```

```
## Var_Wave_Trans Skw_Wave_Trans Cur_Wave_Trans    Entro_Image
##        70.2477       217.6390       419.2870         2.8488
```

The above results showus that our model is suffering from very high Multicollinearity predictors as expected while we were doing Exploration Data Analysis when we checked Feature correlations.

Hence, let us try train our model by excluding Cur_Wave_Trans Feature.

```
# Train Logistic model excluding Cur_Wave_Trans Featur

glm_model <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans + Entro_Image,
 method = "glm", data= train_set, trControl = Control, family = "binomial")
```

Now, let us check the Multicollinearity.

```
# Multicollinearity Check

car::vif(glm_model$finalModel)
```

```
## Var_Wave_Trans Skw_Wave_Trans    Entro_Image
##         1.3401         2.1945         2.0994
```

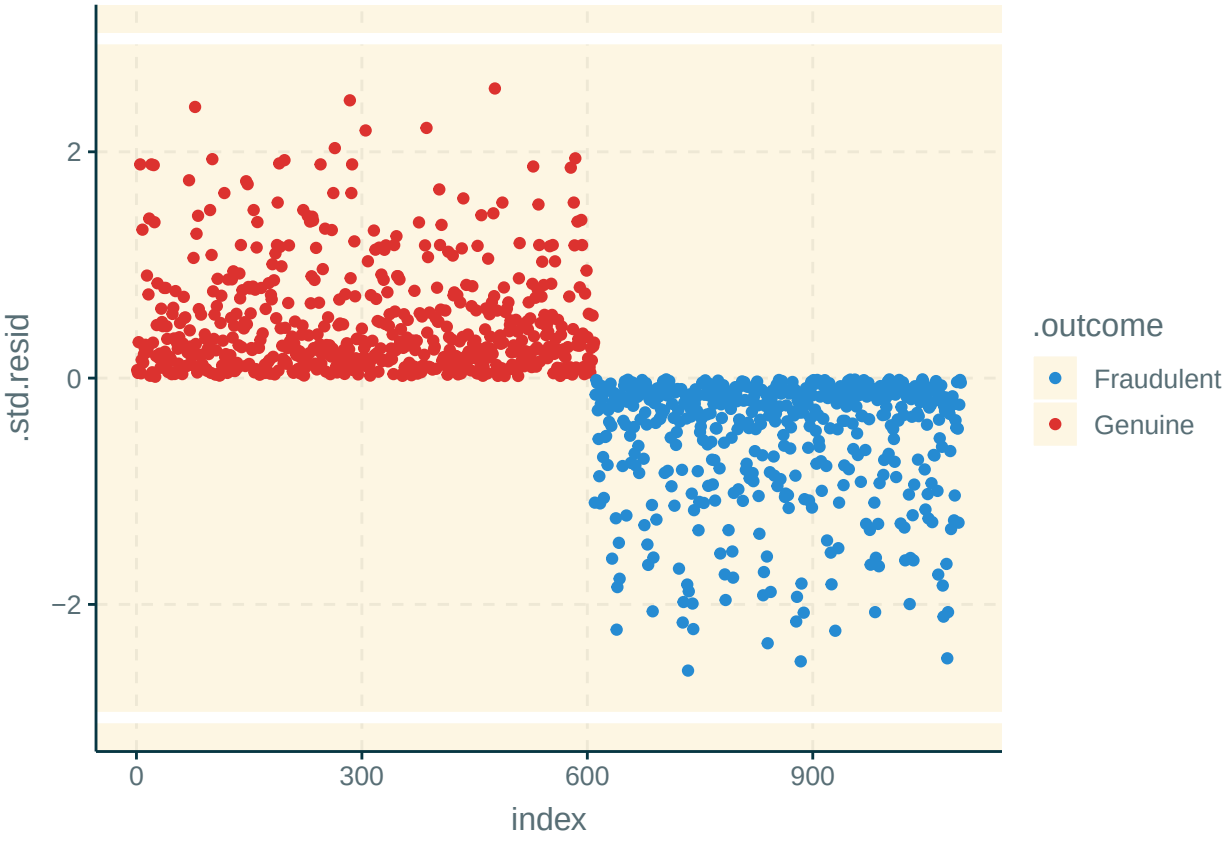
That is good improvement from Multicollinearity coefficient above 100 in the previous try with 4 Predictors comparing to what we have now with 3 Predictors

Second, we need to check whether there are Outliers in Predictors that statistically affects our Model. To do that , We have to check residuals deviation which is useful for determining whether the data contains potential influential observations.

Data points with a residual deviation above or below 3 standard deviation are possible outliers and it is better not to include them in training data.

Hence, let us plot the residuals of our logistic model.

```
# Get all observations residuals

model1_data <- augment(glm_model$finalModel) %>%  mutate(index = 1:n())


# Plot standard deviation for all residuals

ggplot(model1_data, aes(index, .std.resid)) +
geom_point(aes(color = .outcome))+
geom_ref_line(h = 3)+
geom_ref_line(h = -3)
```



That is good news. As per our plot, there are no standardized residuals that exceed 3 standard deviation which means we don't have possible outliers.

Third, let us check Linear relationship between logit outcome and each predictor. It will be easy by plotting relationship.

```
# Get probabilities for Training Set

probabilities <- predict(glm_model, type = "prob")

# Add probability of each observation

p_train_set <- train_set %>% mutate(prob = probabilities$Genuine)

# Keep only used predictors
p_train_set <- p_train_set[,c(-5,-3)]


# Calculate logit and add it to the dataframe

p_train_set <- p_train_set %>% mutate(logit = log(prob/(1-prob))) %>% select(-c(prob))%>%
gather(key = "Predictor", value = "Value",-logit)


# Plot Relation between the logit of the outcome and each predictor

ggplot(p_train_set, aes(logit, Value))+
geom_point(size = 0.5) +
```

```
geom_smooth(method = "loess") +
facet_wrap(~Predictor)+
ggtitle("Predictor Relation with Outcome logit")+
labs(subtitle ="Training Dataset")
```

We can see it is not perfect Linear relationship.

Fourth, let us check each predictor importance to the model.

```
# Check each Predictor importance
caret::varImp(glm_model)
```

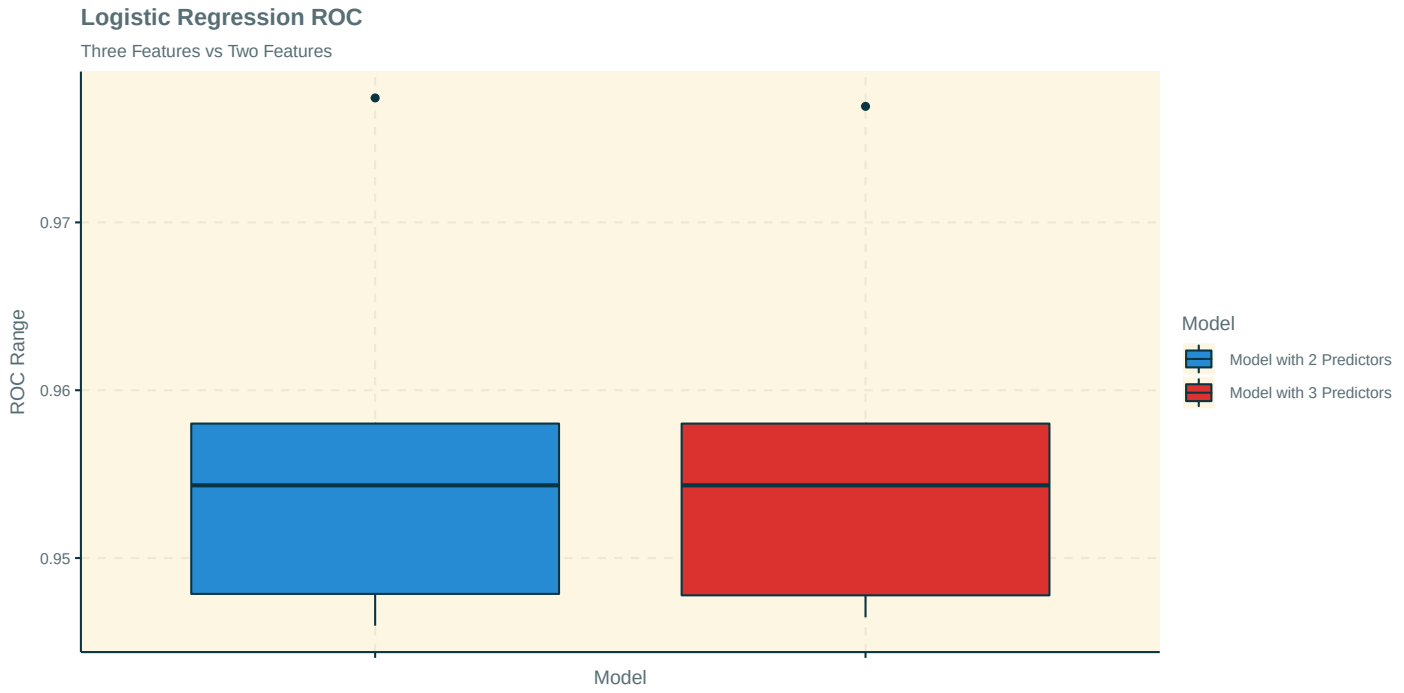
```
## glm variable importance
##
##                Overall
## Var_Wave_Trans   100.0
## Skw_Wave_Trans    58.8
## Entro_Image        0.0
```

From the above we can see that Entro_Image has Zero value for the model, but before taking a decision to exclude it. Let us train our Model using two Predictors then compare results.

```
# Train Logistic model with 2 Predictors

glm_model_2 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans, method = "glm", data=
 train_set, trControl = Control, family = "binomial")
```

Now we can compare ROC in the two models.

```
# Combined Accuracy in on dataset

Comp_models <- rbind(glm_model$resample %>% mutate(Model=" Model with 3 Predictors"),
  glm_model_2$resample %>% mutate(Model=" Model with 2 Predictors"))

# Plot Accuracy Ranges colored by Model

Comp_models %>% ggplot(aes(Model, ROC, fill = Model)) +
  geom_boxplot() +
  theme(axis.text.x = element_blank()) +
  theme(text = element_text(size=10))+
  ggtitle("Logistic Regression ROC")+
  labs(subtitle ="Three Features vs Two Features" ,
  y="ROC Range")
```



We can see that the ROC with 3 Predictors is higher than 2 Predictors. However, the Range of Accuracy for model of 3 Predictors is bigger than 2 Predictors.

The final decision will be based on Anova test to check whether the model with 3 Predictors gives improvement which is statistically significant or not.

```
# Perform Anova using Chisq

anova(glm_model$finalModel , glm_model_2$finalModel, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: .outcome ~ Var_Wave_Trans + Skw_Wave_Trans + Entro_Image
## Model 2: .outcome ~ Var_Wave_Trans + Skw_Wave_Trans
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
```

```
## 1      1093        569
## 2      1094        570 -1   -0.401     0.53
```

We can conclude that the improvement in the model with 3 Predictors is not statistically significant at pvalue 0.28. This suggests that with 3 Predictors does not provide an improved model.

Fifth, Best Threshold or Cut-off. By Default 0.5 is chosen as Threshold but let us examine if we can achieve better performance by other Threshold.

In the following Chunk of code, I will build a general function that could be used with other models as well.

```
# In This Function
### Use Saved Predicted Propability
### Measure Performance of each Test Fold with selected threshold

TryThreshold <- function(threshold, model) {

# Get each Test Fold Name
       folds_names <- levels(as.factor(model$pred$Resample))
# Repeat Performance Meause on each Fold
       lapply(folds_names, function(fold) {
# Get Predicted Propability
             pred_prop <- model$pred %>% filter(Resample==fold) %>% pull(Fraudulent)
             #print(length(pred_prop))
# Get Predicted Class
             pred_class <- ifelse(pred_prop >= threshold , "Fraudulent", "Genuine")
             #print(length(pred_class))
# Get Actual Class
             actual_class <- model$pred %>% filter(Resample==fold) %>% pull(obs)
# Build  confusionMatrix
             cm <- confusionMatrix(factor(pred_class),factor(actual_class))
# Convert  confusionMatrix  to dataframe
             cm_metrics_df <- cm$table %>% as.vector() %>% matrix(ncol = 4) %>% as.data.frame()
# Rename  as the following:

#       Prediction      Actual      Name
############################################################
#       Fraudulent     Fraudulent   FF
#       Fraudulent     Genuine      FG
#       Genuine        Fraudulent   GF
#       Genuine        Genuine      GG

       names(cm_metrics_df) <- c("FF", "GF", "FG", "GG")

 # Get other Performance  Metrics Names
             other_metrics_names <- c(cm$overall, cm$byClass)%>% as.data.frame() %>% row.names()

# Other Performance  Metrics Names:
          ##[1] "Accuracy"             "Kappa"                "AccuracyLower"
          ##[4] "AccuracyUpper"        "AccuracyNull"         "AccuracyPValue"
          ##[7] "McnemarPValue"        "Sensitivity"          "Specificity"
          ##[10] "Pos Pred Value"       "Neg Pred Value"       "Precision"
          ##[13] "Recall"               "F1"                   "Prevalence"
          ##[16] "Detection Rate"       "Detection Prevalence" "Balanced Accuracy"
 # Get other Performance  Metrics Values
```

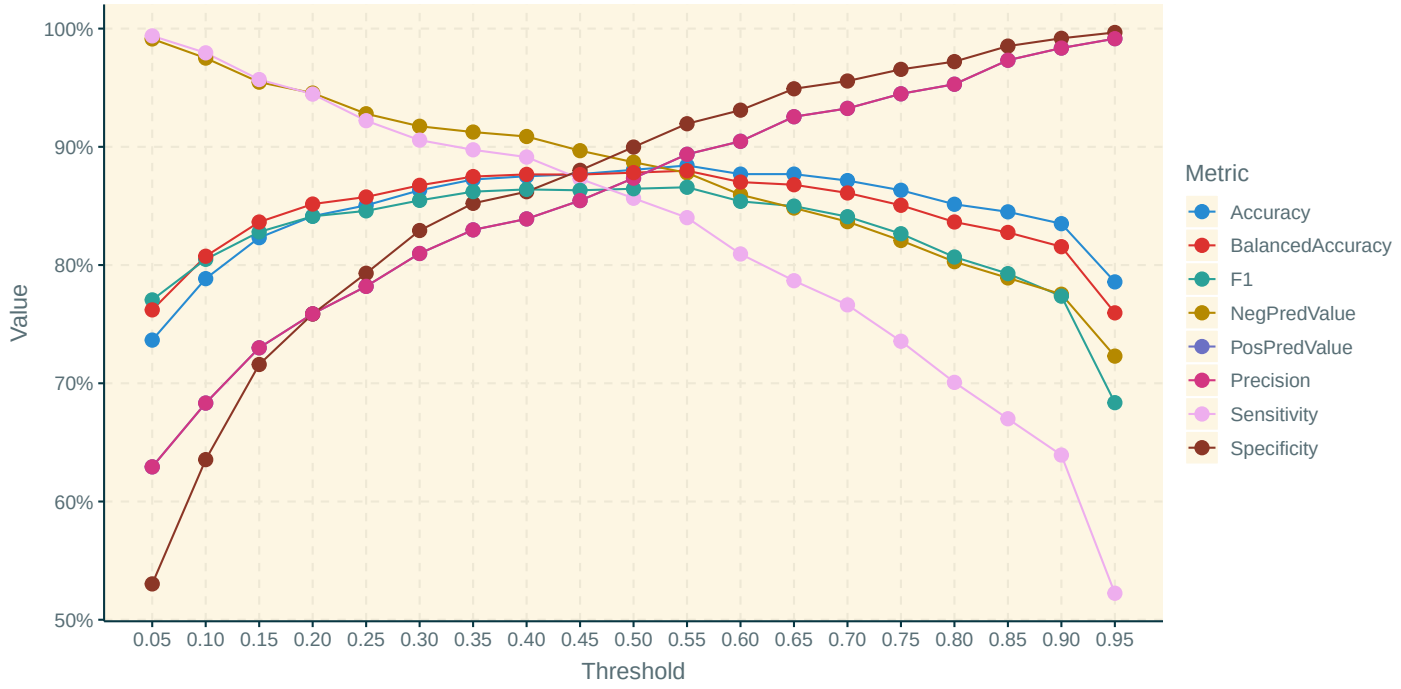
```
              other_metrics_df <- c(cm$overall, cm$byClass) %>% as.vector() %>%
                matrix(ncol = 18) %>% as.data.frame()

 # Rename other Performance  Metrics Values
              names(other_metrics_df) <- other_metrics_names

 # Remove Space from Other Performance  Metrics Names
              names(other_metrics_df) <- names(other_metrics_df) %>% str_replace_all(" ", "")
 # Merge all Performance  Metrics
              all_metrics_df <- data.frame(bind_cols(other_metrics_df, cm_metrics_df))

              return(all_metrics_df)
        })
}
```

Now, it is time to try Threshold from 0.1 to 0.95

```
# Try Threshold from 0.1 to 0.95

metrics_df <- lapply(seq(0.05, 0.95, by = 0.05),TryThreshold, glm_model_2)

# Convert result to dataframe

metrics_df <- do.call("bind_rows", metrics_df)

# Add Threshold to the Performance Measure dataframe
# Each Threshold used for 5 times validation, so this why we replicate each Threshold for Threshold

metrics_df <- metrics_df %>% mutate(Threshold = lapply(seq(0.05, 0.95, by = 0.05),
  function(x) {rep(x, 5)})%>% unlist())
```

It will be better of Visualize the Performance Measures as the following:

```
#For each Threshold, Get Average of each Measure across 5 Validation datasets

metrics_df <- metrics_df %>%
  select(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy,Threshold) %>%
  group_by(Threshold) %>%
  summarise_at(vars(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy), mean)

#Plot measures against Threshold

  metrics_df %>%gather(Metric, Value, -Threshold) %>%
    ggplot(aes(Threshold, Value, color = Metric)) +
    geom_line() +
    geom_point(size = 3) +
    scale_y_continuous(labels = scales::percent_format(accuracy = 1))+
    scale_x_continuous(breaks = seq(0.05, 0.95, by = 0.05))
```



As we have two classed, let us choose the Threshold that maximize F1 score

```
#Get Threshold that maximize F1 score

glm_Threshold_df <- metrics_df %>% select(F1,Threshold)
glm_Threshold <- glm_Threshold_df[ which.max(glm_Threshold_df$F1) ,2]
glm_Threshold
```

```
## # A tibble: 1 x 1
##   Threshold
##       <dbl>
## 1      0.55
```

Let us also plot ROC curve which is sensitivity (TPR) versus 1-specificity or the false positive rate (FPR).

In addition, to plot precision-recall/sensitivity to make sure that we are also maintain precision.

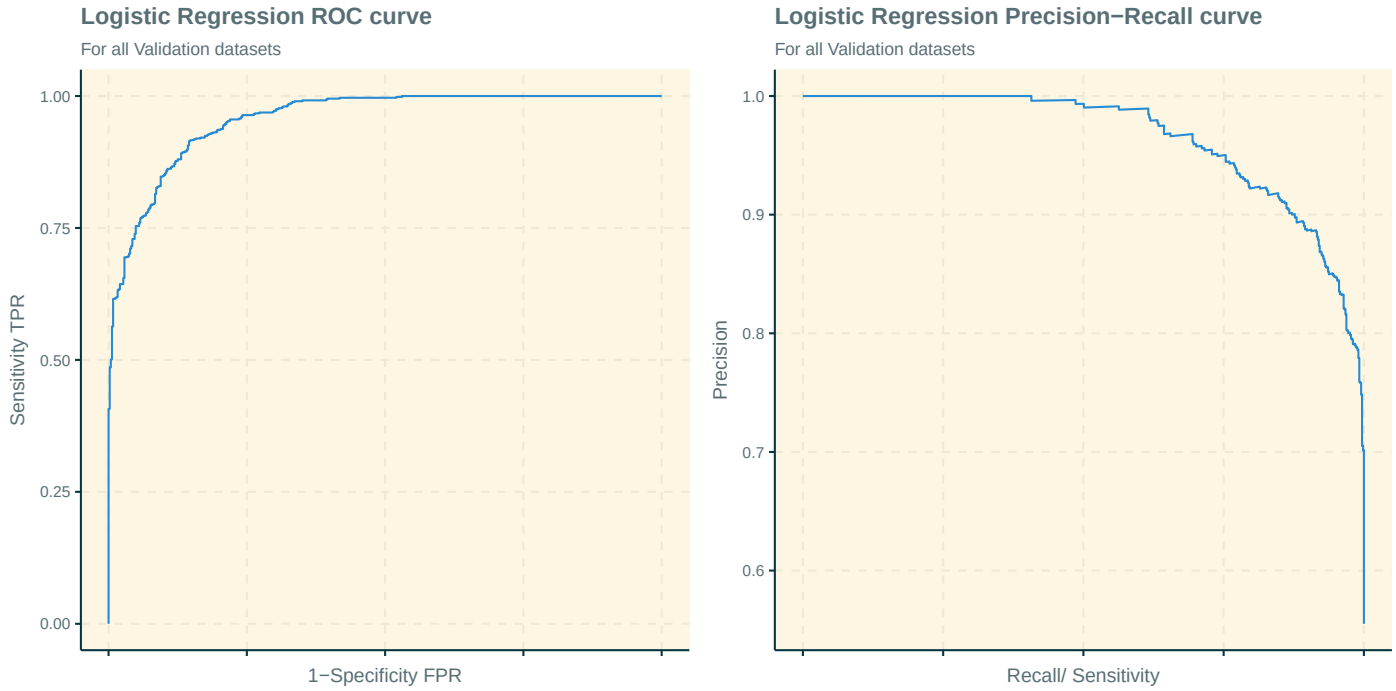
Before plotting curves, let us calculate points data for each curve.

```
# Calculate Points for each curve and convert S3 object to dataframe for easy plotting

perf_curves <- evalmod(scores = glm_model_2$pred$Genuine, labels = glm_model_2$pred$obs)

perf_curves_df <- fortify(perf_curves)
```

Now, let us check ROC curve.

```
# Plot ROC curve

plot1 <- perf_curves_df %>% filter (curvetype=="ROC")%>% ggplot(aes(x, y)) +
```

```
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Logistic Regression ROC curve")+
labs(subtitle ="For all Validation datasets" ,
y="Sensitivity TPR", x="1-Specificity FPR")

plot2 <- perf_curves_df %>% filter (curvetype=="PRC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Logistic Regression Precision-Recall curve")+
labs(subtitle ="For all Validation datasets" ,
y="Precision", x="Recall/ Sensitivity")

grid.arrange(plot1, plot2, ncol=2)
```



From previous graphs to is obvious that we have high value for AUC in both curves.

ROC can be calculated using the following:

```
aucs <- auc(perf_curves)
knitr::kable(aucs)
```

modnames	dsids	curvetypes	aucs
m1	1	ROC	0.95644
m1	1	PRC	0.96482

2.3.3 Logistic Regression Prediction

Now, it is time for prediction on unseen data of the test dataset.

```
# Predict on Test set

test_pred_prob <- predict(glm_model_2,test_set[,1:2],type="prob")

# Get Fraudulent Probability

test_pred_prob <- test_pred_prob %>% pull(Fraudulent)

# Get Predicted Class

test_pred_class <- ifelse(test_pred_prob >= as.numeric(glm_Threshold) , "Fraudulent","Genuine")

# Build confusionMatrix

cf_mtx <-  confusionMatrix(factor(test_pred_class),factor(test_set$Class))
```

Now, let us build a table to save all measures of each algorithm.

```
performance_tb <- NULL

# Build a Table to save all Performance Measures

performance_tb <- tibble(method = "Logistic Regression",
F1 = cf_mtx$byClass["F1"], Sensitivity=cf_mtx$byClass["Sensitivity"],
Specificity=cf_mtx$byClass["Specificity"] ,Precision=cf_mtx$byClass["Precision"] ,
Balanced_Accuracy= cf_mtx$byClass["Balanced Accuracy"] )

kable(performance_tb,"latex", booktabs = T)%>%
kable_styling(latex_options = "striped")
```

method	F1	Sensitivity	Specificity	Precision	Balanced_Accuracy
Logistic Regression	0.8595	0.85246	0.89542	0.86667	0.87394

2.4 k-nearest neighbors

Now, we can start with our Second algorithm k-nearest neighbors.

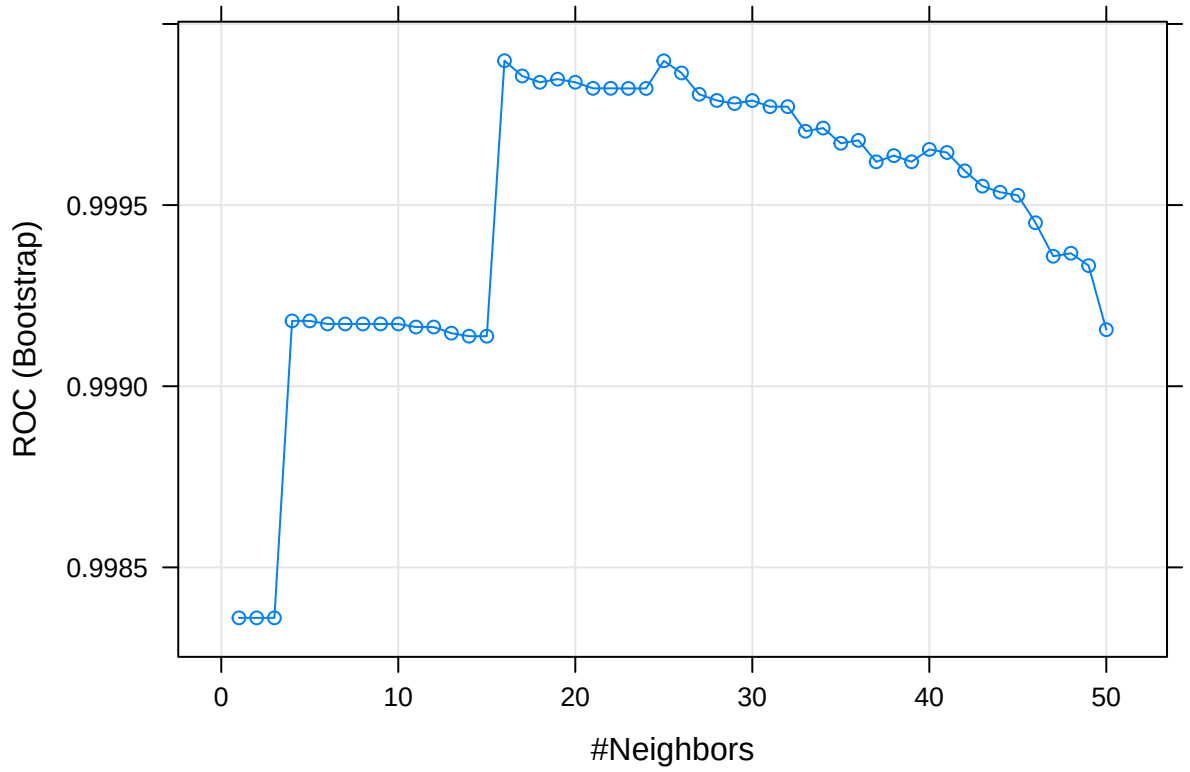
2.4.1 k-nearest neighbors Training

We will use the same control while training Logistic Regression algorithm. However, in KNN we have a parameter to tune which is the number of neighbors to include while training.

Training will start by 4 predictors

```
# Train KNN

KNN_model <- train(Class ~ ., method = "knn", data = train_set,trControl = Control,
tuneGrid = data.frame(k = seq(1, 50, 1)),preProcess = c("center","scale"))

# Plot ROC against number of neighbors
```

```
plot(KNN_model)
```



we can see that we can almost get ROC equals to one by using only 18 neighbors.

2.4.2 k-nearest neighbors Tuning

In that section, we will check the following:

- Number of Predictors/ Variables
- Best Threshold or Cut-off

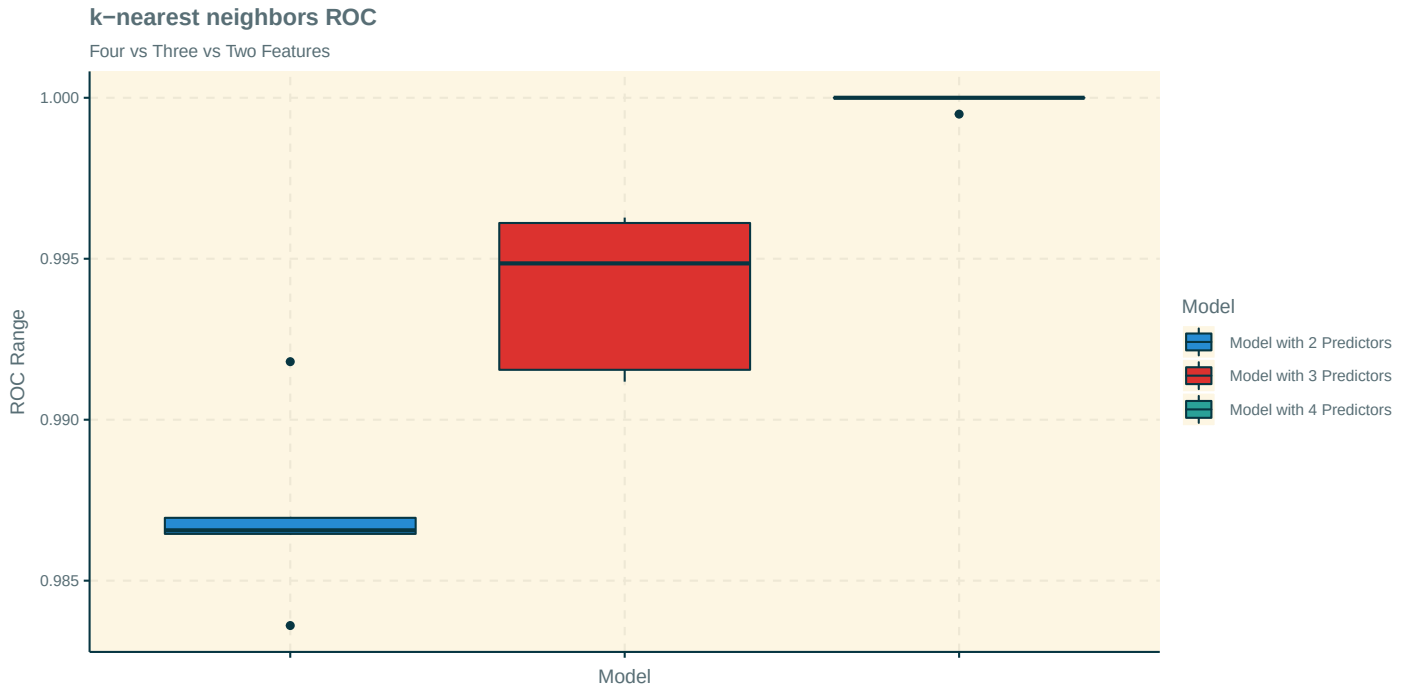
First, let us check train our model by 3 and 2 Predictors as we have done in logistic Regression.

- let us try train our model by excluding Cur_Wave_Trans Feature.
- let us try train our model by excluding Cur_Wave_Trans and Entro_Image Features.

```
# Check each Predictor importance

KNN_model_2 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans + Entro_Image, method = "knn",
  data = train_set,trControl = Control,
  tuneGrid = data.frame(k = seq(1, 50, 1)),preProcess = c("center","scale"))

KNN_model_3 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans , method = "knn",
  data = train_set,trControl = Control,
  tuneGrid = data.frame(k = seq(1, 50, 1)),preProcess = c("center","scale"))
```

Now we can compare ROC in the three models.

```
# Combined Accuracy in on dataset

Comp_models <- rbind(KNN_model$resample %>% mutate(Model=" Model with 4 Predictors"),
                          KNN_model_2$resample %>% mutate(Model=" Model with 3 Predictors"),
                         KNN_model_3$resample %>% mutate(Model=" Model with 2 Predictors"))

# Plot Accuracy Ranges colored by Model

Comp_models %>% ggplot(aes(Model, ROC, fill = Model)) +
geom_boxplot() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("k-nearest neighbors ROC")+
labs(subtitle ="Four vs Three vs Two Features" ,
y="ROC Range")
```



We can see that the ROC with 4 Predictors is higher than 2 and 3 Predictors. In addition, the Range of Accuracy for model of 4 Predictors is limited range almost 1.

Hence, We can conclude that the improvement in the model with 4 Predictors is considerable and we will select the model with 4 Predictors going forward in KNN.

Second, Best Threshold or Cut-off. By Default 0.5 is chosen as Threshold but let us examine if we can achieve better performance by other Threshold.

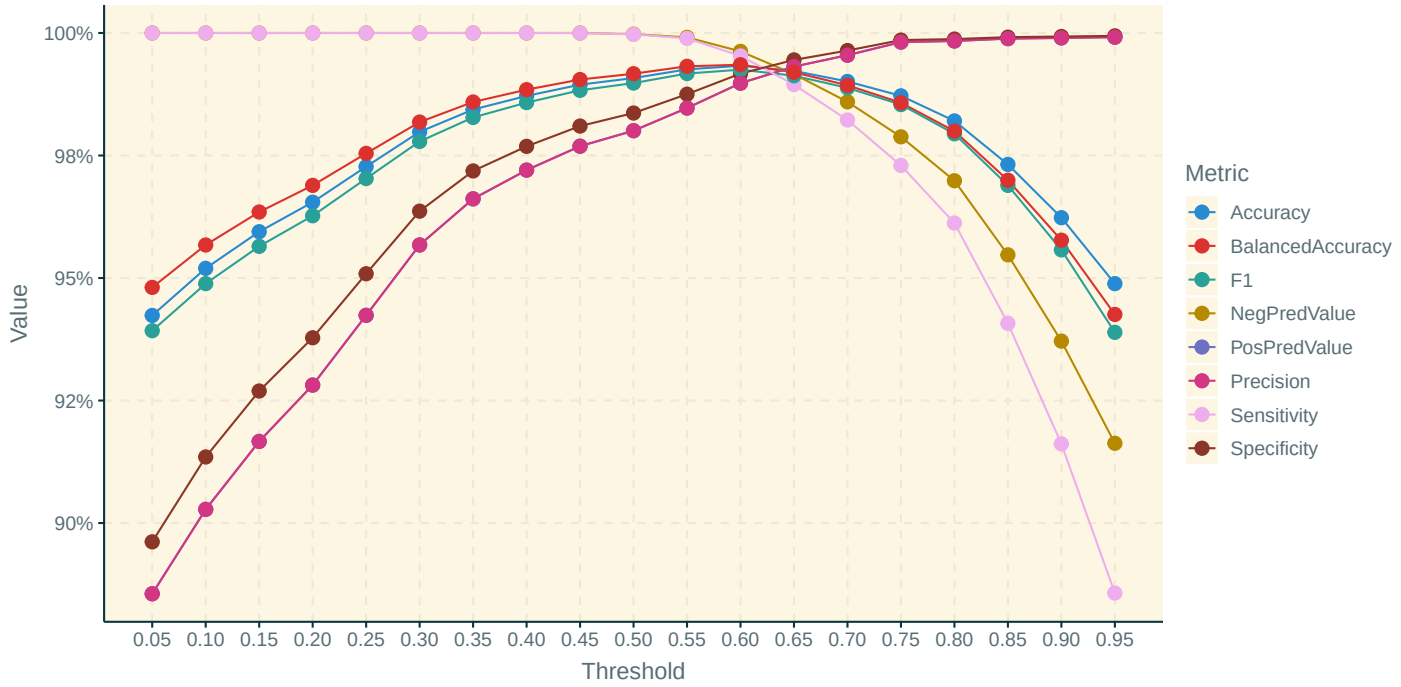
Now, it is time to try Threshold from 0.1 to 0.95

```
# Try Threshold from 0.1 to 0.95

metrics_df <- lapply(seq(0.05, 0.95, by = 0.05),TryThreshold, KNN_model)
```

```
# Convert result to dataframe

metrics_df <- do.call("bind_rows", metrics_df)

# Add Threshold to the Performance Measure dataframe
# Each Threshold used for 5 times validation, so this why we replicate each Threshold for Threshold

metrics_df <- metrics_df %>% mutate(Threshold = lapply(seq(0.05, 0.95, by = 0.05),
  function(x) {rep(x, 5)})%>% unlist())
```

It will be better of Visualize the Performance Measures as the following:

```
#For each Threshold, Get Average of each Measure across 5 Validation datasets

metrics_df <- metrics_df %>%
  select(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy,Threshold) %>%
  group_by(Threshold) %>%
  summarise_at(vars(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy), mean)

#Plot measures against Threshold

  metrics_df %>%gather(Metric, Value, -Threshold) %>%
  ggplot(aes(Threshold, Value, color = Metric)) +
  geom_line() +
  geom_point(size = 3) +
 scale_y_continuous(labels = scales::percent_format(accuracy = 1))+
  scale_x_continuous(breaks = seq(0.05, 0.95, by = 0.05))
```



As we have two classed, let us choose the Threshold that maximize F1 score.

```
#Get Threshold that maximize F1 score

KNN_Threshold_df <- metrics_df %>% select(F1,Threshold)
KNN_Threshold <- KNN_Threshold_df[ which.max(KNN_Threshold_df$F1) ,2]
KNN_Threshold
```

```
## # A tibble: 1 x 1
##   Threshold
##       <dbl>
## 1       0.6
```

Let us also plot ROC curve which is sensitivity (TPR) versus 1-specificity or the false positive rate (FPR).

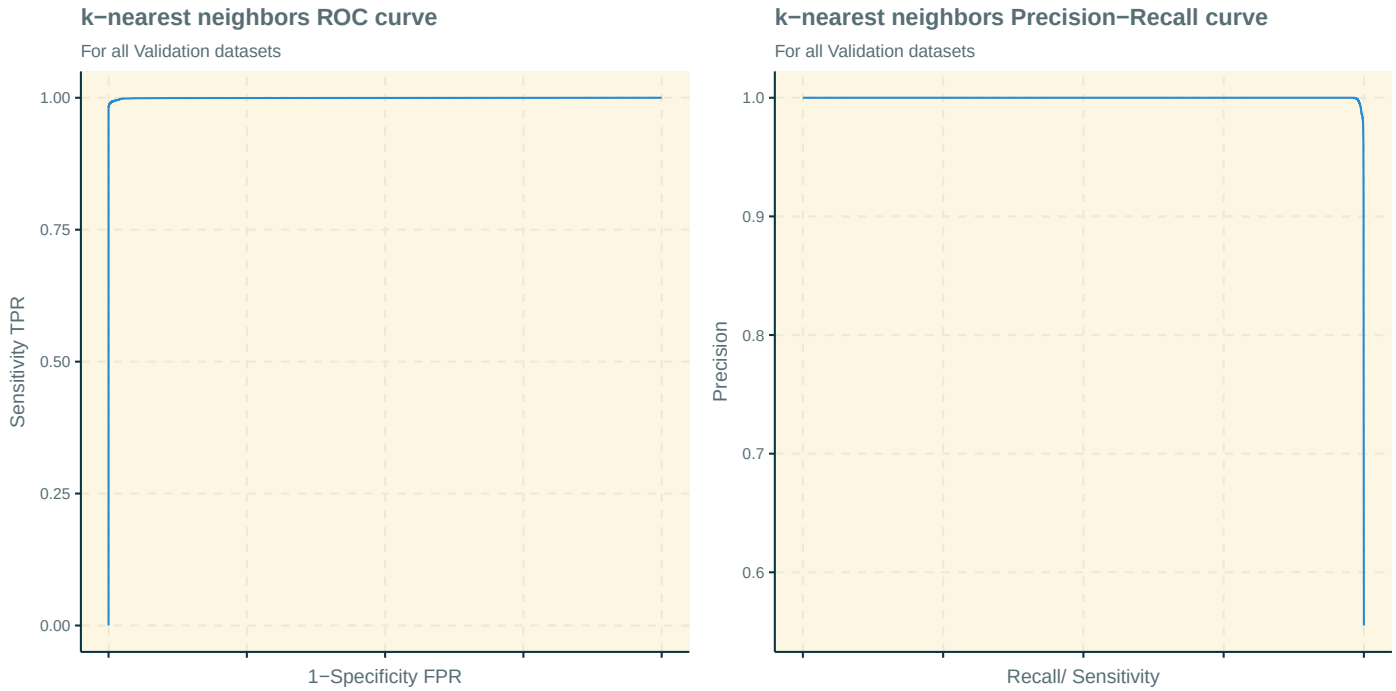
In addition, to plot precision-recall/sensitivity to make sure that we are also maintain precision.

Before plotting curves, let us calculate points data for each curve.

```
# Calculate Points for each curve and convert S3 object to dataframe for easy plotting

perf_curves <- evalmod(scores = KNN_model$pred$Genuine, labels = KNN_model$pred$obs)

perf_curves_df <- fortify(perf_curves)
```

Now, let us check ROC curve.

```
# Plot ROC curve

plot1 <- perf_curves_df %>% filter (curvetype=="ROC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("k-nearest neighbors ROC curve")+
labs(subtitle ="For all Validation datasets" ,
y="Sensitivity TPR", x="1-Specificity FPR")

plot2 <- perf_curves_df %>% filter (curvetype=="PRC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("k-nearest neighbors Precision-Recall curve")+
labs(subtitle ="For all Validation datasets" ,
y="Precision", x="Recall/ Sensitivity")

grid.arrange(plot1, plot2, ncol=2)
```



From previous graphs to is obvious that we have high value for AUC in both curves almost 1.

ROC can be calculated using the following:

```
aucs <- auc(perf_curves)
knitr::kable(aucs)
```

modnames	dsids	curvetypes	aucs
m1	1	ROC	0.9995
m1	1	PRC	0.9997

2.4.3 k-nearest neighbors Prediction

Now, it is time for prediction on unseen data of the test dataset.

```
# Predict on Test set

test_pred_prob <- predict(KNN_model,test_set[,1:4],type="prob")

# Get Fraudulent Probability

test_pred_prob <- test_pred_prob %>% pull(Fraudulent)

# Get Predicted Class

test_pred_class <- ifelse(test_pred_prob >= as.numeric(KNN_Threshold) , "Fraudulent","Genuine")

# Build confusionMatrix

cf_mtx <-  confusionMatrix(factor(test_pred_class),factor(test_set$Class))
```

Now, let us add to measures of each algorithm.

```
# Build a Table to save all Performance Measures

performance_tb <- rbind(performance_tb, tibble(method = "k-nearest neighbors",
F1 = cf_mtx$byClass["F1"], Sensitivity=cf_mtx$byClass["Sensitivity"],
Specificity=cf_mtx$byClass["Specificity"] ,Precision=cf_mtx$byClass["Precision"] ,
Balanced_Accuracy= cf_mtx$byClass["Balanced Accuracy"] ))

kable(performance_tb,"latex", booktabs = T)%>%
kable_styling(latex_options = "striped")
```

method	F1	Sensitivity	Specificity	Precision	Balanced_Accuracy
Logistic Regression	0.8595	0.85246	0.89542	0.86667	0.87394
k-nearest neighbors	1.0000	1.00000	1.00000	1.00000	1.00000

2.5 Support Vector Machine

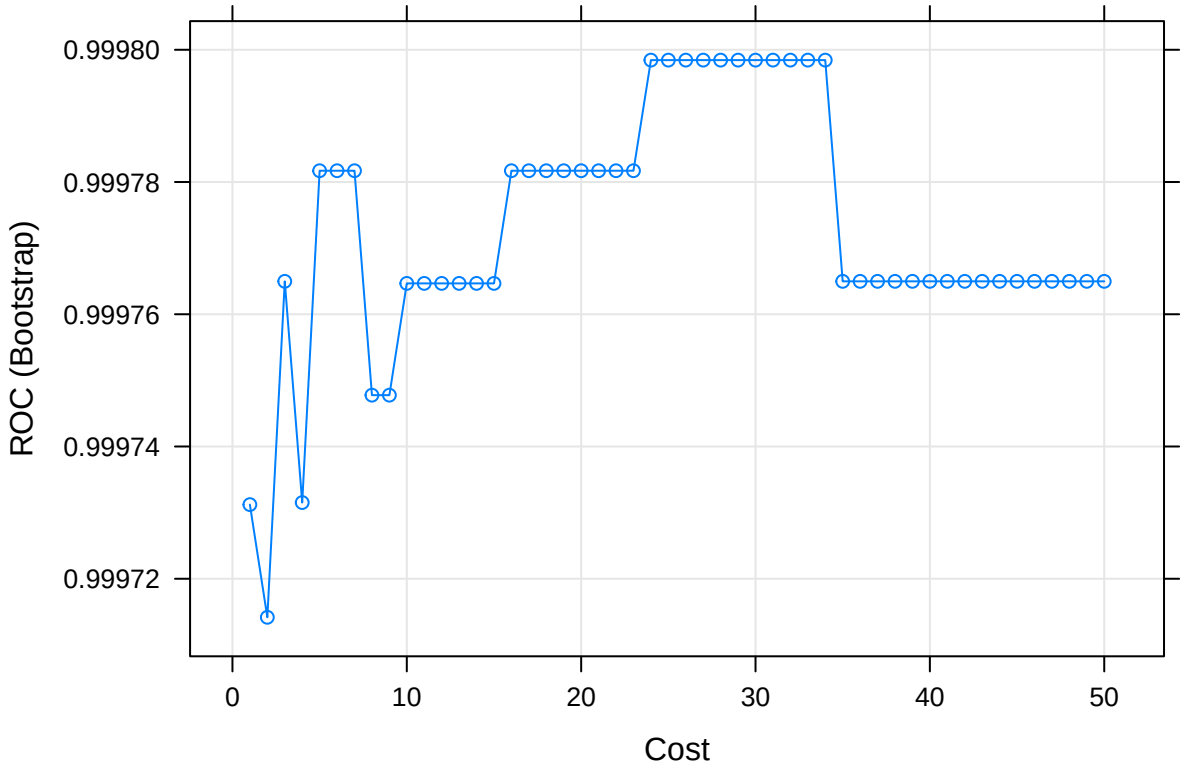
Now, we can start with our Third algorithm Support Vector Machine.

2.5.1 Support Vector Machine Training

We will use the same control while training Logistic Regression algorithm. However, in SVM we have a parameter to tune which is known as Cost, that determines the possible miss classifications. It essentially imposes a penalty to the model for making an error, the higher the value of C, the less likely it is that the SVM algorithm will miss classify an observation.

Training will start by 4 predictors

```
# Train SVM

SVM_model <- train(Class ~ ., method = "svmLinear", data = train_set,trControl = Control,
tuneGrid = data.frame(C = seq(1, 50, 1)),preProcess = c("center","scale"))

# Plot ROC against C

plot(SVM_model)
```



we can see that we can almost get ROC equals to one by using only Cost equals to 27.

2.5.2 Support Vector Machine Tuning

In that section, we will check the following:

- Number of Predictors/ Variables
- Best Threshold or Cut-off

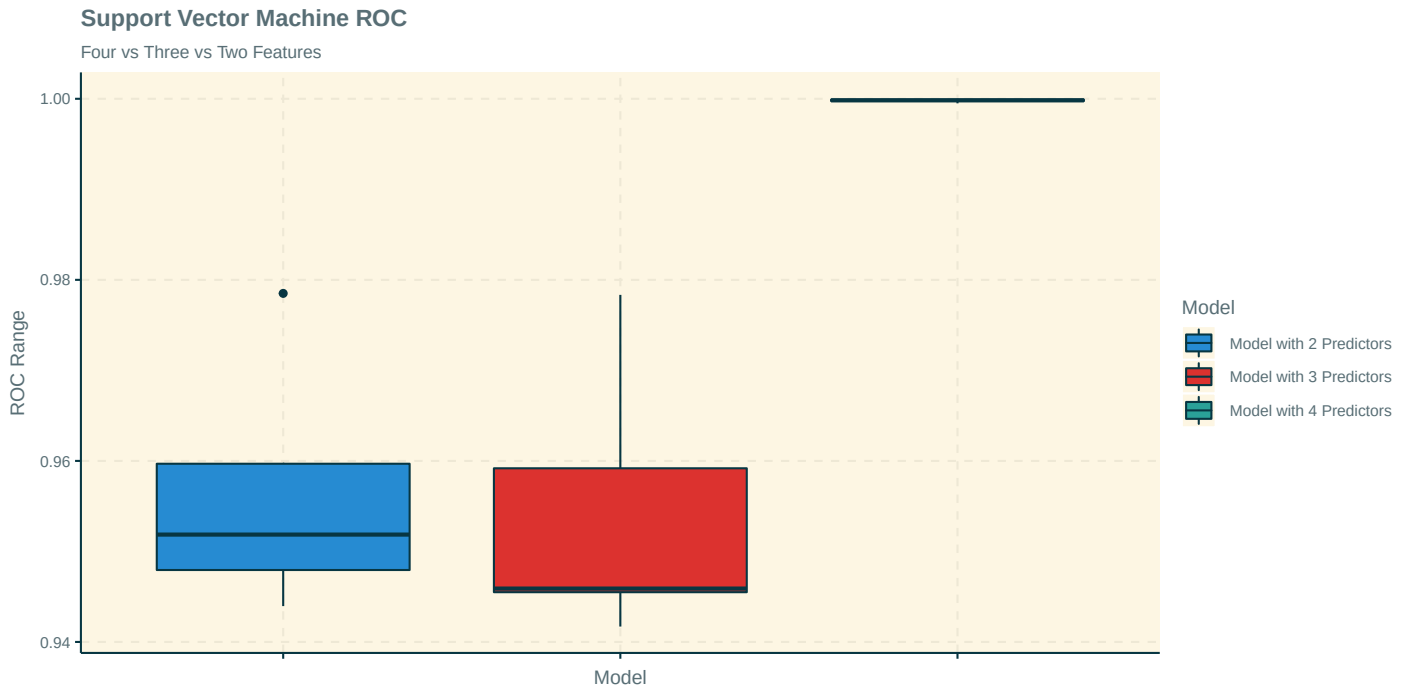
First, let us check train our model by 3 and 2 Predictors as we have done in logistic Regression.

- let us try train our model by excluding Cur_Wave_Trans Feature.
- let us try train our model by excluding Cur_Wave_Trans and Entro_Image Features.

```
# Check each Predictor importance

SVM_model_2 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans + Entro_Image,
  method ="svmLinear", data = train_set,trControl = Control,
  tuneGrid = data.frame(C = seq(1, 50, 1)),preProcess = c("center","scale"))

SVM_model_3 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans , method = "svmLinear",
 data = train_set,trControl = Control,
 tuneGrid = data.frame(C = seq(1, 50, 1)),preProcess = c("center","scale"))
```

Now we can compare ROC in the three models.

```
# Combined Accuracy in on dataset

Comp_models <- rbind(SVM_model$resample %>% mutate(Model=" Model with 4 Predictors"),
                          SVM_model_2$resample %>% mutate(Model=" Model with 3 Predictors"),
                         SVM_model_3$resample %>% mutate(Model=" Model with 2 Predictors"))

# Plot Accuracy Ranges colored by Model

Comp_models %>% ggplot(aes(Model, ROC, fill = Model)) +
geom_boxplot() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Support Vector Machine ROC")+
labs(subtitle ="Four vs Three vs Two Features" ,
y="ROC Range")
```



We can see that the ROC with 4 Predictors is higher than 2 and 3 Predictors. In addition, the Range of Accuracy for model of 4 Predictors is limited range almost 1. Hence, We can conclude that the improvement in the model with 4 Predictors is considerable and we will select the model with 4 Predictors going forward in SVM

Second, Best Threshold or Cut-off. By Default 0.5 is chosen as Threshold but let us examine if we can achieve better performance by other Threshold.

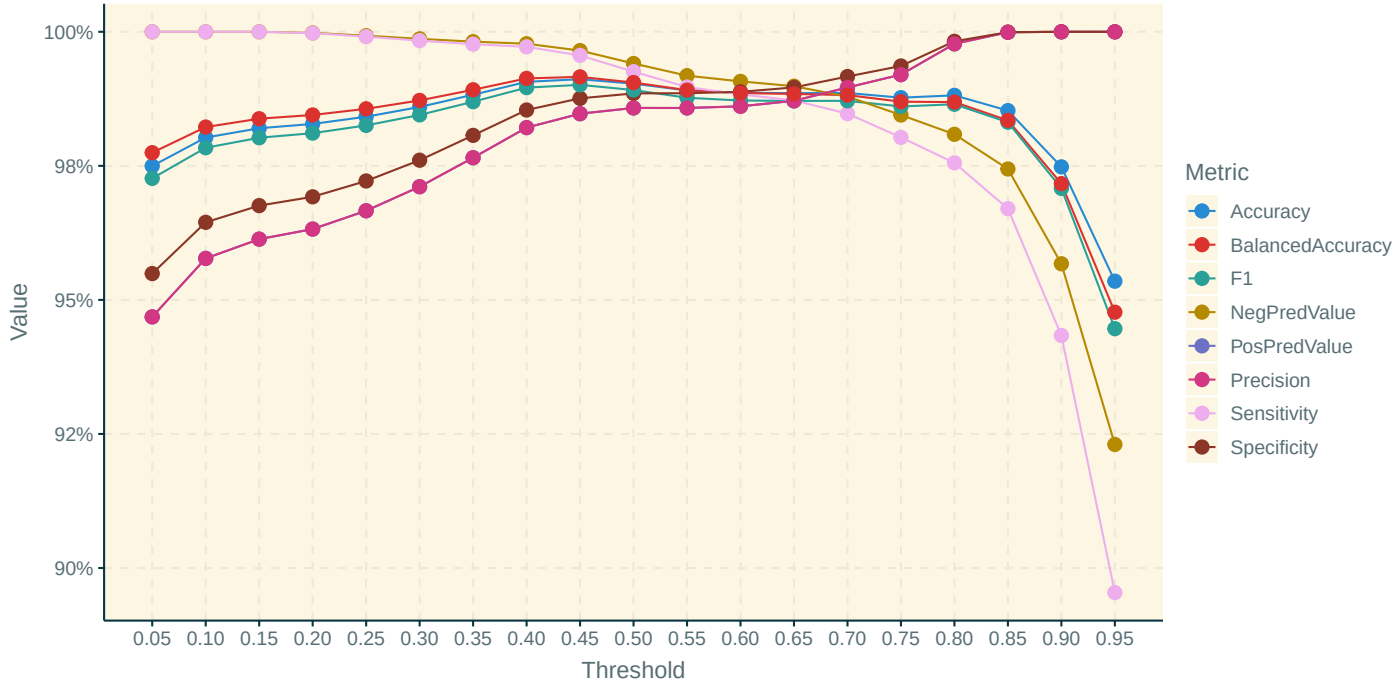
Now, it is time to try Threshold from 0.1 to 0.95

```
# Try Threshold from 0.1 to 0.95

metrics_df <- lapply(seq(0.05, 0.95, by = 0.05),TryThreshold, SVM_model)

# Convert result to dataframe
```

```
metrics_df <- do.call("bind_rows", metrics_df)

# Add Threshold to the Performance Measure dataframe
# Each Threshold used for 5 times validation, so this why we replicate each Threshold for Threshold

metrics_df <- metrics_df %>% mutate(Threshold = lapply(seq(0.05, 0.95, by = 0.05),
  function(x) {rep(x, 5)})%>% unlist())
```

It will be better of Visualize the Performance Measures as the following:

```
#For each Threshold, Get Average of each Measure across 5 Validation datasets

metrics_df <- metrics_df %>%
  select(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy,Threshold) %>%
  group_by(Threshold) %>%
  summarise_at(vars(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy), mean)

#Plot measures against Threshold

  metrics_df %>%gather(Metric, Value, -Threshold) %>%
  ggplot(aes(Threshold, Value, color = Metric)) +
  geom_line() +
  geom_point(size = 3) +
 scale_y_continuous(labels = scales::percent_format(accuracy = 1))+
  scale_x_continuous(breaks = seq(0.05, 0.95, by = 0.05))
```



As we have two classed, let us choose the Threshold that maximize F1 score.

```
#Get Threshold that maximize F1 score

SVM_Threshold_df <- metrics_df %>% select(F1,Threshold)
SVM_Threshold <- SVM_Threshold_df[ which.max(SVM_Threshold_df$F1) ,2]
SVM_Threshold
```

```
## # A tibble: 1 x 1
##   Threshold
##       <dbl>
## 1      0.45
```

Let us also plot ROC curve which is sensitivity (TPR) versus 1-specificity or the false positive rate (FPR).

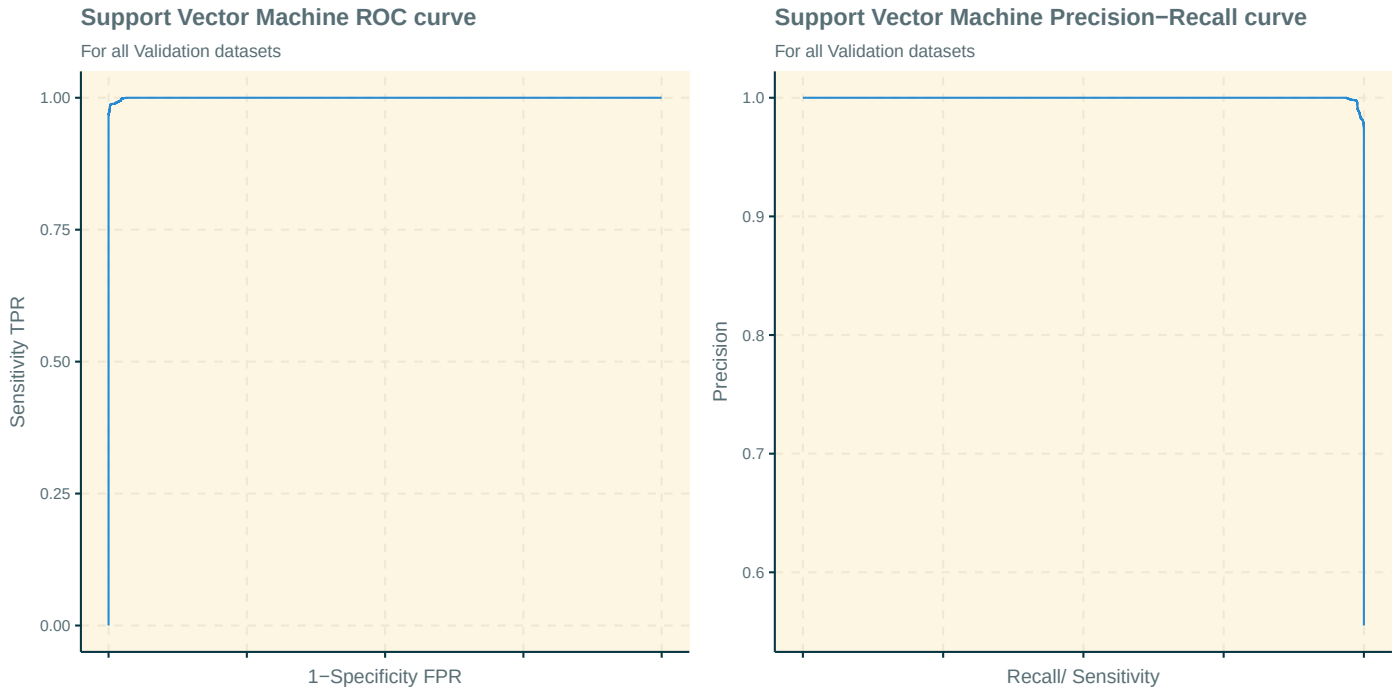
In addition, to plot precision-recall/sensitivity to make sure that we are also maintain precision.

Before plotting curves, let us calculate points data for each curve.

```
# Calculate Points for each curve and convert S3 object to dataframe for easy plotting

perf_curves <- evalmod(scores = SVM_model$pred$Genuine, labels = SVM_model$pred$obs)

perf_curves_df <- fortify(perf_curves)
```

Now, let us check ROC curve.

```
# Plot ROC curve

plot1 <- perf_curves_df %>% filter (curvetype=="ROC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Support Vector Machine ROC curve")+
labs(subtitle ="For all Validation datasets" ,
y="Sensitivity TPR", x="1-Specificity FPR")

plot2 <- perf_curves_df %>% filter (curvetype=="PRC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Support Vector Machine Precision-Recall curve")+
labs(subtitle ="For all Validation datasets" ,
y="Precision", x="Recall/ Sensitivity")

grid.arrange(plot1, plot2, ncol=2)
```



From previous graphs to is obvious that we have high value for AUC in both curves almost 1.

ROC can be calculated using the following:

```
aucs <- auc(perf_curves)
knitr::kable(aucs)
```

modnames	dsids	curvetypes	aucs
m1	1	ROC	0.99973
m1	1	PRC	0.99979

2.5.3 Support Vector Machine Prediction

Now, it is time for prediction on unseen data of the test dataset.

```
# Predict on Test set

test_pred_prob <- predict(SVM_model,test_set[,1:4],type="prob")

# Get Fraudulent Probability

test_pred_prob <- test_pred_prob %>% pull(Fraudulent)

# Get Predicted Class

test_pred_class <- ifelse(test_pred_prob >= as.numeric(SVM_Threshold) , "Fraudulent","Genuine")

# Build confusionMatrix

cf_mtx <-  confusionMatrix(factor(test_pred_class),factor(test_set$Class))
```

Now, let us add to measures table of each algorithm.

```
# Build a Table to save all Performance Measures

performance_tb <- rbind(performance_tb, tibble(method = "Support Vector Machine",
F1 = cf_mtx$byClass["F1"], Sensitivity=cf_mtx$byClass["Sensitivity"],
Specificity=cf_mtx$byClass["Specificity"] ,Precision=cf_mtx$byClass["Precision"] ,
Balanced_Accuracy= cf_mtx$byClass["Balanced Accuracy"] ))

kable(performance_tb,"latex", booktabs = T)%>%
kable_styling(latex_options = "striped")
```

method	F1	Sensitivity	Specificity	Precision	Balanced_Accuracy
Logistic Regression	0.85950	0.85246	0.89542	0.86667	0.87394
k-nearest neighbors	1.00000	1.00000	1.00000	1.00000	1.00000
Support Vector Machine	0.98785	1.00000	0.98039	0.97600	0.99020

2.6 Random forests

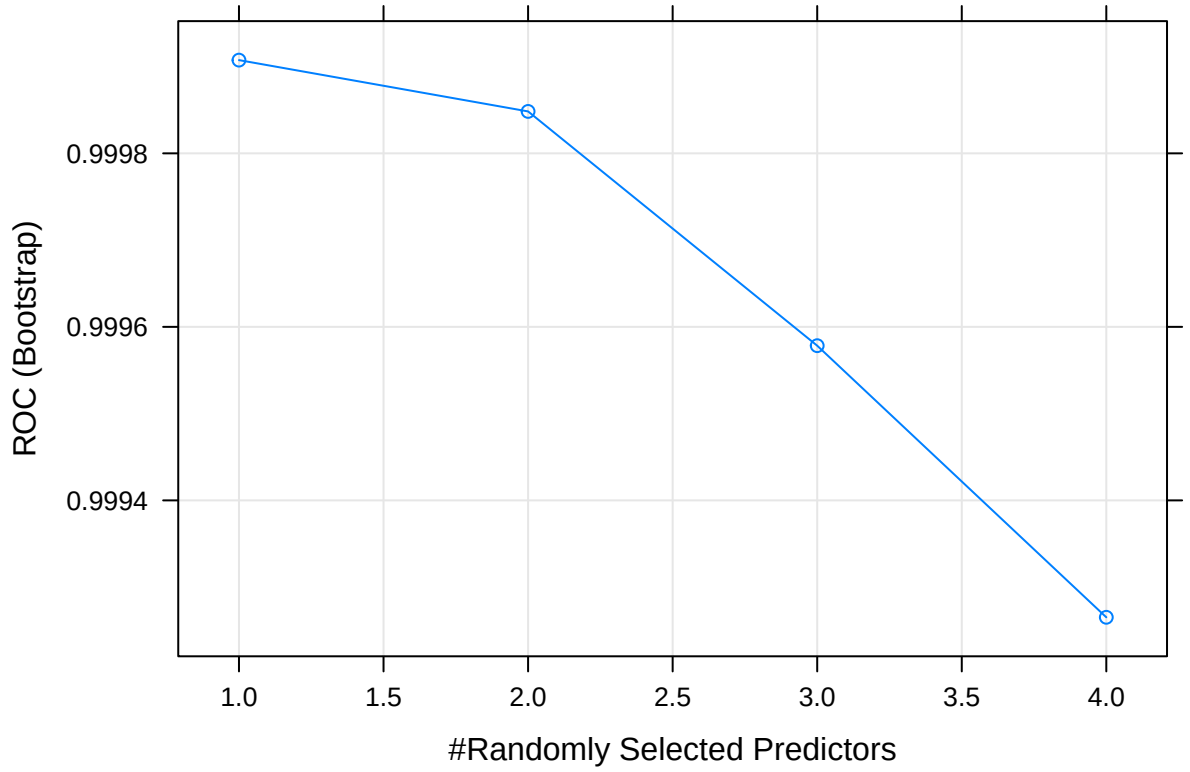
Now, we can start with our Fourth algorithm Random forests.

2.6.1 Random forests Training

We will use the same control while training Logistic Regression algorithm. However, in RF we have a parameter to tune which is known as mtry, that determines the Number of variables randomly sampled as candidates at each split. Actually we have maximum four predictors.

Training will start by 4 predictors

```
# Train RF

RF_model <- train(Class ~ ., method = "rf", data = train_set,trControl = Control,
tuneGrid = data.frame(mtry = seq(1, 4, 1)))

# Plot ROC against mtry

plot(RF_model)
```



we can see that we can almost get ROC equals to one by using mtry equals to 27.

2.6.2 Random forests Tuning

In that section, we will check the following:

- Number of Predictors/ Variables
- Best Threshold or Cut-off

First, let us check variable importance for 4 Predictors model.

```
# Check each Predictor importance

caret::varImp(RF_model)
```

```
## rf variable importance
##
##                Overall
## Var_Wave_Trans   100.0
## Skw_Wave_Trans    43.4
## Cur_Wave_Trans    20.5
## Entro_Image        0.0
```

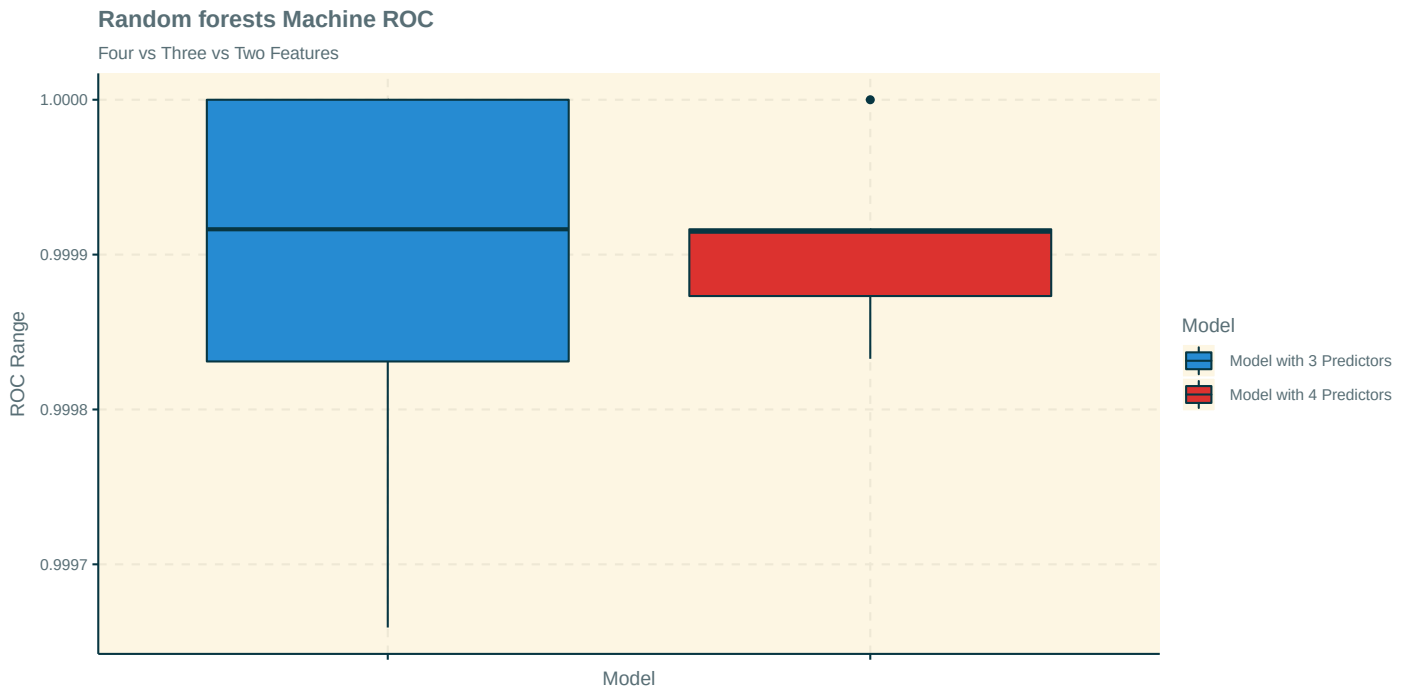
First, let us check train our model by 3 Predictors as we have done in logistic Regression.

- let us try train our model by excluding Entro_Image Feature.

```
# Train with 3 Predictors

RF_model_2 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans + Cur_Wave_Trans,
  method = "rf", data = train_set,trControl = Control,
  tuneGrid = data.frame(mtry = seq(1, 3, 1)))
```

Now we can compare ROC in the two models.

```
# Combined Accuracy in on dataset

Comp_models <- rbind(RF_model$resample %>% mutate(Model=" Model with 4 Predictors"),
                          RF_model_2$resample %>% mutate(Model=" Model with 3 Predictors"))

# Plot Accuracy Ranges colored by Model

Comp_models %>% ggplot(aes(Model, ROC, fill = Model)) +
geom_boxplot() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Random forests Machine ROC")+
labs(subtitle ="Four vs Three vs Two Features" ,
y="ROC Range")
```



We can see that the ROC with 4 Predictors is same as 3 Predictors. However, the Range of Accuracy for model of 3 Predictors is slightly bigger than 4 Predictors.

Hence, We can conclude that the improvement in the model with 4 Predictors is not considerable and we will select the model with 3 Predictors going forward in RF.

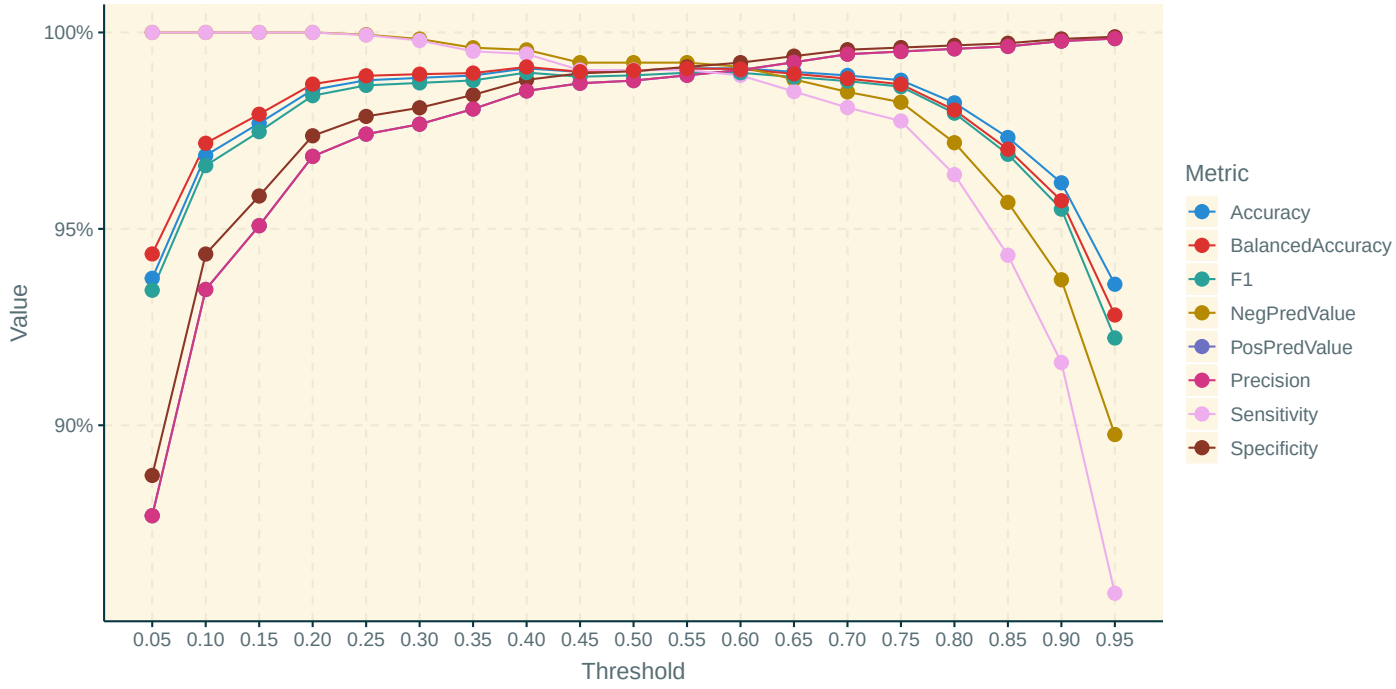
Second, Best Threshold or Cut-off. By Default 0.5 is chosen as Threshold but let us examine if we can achieve better performance by other Threshold.

Now, it is time to try Threshold from 0.1 to 0.95

```
# Try Threshold from 0.1 to 0.95

metrics_df <- lapply(seq(0.05, 0.95, by = 0.05),TryThreshold, RF_model_2)

# Convert result to dataframe

metrics_df <- do.call("bind_rows", metrics_df)

# Add Threshold to the Performance Measure dataframe
# Each Threshold used for 5 times validation, so this why we replicate each Threshold for Threshold

metrics_df <- metrics_df %>% mutate(Threshold = lapply(seq(0.05, 0.95, by = 0.05),
  function(x) {rep(x, 5)})%>% unlist())
```

It will be better of Visualize the Performance Measures as the following:

```
#For each Threshold, Get Average of each Measure across 5 Validation datasets

metrics_df <- metrics_df %>%
  select(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy,Threshold) %>%
  group_by(Threshold) %>%
  summarise_at(vars(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy), mean)

#Plot measures against Threshold

  metrics_df %>%gather(Metric, Value, -Threshold) %>%
  ggplot(aes(Threshold, Value, color = Metric)) +
  geom_line() +
  geom_point(size = 3) +
 scale_y_continuous(labels = scales::percent_format(accuracy = 1))+
  scale_x_continuous(breaks = seq(0.05, 0.95, by = 0.05))
```



As we have two classed, let us choose the Threshold that maximize F1 score.

```
#Get Threshold that maximize F1 score

RF_Threshold_df <- metrics_df %>% select(F1,Threshold)
RF_Threshold <- RF_Threshold_df[ which.max(RF_Threshold_df$F1) ,2]
RF_Threshold
```

```
## # A tibble: 1 x 1
##   Threshold
##       <dbl>
## 1       0.4
```

Let us also plot ROC curve which is sensitivity (TPR) versus 1-specificity or the false positive rate (FPR).

In addition, to plot precision-recall/sensitivity to make sure that we are also maintain precision.

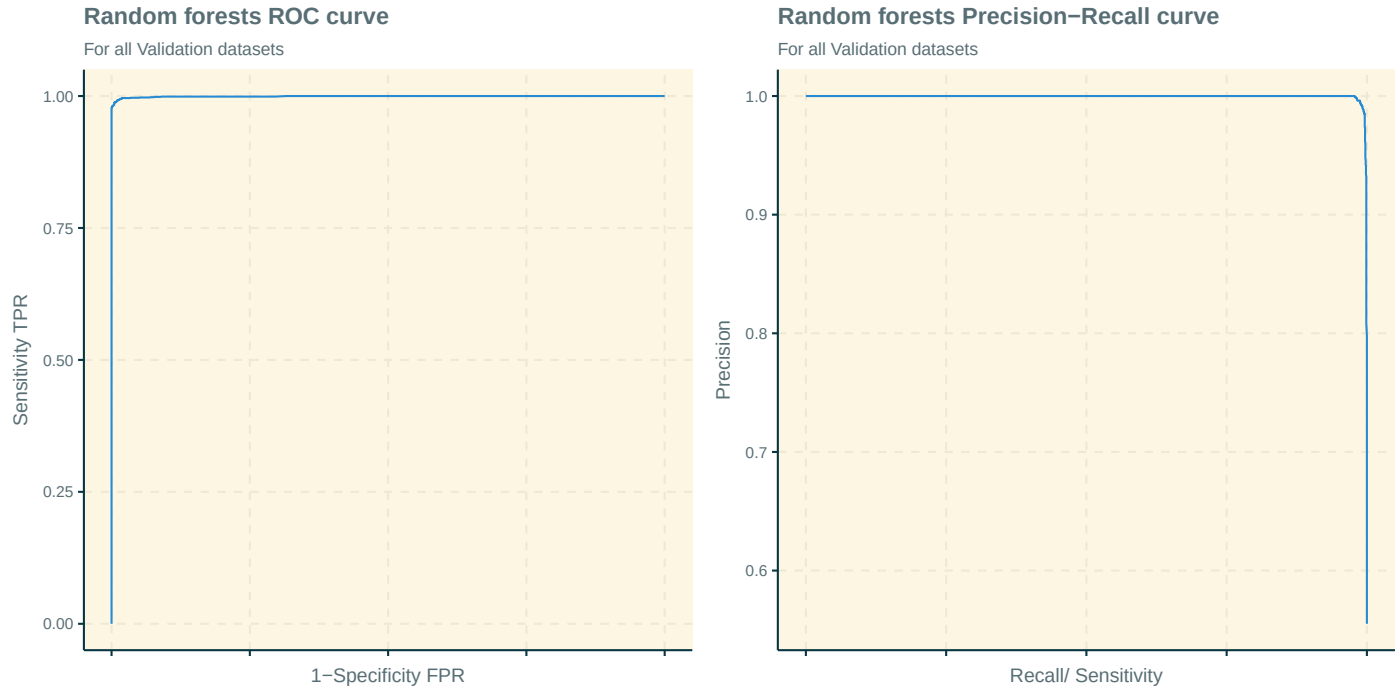
Before plotting curves, let us calculate points data for each curve.

```
# Calculate Points for each curve and convert S3 object to dataframe for easy plotting

perf_curves <- evalmod(scores = RF_model_2$pred$Genuine, labels = RF_model_2$pred$obs)

perf_curves_df <- fortify(perf_curves)
```

Now, let us check ROC curve.

```
# Plot ROC curve

plot1 <- perf_curves_df %>% filter (curvetype=="ROC")%>% ggplot(aes(x, y)) +
```

```
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Random forests ROC curve")+
labs(subtitle ="For all Validation datasets" ,
y="Sensitivity TPR", x="1-Specificity FPR")

plot2 <- perf_curves_df %>% filter (curvetype=="PRC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Random forests Precision-Recall curve")+
labs(subtitle ="For all Validation datasets" ,
y="Precision", x="Recall/ Sensitivity")

grid.arrange(plot1, plot2, ncol=2)
```



From previous graphs to is obvious that we have high value for AUC in both curves almost 1.

ROC can be calculated using the following:

```
aucs <- auc(perf_curves)
knitr::kable(aucs)
```

modnames	dsids	curvetypes	aucs
m1	1	ROC	0.99935
m1	1	PRC	0.99954

2.6.3 Random forests Prediction

Now, it is time for prediction on unseen data of the test dataset.

```
# Predict on Test set

test_pred_prob <- predict(RF_model_2,test_set[,1:3],type="prob")

# Get Fraudulent Probability

test_pred_prob <- test_pred_prob %>% pull(Fraudulent)

# Get Predicted Class

test_pred_class <- ifelse(test_pred_prob >= as.numeric(RF_Threshold) , "Fraudulent","Genuine")

# Build confusionMatrix

cf_mtx <-  confusionMatrix(factor(test_pred_class),factor(test_set$Class))
```

Now, let us add to measures table of each algorithm.

```
# Build a Table to save all Performance Measures

performance_tb <- rbind(performance_tb, tibble(method = "Random forests",
F1 = cf_mtx$byClass["F1"], Sensitivity=cf_mtx$byClass["Sensitivity"],
Specificity=cf_mtx$byClass["Specificity"] ,Precision=cf_mtx$byClass["Precision"] ,
Balanced_Accuracy= cf_mtx$byClass["Balanced Accuracy"] ))

kable(performance_tb,"latex", booktabs = T)%>%
kable_styling(latex_options = "striped")
```

method	F1	Sensitivity	Specificity	Precision	Balanced_Accuracy
Logistic Regression	0.85950	0.85246	0.89542	0.86667	0.87394
k-nearest neighbors	1.00000	1.00000	1.00000	1.00000	1.00000
Support Vector Machine	0.98785	1.00000	0.98039	0.97600	0.99020
Random forests	0.98374	0.99180	0.98039	0.97581	0.98610

2.7 Neural Network

Now, we can start with our Fifth algorithm Neural Network.

2.7.1 Neural Network Training

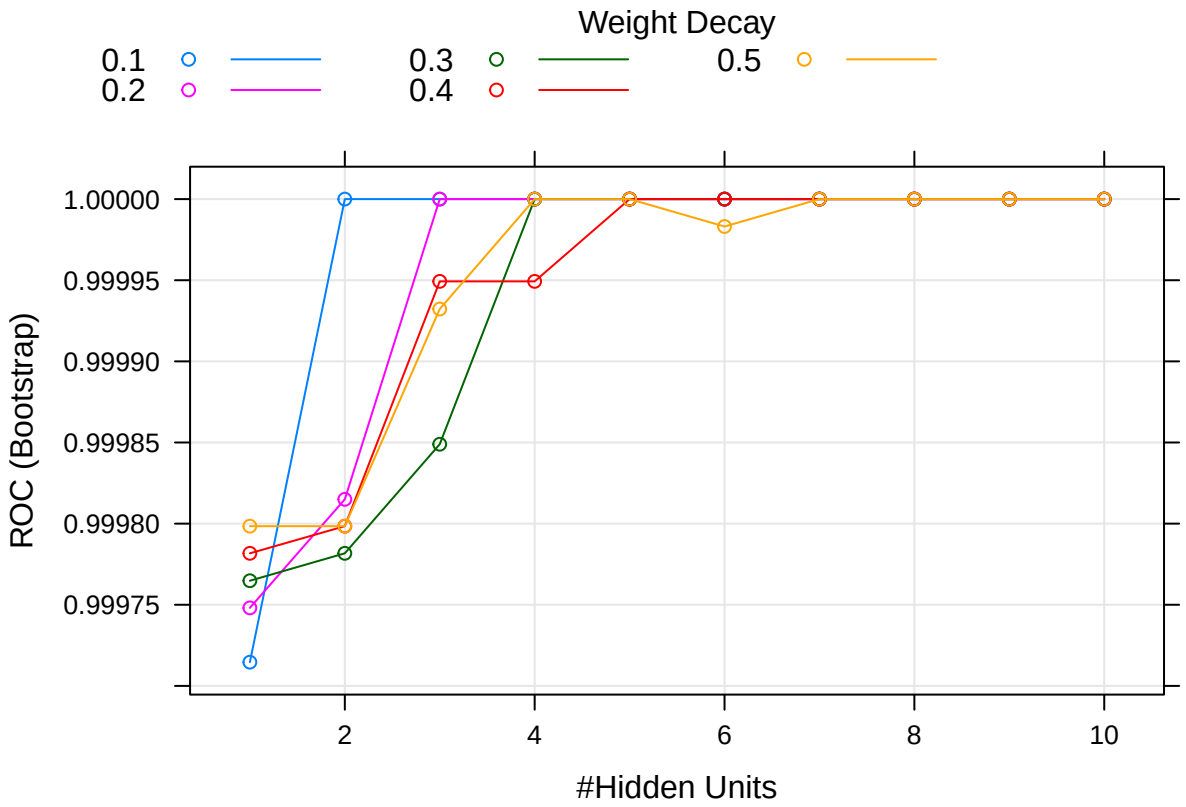
We will use the same control while training Logistic Regression algorithm. However, in nnet we have a selected parameters to tune which are Number of units in the hidden layer (size),the weight penalty parameter for used as regularization parameter to avoid over-fitting (decay).

Training will start by 4 predictors

```
# Train nnet

NN_tune <-  expand.grid(size = seq(from = 1, to = 10, by = 1),
                        decay = seq(from = 0.1, to = 0.5, by = 0.1))
```

```
NN_model <- train(Class ~ ., method = "nnet", data = train_set,trControl = Control,
tuneGrid = NN_tune,trace=FALSE)
```

```
# Plot ROC against mtry
plot(NN_model)
```



we can see that we can almost get ROC equals to one by using size equals to 3 and any decay equals 0.2.

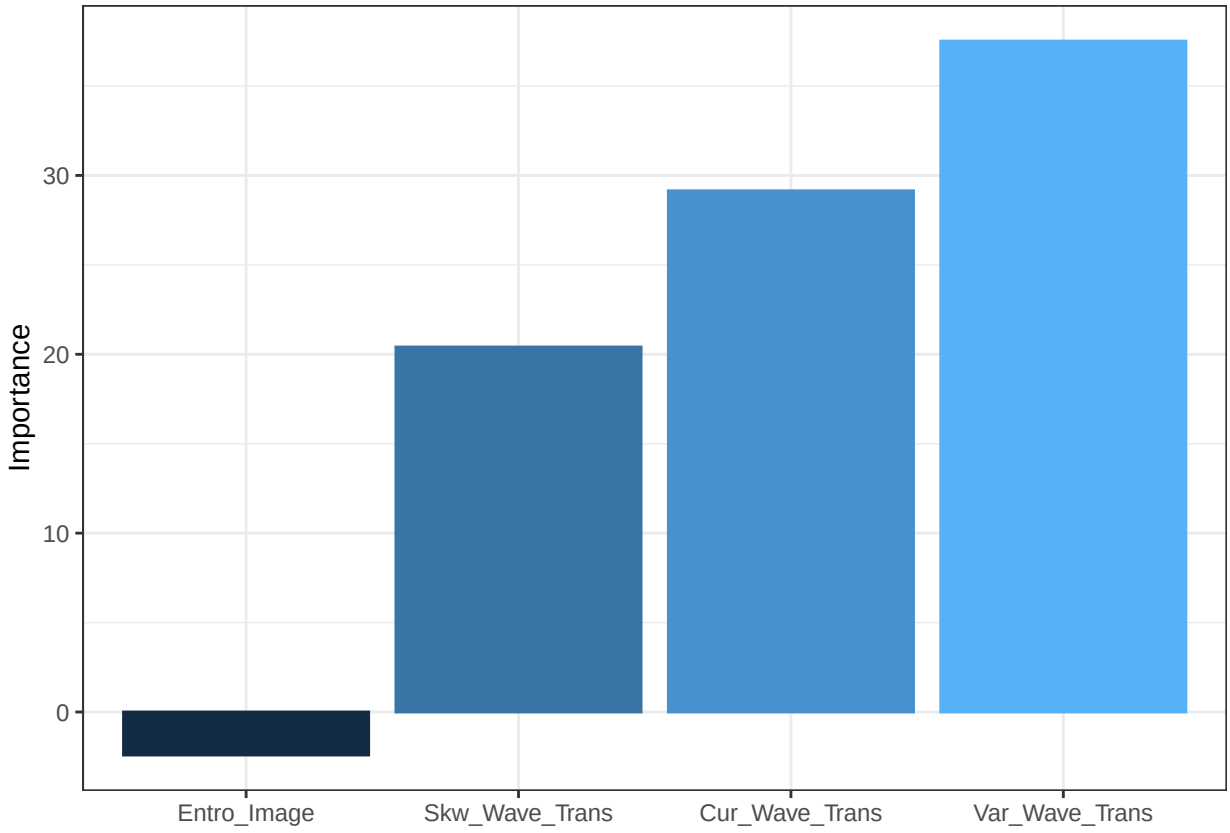
2.7.2 Neural Network Tuning

In that section, we will check the following:

- Number of Predictors/ Variables
- Best Threshold or Cut-off

First, let us check variable importance for 4 Predictors model.

```
# Check each Predictor importance

olden(NN_model)
```



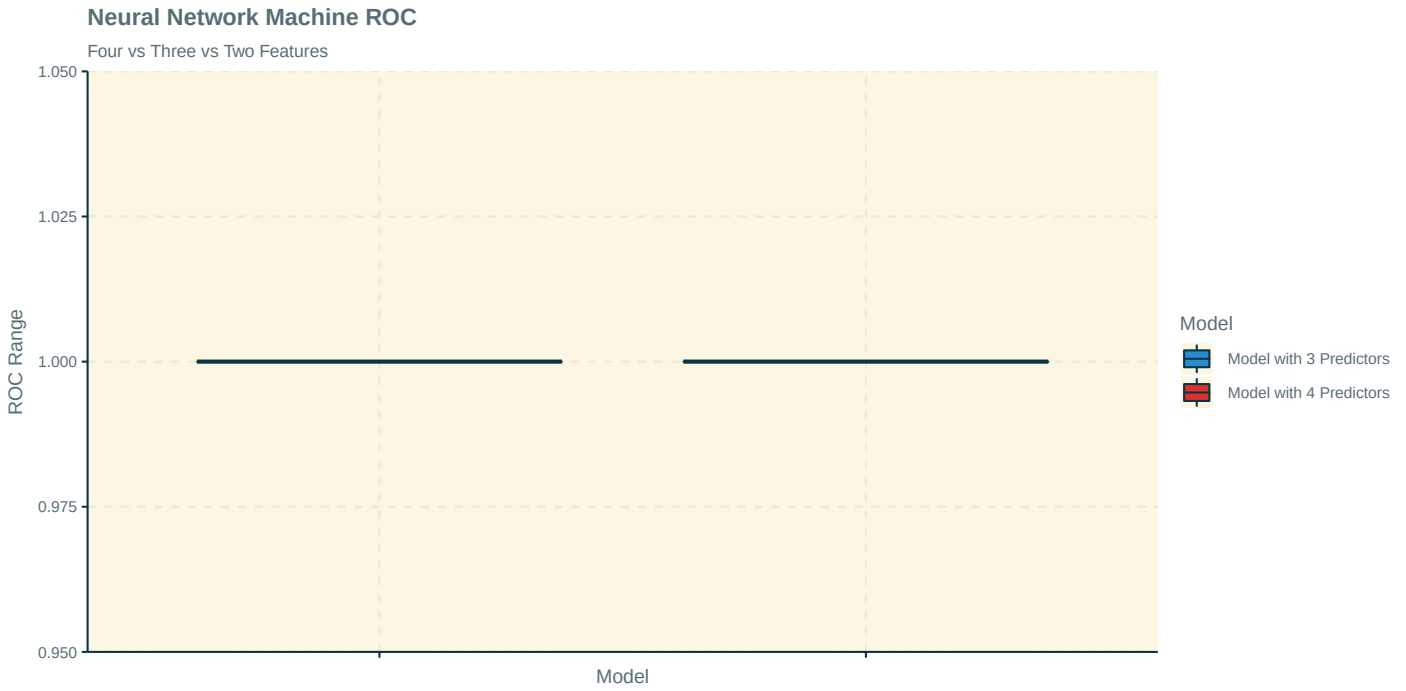
First, let us check train our model by 3 Predictors as we have done in logistic Regression.

- let us try train our model by excluding Entro_Image Feature.

```
# Train with 3 Predictors

NN_model_2 <- train(Class ~ Var_Wave_Trans + Skw_Wave_Trans + Cur_Wave_Trans,
  method = "nnet", data = train_set,trControl = Control,tuneGrid = NN_tune,trace=FALSE)
```

Now we can compare ROC in the three models.

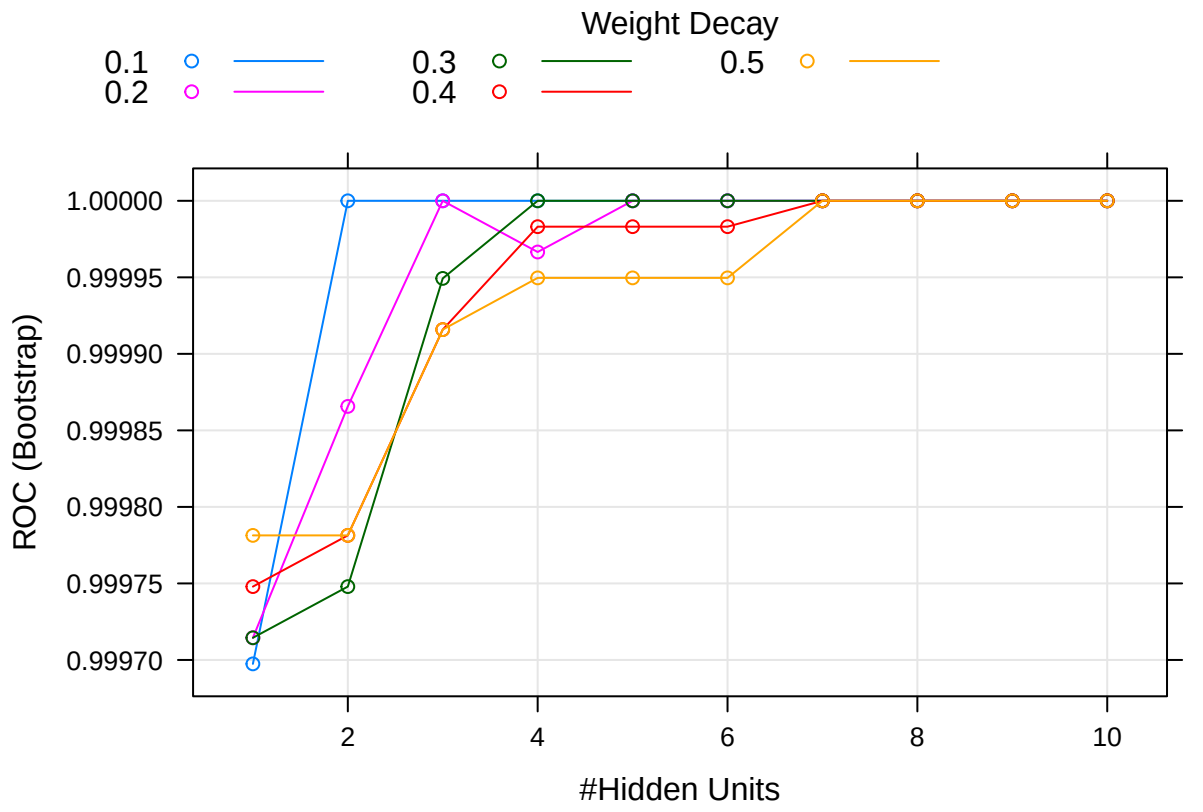
```
# Combined Accuracy in on dataset

Comp_models <- rbind(NN_model$resample %>% mutate(Model=" Model with 4 Predictors"),
                          NN_model_2$resample %>% mutate(Model=" Model with 3 Predictors"))

# Plot Accuracy Ranges colored by Model

Comp_models %>% ggplot(aes(Model, ROC, fill = Model)) +
geom_boxplot() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Neural Network Machine ROC")+
labs(subtitle ="Four vs Three vs Two Features" ,
y="ROC Range")
```



We can see that the ROC with 4 Predictors is same as 3 Predictors.

Hence, We can conclude that the improvement in the model with 4 Predictors is not considerable and we will select the model with 3 Predictors going forward in nnet. Hence, let us see how many units in the hidden layer for 3 Predictors model.

```
plot(NN_model_2)
```



We can see that we will use 2 units in the hidden layer.

Second, Best Threshold or Cut-off. By Default 0.5 is chosen as Threshold but let us examine if we can achieve better performance by other Threshold.

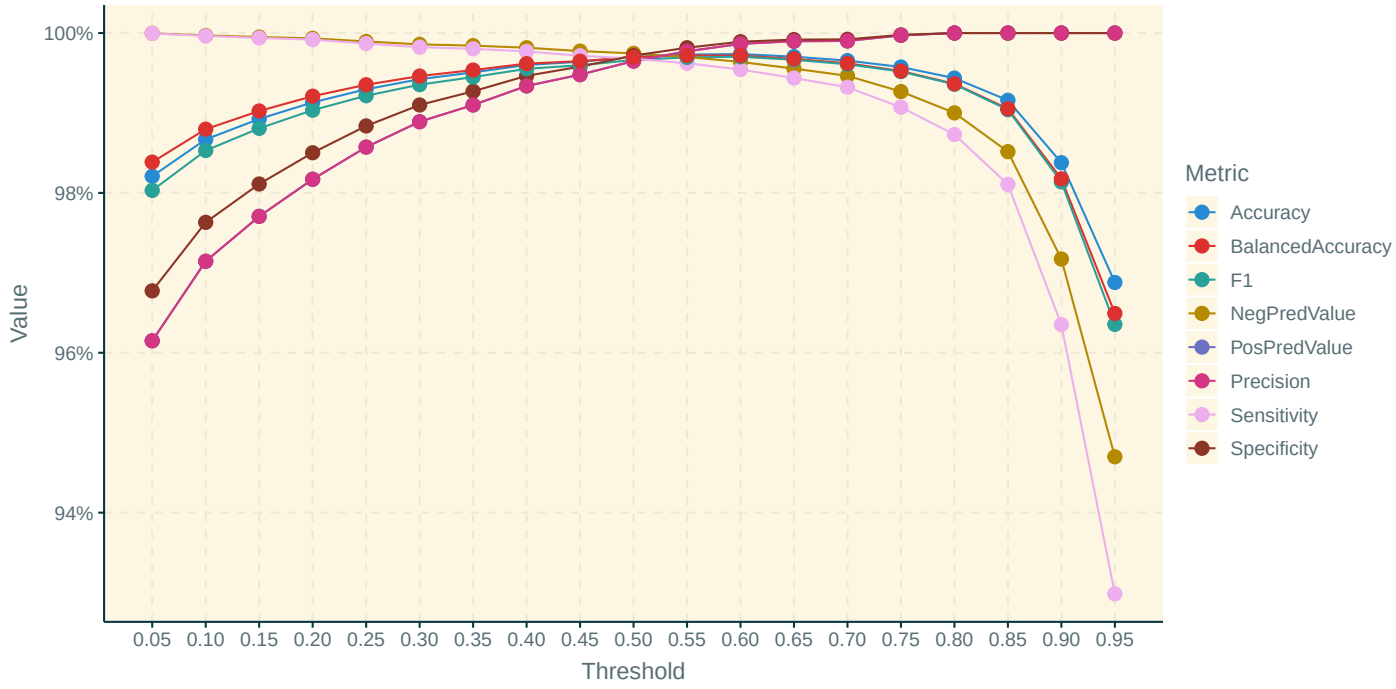
Now, it is time to try Threshold from 0.1 to 0.95

```
# Try Threshold from 0.1 to 0.95

metrics_df <- lapply(seq(0.05, 0.95, by = 0.05),TryThreshold, NN_model_2)

# Convert result to dataframe

metrics_df <- do.call("bind_rows", metrics_df)

# Add Threshold to the Performance Measure dataframe
# Each Threshold used for 5 times validation, so this why we replicate each Threshold for Threshold

metrics_df <- metrics_df %>% mutate(Threshold = lapply(seq(0.05, 0.95, by = 0.05),
  function(x) {rep(x, 5)})%>% unlist())
```

It will be better of Visualize the Performance Measures as the following:

```
#For each Threshold, Get Average of each Measure across 5 Validation datasets

metrics_df <- metrics_df %>%
  select(Accuracy, NegPredValue, PosPredValue, Sensitivity,
```

```
  Specificity,F1,Precision,BalancedAccuracy,Threshold) %>%
  group_by(Threshold) %>%
  summarise_at(vars(Accuracy, NegPredValue, PosPredValue, Sensitivity,
  Specificity,F1,Precision,BalancedAccuracy), mean)

#Plot measures against Threshold

  metrics_df %>%gather(Metric, Value, -Threshold) %>%
  ggplot(aes(Threshold, Value, color = Metric)) +
  geom_line() +
  geom_point(size = 3) +
 scale_y_continuous(labels = scales::percent_format(accuracy = 1))+
  scale_x_continuous(breaks = seq(0.05, 0.95, by = 0.05))
```



As we have two classed, let us choose the Threshold that maximize F1 score.

```
#Get Threshold that maximize F1 score

NN_Threshold_df <- metrics_df %>% select(F1,Threshold)
NN_Threshold <- NN_Threshold_df[ which.max(NN_Threshold_df$F1) ,2]
NN_Threshold
```

```
## # A tibble: 1 x 1
##   Threshold
##       <dbl>
## 1       0.6
```

Let us also plot ROC curve which is sensitivity (TPR) versus 1-specificity or the false positive rate (FPR).

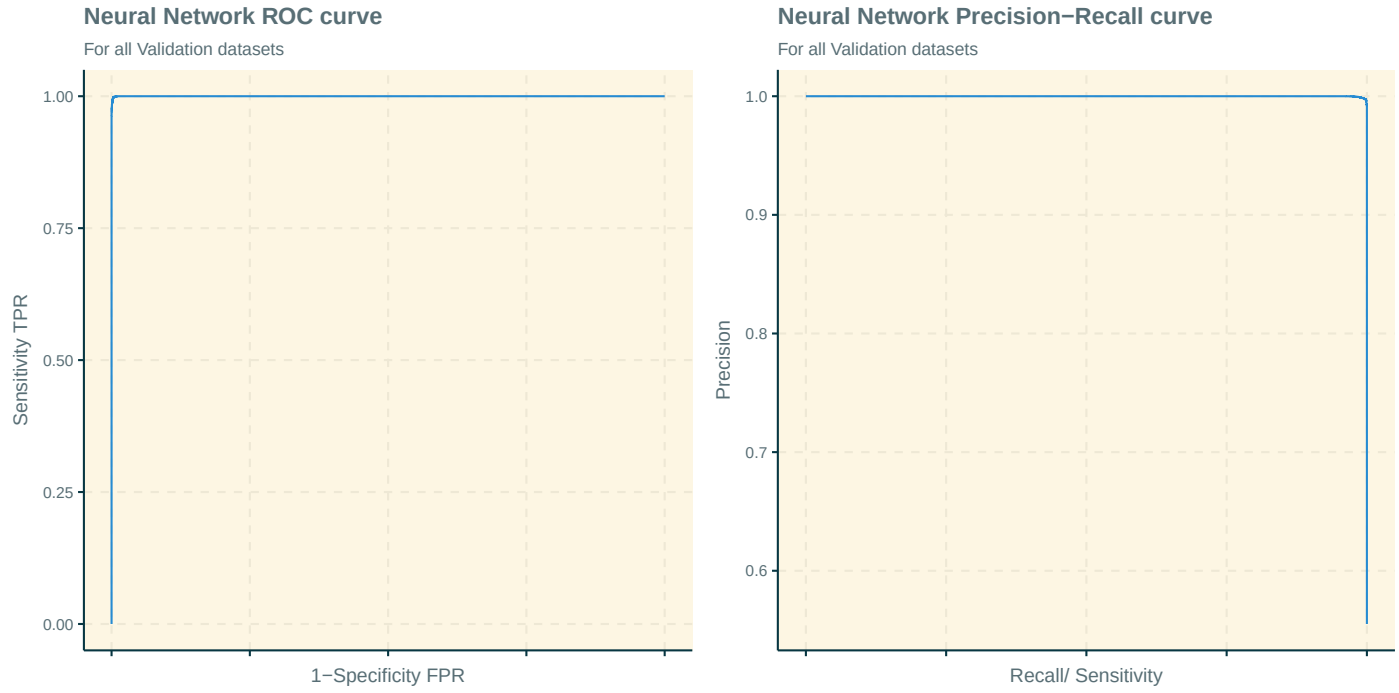
In addition, to plot precision-recall/sensitivity to make sure that we are also maintain precision.

Before plotting curves, let us calculate points data for each curve.

```
# Calculate Points for each curve and convert S3 object to dataframe for easy plotting

perf_curves <- evalmod(scores = NN_model_2$pred$Genuine, labels = NN_model_2$pred$obs)

perf_curves_df <- fortify(perf_curves)
```

Now, let us check ROC curve.

```
# Plot ROC curve

plot1 <- perf_curves_df %>% filter (curvetype=="ROC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Neural Network ROC curve")+
labs(subtitle ="For all Validation datasets" ,
y="Sensitivity TPR", x="1-Specificity FPR")

plot2 <- perf_curves_df %>% filter (curvetype=="PRC")%>% ggplot(aes(x, y)) +
geom_line() +
theme(axis.text.x = element_blank()) +
theme(text = element_text(size=10))+
ggtitle("Neural Network Precision-Recall curve")+
labs(subtitle ="For all Validation datasets" ,
y="Precision", x="Recall/ Sensitivity")

grid.arrange(plot1, plot2, ncol=2)
```



From previous graphs to is obvious that we have high value for AUC in both curves almost 1.

ROC can be calculated using the following:

```
aucs <- auc(perf_curves)
knitr::kable(aucs)
```

modnames	dsids	curvetypes	aucs
m1	1	ROC	0.99996
m1	1	PRC	0.99997

2.7.3 Neural Network Prediction

Now, it is time for prediction on unseen data of the test dataset.

```
# Predict on Test set

test_pred_prob <- predict(NN_model_2,test_set[,1:3],type="prob")

# Get Fraudulent Probability

test_pred_prob <- test_pred_prob %>% pull(Fraudulent)

# Get Predicted Class

test_pred_class <- ifelse(test_pred_prob >= as.numeric(NN_Threshold) , "Fraudulent","Genuine")

# Build confusionMatrix

cf_mtx <-  confusionMatrix(factor(test_pred_class),factor(test_set$Class))
```

Now, let us add to measures table of each algorithm.

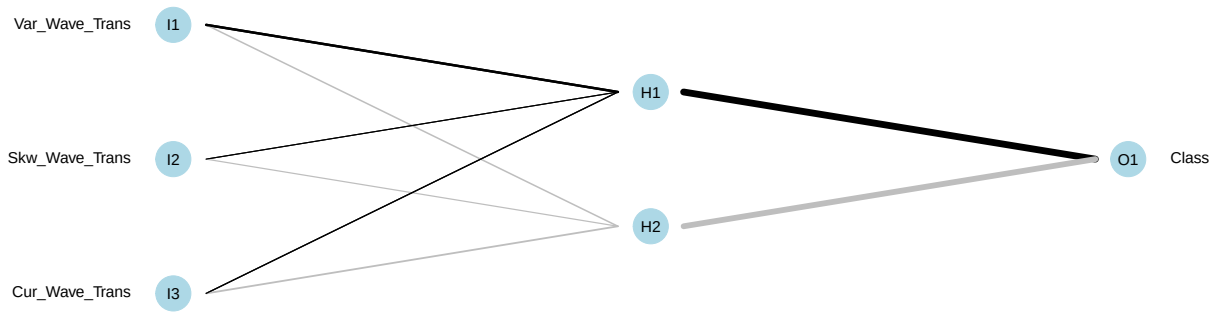
```
# Build a Table to save all Performance Measures

performance_tb <- rbind(performance_tb, tibble(method = "Neural Network",
F1 = cf_mtx$byClass["F1"], Sensitivity=cf_mtx$byClass["Sensitivity"],
Specificity=cf_mtx$byClass["Specificity"] ,Precision=cf_mtx$byClass["Precision"] ,
Balanced_Accuracy= cf_mtx$byClass["Balanced Accuracy"] ))

kable(performance_tb,"latex", booktabs = T)%>%
kable_styling(latex_options = "striped")
```

method	F1	Sensitivity	Specificity	Precision	Balanced_Accuracy
Logistic Regression	0.85950	0.85246	0.89542	0.86667	0.87394
k-nearest neighbors	1.00000	1.00000	1.00000	1.00000	1.00000
Support Vector Machine	0.98785	1.00000	0.98039	0.97600	0.99020
Random forests	0.98374	0.99180	0.98039	0.97581	0.98610
Neural Network	1.00000	1.00000	1.00000	1.00000	1.00000

3 Results

From the above final summary of the results from all algorithms, we can conclude that k-nearest neighbors and Neural Network give us the best values for all the measures F1 Score, Sensitivity, Specificity, Precision and Balanced Accuracy. However, Neural Network use only three Predictors. Let us plot the Neural Network to get more inside our results.

```
plotnet(NN_model_2 ,bias=FALSE)
```



First, we can see that Input Layer has three Predictors while the hidden Layer has two units as per tuning result and Output Layer has one unit hold class Probability.

As expected from the analysis section, the biggest weight is for Var_Wave_Trans as it has the darkest line to the middle layer.

On the other hand, Support vector Machine and Random Forest did very well with high values for all our measures

Finally, Logistic Regression was the lowest performance. However, we trained it using only two Predictors and the Features were not having a solid Linear relationship with logit of the outcome.

4 Conclusion

We have examined different Machine Learning algorithm to check banknote authentication

All algorithms results excellent performance measures. We have seen also that many predicted Classes is almost similar to actual Classes.

These achieved results, drop the light on the value of Machine Learning for recognizing the banknote as genuine or Fraudulent by using supervised machine learning techniques.

The presented algorithms were implemented on basic PC with 16 GB of RAM. Better results could be achieved using high end PC or server that allows more processing for a bigger dataset or more iterations of cross validation.

With high-end server, there is a possibility to do future work by getting a bigger dataset and combining prediction from different algorithms and choose the best performance by Ensembles.

Finally, the value of Machine Learning to many business areas that require banknote or any ID authentication is crucial and required to be utilized effectively. Hence, business can speed growth process and gain more profit through predicting fake documents or IDs.

On the other hand, catching circulated fake banknote will protect economy from reduction in the value of real money, and increase in prices (inflation) due to more fake money getting circulated in the economy.

5 References

[1]https://www.ijcaonline.org/archives/volume179/number20/shahani-2018-ijca-916343.pdf

[2]https://www.cnbc.com/2015/09/02/can-you-find-the-forgery.html

[3]Irizzary,R., 2018,Introduction to Data Science,github,https://rafalab.github.io/dsbook/

6 GitHub

- https://github.com/sherif-allam/BanknoteAuthentication