

Credit Card Fraudulent Detection_ Final Project

Vasu Gandhapudi

September 06, 2020

Overview: A Gentle Introduction:

This project is designed as a final project submission towards Harvardx data science learning initiative. We use the knowledge gained in the course series and its interdisciplinary area to apply data knowledge using scientific methods, techniques, models and algorithms.

Aim of the project

The aim of the project is to build a classifier and discover the **Credit Card Fraudulent transaction** which is a major concern in most of the financial institutes. Analysing fraudulent manually is unfeasible due to huge amount of data and it's complexity.

Hence we want to make the process ease we will apply some of the modern ML Models,algorithms and techniques in this project.

In the process of finding data information, we use some of the popular Machine Learning methods and techniques, to name: data cleaning, data discovery, data visualisation and some model approaches.

this data set contains transactions made by credit card in september 2013. this dataset was originally maintained by [www. kaggle.com/](https://www.kaggle.com/). which it has 492 frauds and 284807 transactions.

This dataset is highly unbalanced.

It contains only numerical variables which are results for PCA transactions, and this dataset contains features from V1 to V28 are the principal component obtained with PCA. the only features which are not been transformed are "Time" and "Amount". Feature "Class" is the response variable and it takes values 1 and 0.

some of the models that we use in this project are: 1. Logistic Regression Model. 2. Decision Tree Model. 3. K-Fold Cross Validation. 4. Random Forest. 5. XGBoost.

Dataset download urls

this datasets can be obtained from the below links:

<https://ln2.sync.com/dl/cc02c4800/pv3day6w-i2xn38sr-izd9apca-dsxrqhnd/view/default/9384339740003>

<https://www.kaggle.com/vasugv/credit-card-fraud-detection>

<https://www.kaggle.com/mlg-ulb/creditcardfraud>

<https://www.researchgate.net/project/Fraud-detection-with-machine-learning>

Note: This dataset was purely intended and stored in below Git hub account is for learning purpose

<https://github.com/vasu0907/Datasets>

Project Librarys

```
### Import Library's for the project

if(!require(dplyr))
  install.packages("dplyr", repos = "https://dplyr.tidyverse.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2))
  install.packages("ggplot2", repos = "http://ggplot2.tidyverse.org")
if(!require(pROC))
  install.packages("pROC", repos = "https://web.expasy.org/pROC/")
if(!require(rpart))
  install.packages("rPart", repos = "https://cran.r-project.org/package=rpart")
if(!require(rpart.plot))
  install.packages("rpart.plot", repos = "http://www.milbo.org/rpart-plot")
if(!require(gbm))
  install.packages("gbm", repos = "https://github.com/gbm-developers/gbm")
if(!require(randomForest))
  install.packages("randomForest", repos = "https://www.stat.berkeley.edu/~breiman/RandomForests/")
if(!require(xgboost))
  install.packages("xgboost", repos = "https://github.com/dmlc/xgboost")
if(!require(caTools))
  install.packages("caTools", repos = "https://CRAN.R-project.org/package=caTools")
if(!require(ranger)) install.packages("ranger", repos = "https://github.com/imbs-hl/ranger")
```

Download Dataset

```
## For demonstration and better access, The dataset was download and placed in Github account.

creditcard_data <- fread("https://media.githubusercontent.com/media/vasu0907/Datasets/master/creditcard
```

Data Exploration:- Revising R concepts on df.

```
dim(creditcard_data)

## [1] 284807      31

head(creditcard_data, 3)

##   Time        V1        V2        V3        V4        V5        V6
## 1: 0 -1.359807 -0.07278117 2.5363467 1.3781552 -0.33832077 0.46238778
## 2: 0  1.191857  0.26615071 0.1664801 0.4481541  0.06001765 -0.08236081
## 3: 1 -1.358354 -1.34016307 1.7732093 0.3797796 -0.50319813  1.80049938
```

```

##          V7          V8          V9          V10         V11         V12
## 1:  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2: -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3:  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
##          V13         V14         V15         V16         V17         V18         V19
## 1: -0.9913898 -0.3111694  1.4681770 -0.4704005  0.2079712  0.02579058  0.403993
## 2:  0.4890950 -0.1437723  0.6355581  0.4639170 -0.1148047 -0.18336127 -0.145783
## 3:  0.7172927 -0.1659459  2.3458649 -2.8900832  1.1099694 -0.12135931 -2.261857
##          V20         V21         V22         V23         V24         V25
## 1:  0.25141210 -0.01830678  0.2778376 -0.1104739  0.06692807  0.1285394
## 2: -0.06908314 -0.22577525 -0.6386720  0.1012880 -0.33984648  0.1671704
## 3:  0.52497973  0.24799815  0.7716794  0.9094123 -0.68928096 -0.3276418
##          V26         V27         V28 Amount Class
## 1: -0.1891148  0.133558377 -0.02105305 149.62      0
## 2:  0.1258945 -0.008983099  0.01472417  2.69      0
## 3: -0.1390966 -0.055352794 -0.05975184 378.66      0

```

```
tail(creditcard_data, 3)
```

```

##          Time         V1         V2         V3         V4         V5         V6
## 1: 172788  1.9195650 -0.3012538 -3.2496398 -0.5578281  2.63051512  3.0312601
## 2: 172788 -0.2404400  0.5304825  0.7025102  0.6897992 -0.37796113  0.6237077
## 3: 172792 -0.5334125 -0.1897333  0.7033374 -0.5062712 -0.01254568 -0.6496167
##          V7         V8         V9         V10        V11        V12        V13
## 1: -0.2968265  0.7084172  0.4324540 -0.4847818  0.4116137  0.06311886 -0.1836987
## 2: -0.6861800  0.6791455  0.3920867 -0.3991257 -1.9338488 -0.96288614 -1.0420817
## 3:  1.5770063 -0.4146504  0.4861795 -0.9154266 -1.0404583 -0.03151305 -0.1880929
##          V14        V15        V16        V17        V18        V19        V20
## 1: -0.51060184  1.32928351  0.1407160  0.3135018  0.3956525 -0.5772518  0.00139597
## 2:  0.44962444  1.96256312 -0.6085771  0.5099285  1.1139806  2.8978488  0.12743352
## 3: -0.08431647  0.04133346 -0.3026201 -0.6603766  0.1674299 -0.2561169  0.38294810
##          V21        V22        V23        V24        V25        V26
## 1:  0.2320450  0.5782290 -0.03750086  0.640133881  0.2657455 -0.0873706
## 2:  0.2652449  0.8000487 -0.16329794  0.123205244 -0.5691589  0.5466685
## 3:  0.2610573  0.6430784  0.37677701  0.008797379 -0.4736487 -0.8182671
##          V27        V28 Amount Class
## 1:  0.004454772 -0.02656083  67.88      0
## 2:  0.108820735  0.10453282  10.00      0
## 3: -0.002415309  0.01364891 217.00      0

```

```
names(creditcard_data)
```

```

## [1] "Time"      "V1"       "V2"       "V3"       "V4"       "V5"       "V6"       "V7"
## [9] "V8"        "V9"       "V10"      "V11"      "V12"      "V13"      "V14"      "V15"
## [17] "V16"      "V17"      "V18"      "V19"      "V20"      "V21"      "V22"      "V23"
## [25] "V24"      "V25"      "V26"      "V27"      "V28"      "Amount"   "Class"

```

```
summary(creditcard_data)
```

```

##          Time              V1              V2              V3
##  Min.   : 0   Min.   :-56.40751   Min.   :-72.71573   Min.   :-48.3256
##  1st Qu.: 54202  1st Qu.:-0.92037  1st Qu.:-0.59855  1st Qu.:-0.8904

```

```

## Median : 84692   Median : 0.01811   Median : 0.06549   Median : 0.1799
## Mean   : 94814   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
## 3rd Qu.:139321  3rd Qu.: 1.31564   3rd Qu.: 0.80372   3rd Qu.: 1.0272
## Max.   :172792   Max.   : 2.45493   Max.   : 22.05773   Max.   : 9.3826
##          V4           V5           V6           V7
## Min.   :-5.68317  Min.   :-113.74331  Min.   :-26.1605  Min.   :-43.5572
## 1st Qu.:-0.84864  1st Qu.: -0.69160  1st Qu.: -0.7683  1st Qu.: -0.5541
## Median :-0.01985  Median : -0.05434  Median : -0.2742  Median : 0.0401
## Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.0000  Mean   : 0.0000
## 3rd Qu.: 0.74334  3rd Qu.: 0.61193  3rd Qu.: 0.3986  3rd Qu.: 0.5704
## Max.   :16.87534  Max.   : 34.80167  Max.   : 73.3016  Max.   :120.5895
##          V8           V9           V10          V11
## Min.   :-73.21672  Min.   :-13.43407  Min.   :-24.58826  Min.   :-4.79747
## 1st Qu.:-0.20863  1st Qu.: -0.64310  1st Qu.: -0.53543  1st Qu.: -0.76249
## Median : 0.02236  Median : -0.05143  Median : -0.09292  Median : -0.03276
## Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.00000
## 3rd Qu.: 0.32735  3rd Qu.: 0.59714  3rd Qu.: 0.45392  3rd Qu.: 0.73959
## Max.   :20.00721  Max.   : 15.59500  Max.   : 23.74514  Max.   :12.01891
##          V12          V13          V14          V15
## Min.   :-18.6837  Min.   :-5.79188  Min.   :-19.2143  Min.   :-4.49894
## 1st Qu.:-0.4056  1st Qu.: -0.64854  1st Qu.: -0.4256  1st Qu.: -0.58288
## Median : 0.1400  Median : -0.01357  Median : 0.0506  Median : 0.04807
## Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.0000  Mean   : 0.00000
## 3rd Qu.: 0.6182  3rd Qu.: 0.66251  3rd Qu.: 0.4931  3rd Qu.: 0.64882
## Max.   : 7.8484  Max.   : 7.12688  Max.   : 10.5268  Max.   : 8.87774
##          V16          V17          V18
## Min.   :-14.12985  Min.   :-25.16280  Min.   :-9.498746
## 1st Qu.:-0.46804  1st Qu.: -0.48375  1st Qu.: -0.498850
## Median : 0.06641  Median : -0.06568  Median : -0.003636
## Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.000000
## 3rd Qu.: 0.52330  3rd Qu.: 0.39968  3rd Qu.: 0.500807
## Max.   : 17.31511  Max.   : 9.25353  Max.   : 5.041069
##          V19          V20          V21
## Min.   :-7.213527  Min.   :-54.49772  Min.   :-34.83038
## 1st Qu.:-0.456299  1st Qu.: -0.21172  1st Qu.: -0.22839
## Median : 0.003735  Median : -0.06248  Median : -0.02945
## Mean   : 0.000000  Mean   : 0.00000  Mean   : 0.00000
## 3rd Qu.: 0.458949  3rd Qu.: 0.13304  3rd Qu.: 0.18638
## Max.   : 5.591971  Max.   : 39.42090  Max.   : 27.20284
##          V22          V23          V24
## Min.   :-10.933144  Min.   :-44.80774  Min.   :-2.83663
## 1st Qu.:-0.542350  1st Qu.: -0.16185  1st Qu.: -0.35459
## Median : 0.006782  Median : -0.01119  Median : 0.04098
## Mean   : 0.000000  Mean   : 0.00000  Mean   : 0.00000
## 3rd Qu.: 0.528554  3rd Qu.: 0.14764  3rd Qu.: 0.43953
## Max.   : 10.503090  Max.   : 22.52841  Max.   : 4.58455
##          V25          V26          V27
## Min.   :-10.29540  Min.   :-2.60455  Min.   :-22.565679
## 1st Qu.:-0.31715  1st Qu.: -0.32698  1st Qu.: -0.070840
## Median : 0.01659  Median : -0.05214  Median : 0.001342
## Mean   : 0.00000  Mean   : 0.00000  Mean   : 0.000000
## 3rd Qu.: 0.35072  3rd Qu.: 0.24095  3rd Qu.: 0.091045
## Max.   : 7.51959  Max.   : 3.51735  Max.   : 31.612198
##          V28          Amount          Class

```

```

##  Min.   : -15.43008   Min.   :  0.00   Min.   :0.000000
##  1st Qu.: -0.05296   1st Qu.:  5.60   1st Qu.:0.000000
##  Median :  0.01124   Median : 22.00   Median :0.000000
##  Mean   :  0.00000   Mean   : 88.35   Mean   :0.001728
##  3rd Qu.:  0.07828   3rd Qu.: 77.17   3rd Qu.:0.000000
##  Max.   : 33.84781   Max.   :25691.16  Max.   :1.000000

```

Standard deviation for the value name “Amount”

```
sd(creditcard_data$Amount)
```

```
## [1] 250.1201
```

Data Manipulation

```
apply(creditcard_data, 2, function(x) sum(is.na(x)))
```

check is there any NA values in the dataset

```

##   Time    V1    V2    V3    V4    V5    V6    V7    V8    V9    V10
##   0      0     0     0     0     0     0     0     0     0     0
##   V11   V12   V13   V14   V15   V16   V17   V18   V19   V20   V21
##   0      0     0     0     0     0     0     0     0     0     0
##   V22   V23   V24   V25   V26   V27   V28 Amount Class
##   0      0     0     0     0     0     0     0     0     0

```

```
creditcard_data %>% group_by(Class) %>% summarise(mean(Amount), median(Amount))
```

```

## # A tibble: 2 x 3
##   Class `mean(Amount)` `median(Amount)`
##   <int>      <dbl>        <dbl>
## 1     0        88.3        22
## 2     1       122.         9.25

```

We will scale our data using the scale() function. We will apply this to the amount component of our creditcard_data amount.

Scaling is also known as feature standardization. With the help of scaling, the data is structured according to a specified range.

Therefore, there are no extreme values in our dataset that might interfere with the functioning of our model.

```
creditcard_data$Amount=scale(creditcard_data$Amount)
NewData=creditcard_data[,-c(1)]
```

Data Modeling

```

## Split data set into train and test
set.seed(123)
data_sample <- sample.split(NewData$Class, SplitRatio = 0.80)
train_dataset <- subset(NewData, data_sample == TRUE)
test_dataset <- subset(NewData, data_sample == FALSE)
dim(train_dataset)

```

```
## [1] 227846      30
```

```
dim(test_dataset)
```

```
## [1] 56961      30
```

Modeling technique for optimize & Algorithms to predict.

Fitting Logistic Regression Model :1

Logistic regression is a simple regression model whose output is a score between 0 and 1.

This model can be fitted using Gradient descent on the parameter vector beta. Equipped with some basic information.

```

Logistic_Model=glm(Class~., train_dataset, family = binomial())
summary(Logistic_Model)

```

```

##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = train_dataset)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -4.6108  -0.0292  -0.0194  -0.0125   4.6021
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.651305  0.160212 -53.999 < 2e-16 ***
## V1          0.072540  0.044144   1.643  0.100332
## V2          0.014818  0.059777   0.248  0.804220
## V3          0.026109  0.049776   0.525  0.599906
## V4          0.681286  0.078071   8.726 < 2e-16 ***
## V5          0.087938  0.071553   1.229  0.219079
## V6         -0.148083  0.085192  -1.738  0.082170 .
## V7         -0.117344  0.068940  -1.702  0.088731 .
## V8         -0.146045  0.035667  -4.095 4.23e-05 ***
## V9          -0.339828  0.117595  -2.890  0.003855 **
## V10         -0.785462  0.098486  -7.975 1.52e-15 ***
## V11         0.001492  0.085147   0.018  0.986018
## V12         0.087106  0.094869   0.918  0.358532
## V13         -0.343792  0.092381  -3.721  0.000198 ***
## V14         -0.526828  0.067084  -7.853 4.05e-15 ***
## V15         -0.095471  0.094037  -1.015  0.309991

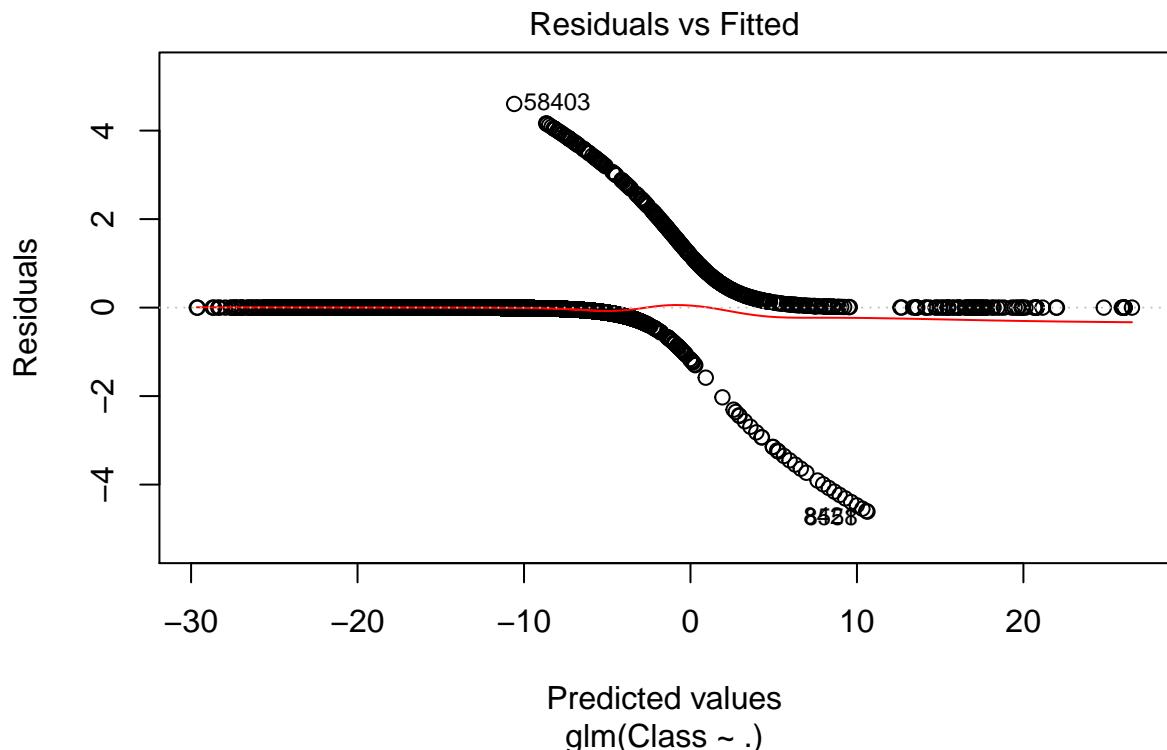
```

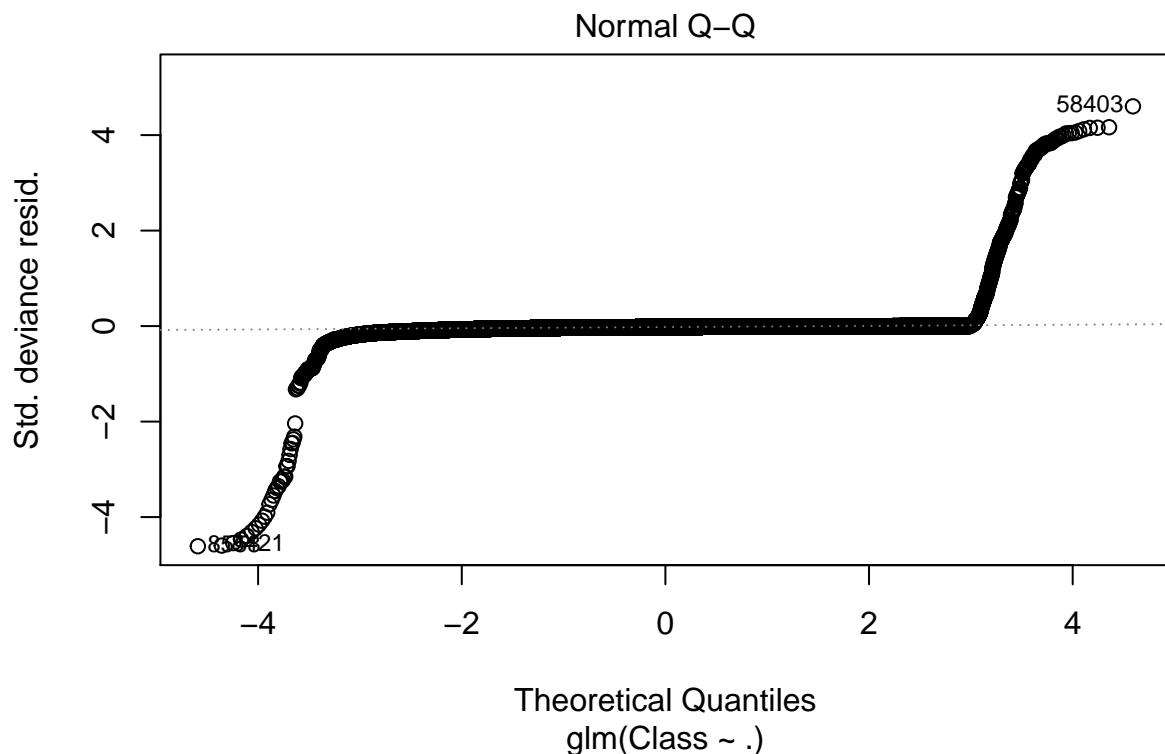
```

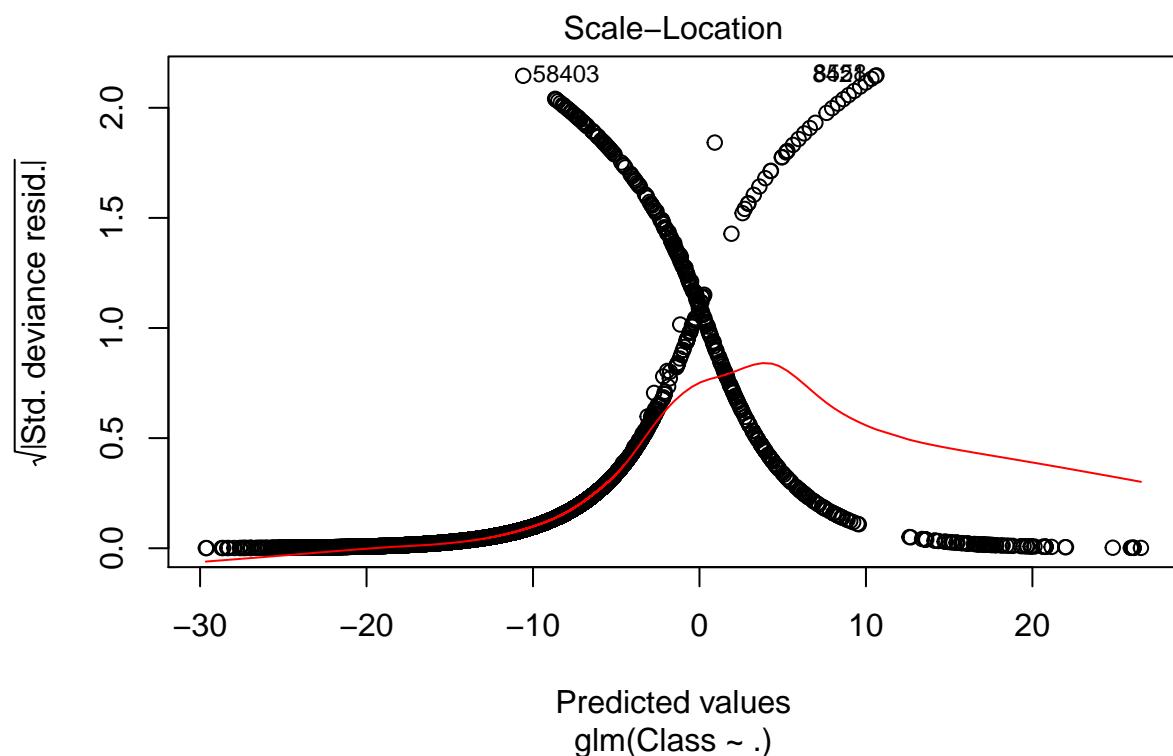
## V16      -0.130225  0.138629 -0.939  0.347537
## V17      0.032463  0.074471  0.436  0.662900
## V18     -0.100964  0.140985 -0.716  0.473909
## V19      0.083711  0.105134  0.796  0.425897
## V20     -0.463946  0.081871 -5.667  1.46e-08 ***
## V21      0.381206  0.065880  5.786 7.19e-09 ***
## V22      0.610874  0.142086  4.299 1.71e-05 ***
## V23     -0.071406  0.058799 -1.214  0.224589
## V24      0.255791  0.170568  1.500  0.133706
## V25     -0.073955  0.142634 -0.519  0.604109
## V26      0.120841  0.202553  0.597  0.550783
## V27     -0.852018  0.118391 -7.197 6.17e-13 ***
## V28     -0.323854  0.090075 -3.595 0.000324 ***
## Amount    0.292477  0.092075  3.177  0.001491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 5799.1 on 227845 degrees of freedom
## Residual deviance: 1790.9 on 227816 degrees of freedom
## AIC: 1850.9
##
## Number of Fisher Scoring iterations: 12

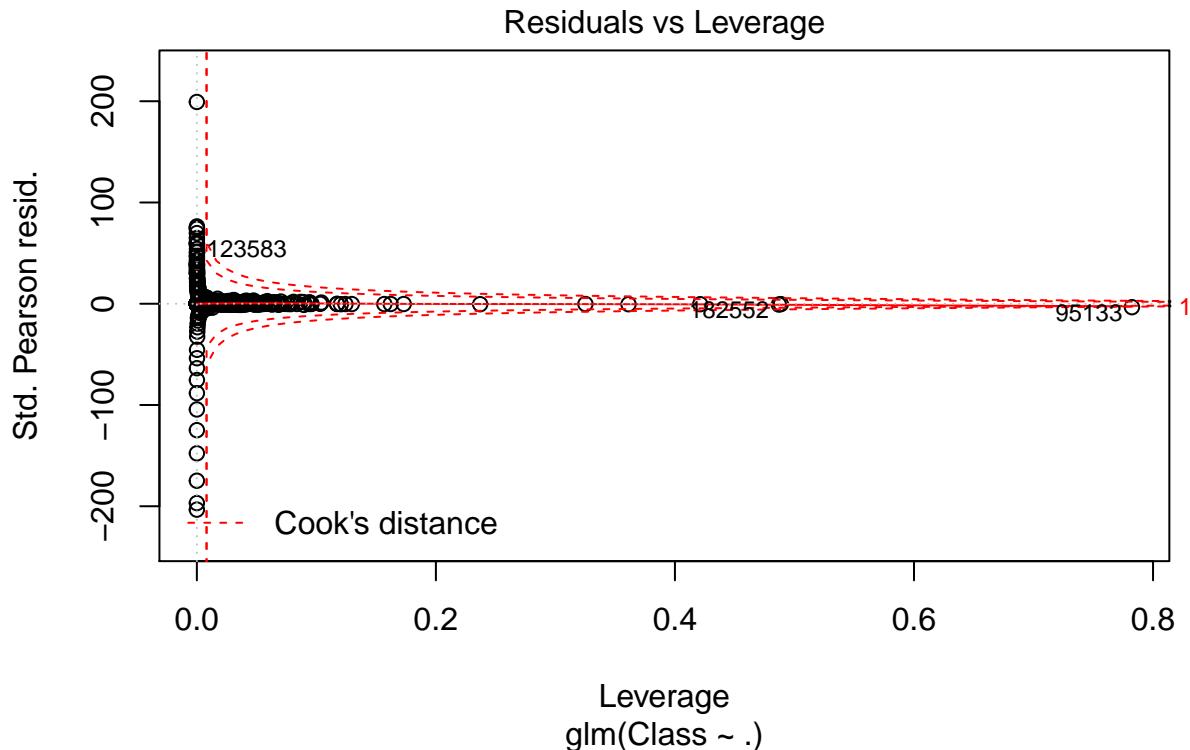
plot(Logistic_Model)

```









Confusion Matrix

Confusion matrix is a very useful tool for calibrating the output of a model and examining all possible outcomes of your predictions (true positive, true negative, false positive, false negative). let us Use a threshold of 0.5 to transform predictions to binary and we will see how the model will fit.

```
Confusion_matrix <- confusionMatrix(table(test_dataset$Class, as.numeric(predict(Logistic_Model, test_d
```

```
print(Confusion_matrix)

## Confusion Matrix and Statistics
##
##          0      1
## 0 56856     7
## 1    41    57
##
##          Accuracy : 0.9992
##                 95% CI : (0.9989, 0.9994)
##    No Information Rate : 0.9989
##    P-Value [Acc > NIR] : 0.02253
##
##          Kappa : 0.7033
##
##  Mcnemar's Test P-Value : 1.906e-06
##
##          Sensitivity : 0.9993
```

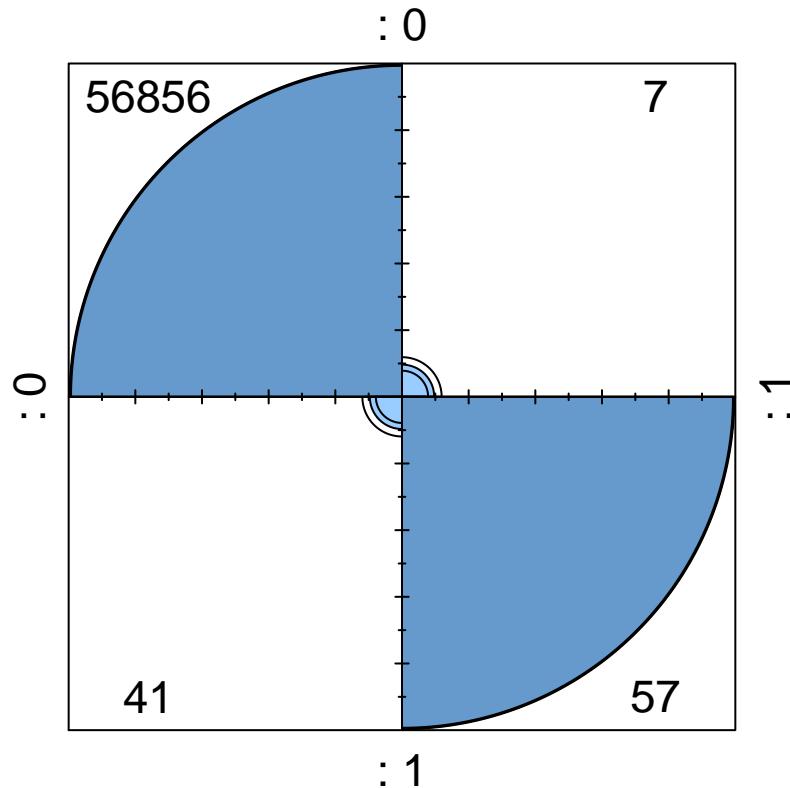
```

##          Specificity : 0.8906
##      Pos Pred Value : 0.9999
##      Neg Pred Value : 0.5816
##          Prevalence : 0.9989
##      Detection Rate : 0.9982
## Detection Prevalence : 0.9983
##      Balanced Accuracy : 0.9450
##
##      'Positive' Class : 0
##

```

A simple logistic regression model achieved nearly 100% accuracy, with ~99% precision (positive predictive value) and ~100% recall (sensitivity). We can see there are only 7 false negatives (transactions which were fraudulent in reality but not identified as such by the model).

```
fourfoldplot(Confusion_matrix$table)
```



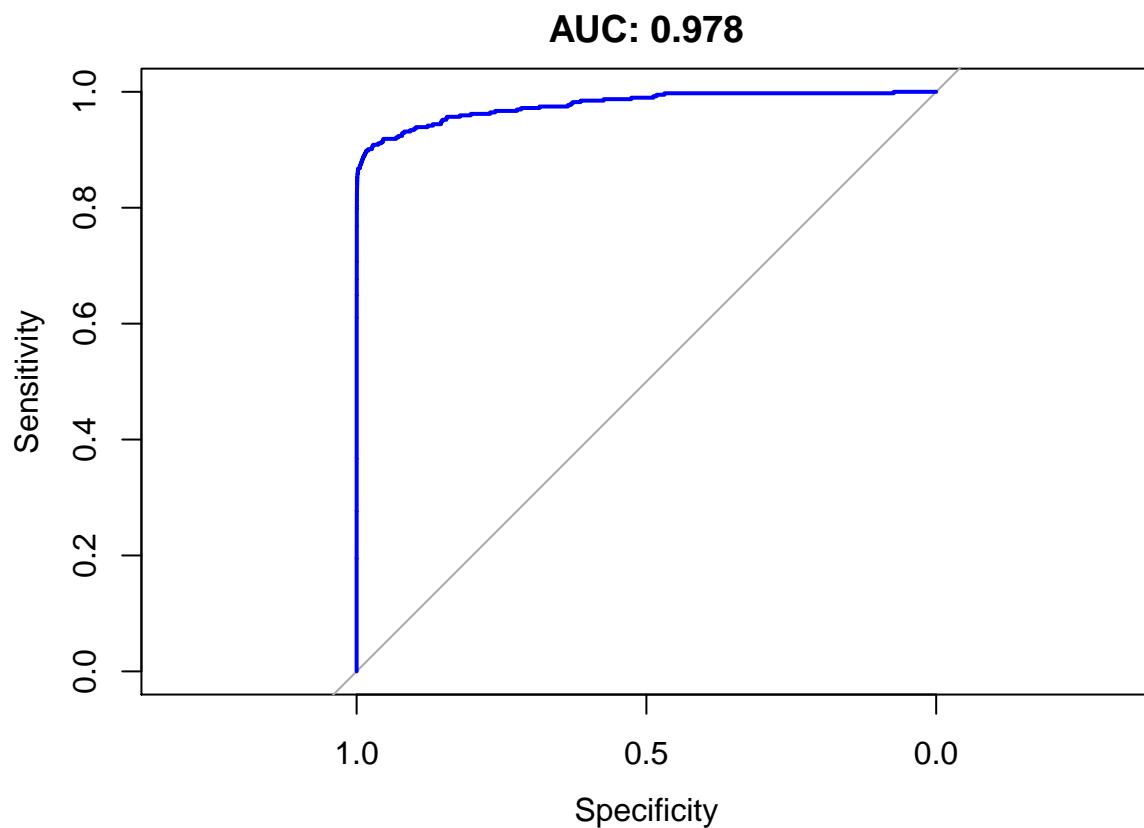
In order to assess the performance of our model, we will describe the ROC curve.

ROC is also known as Receiver Operating Characteristic.

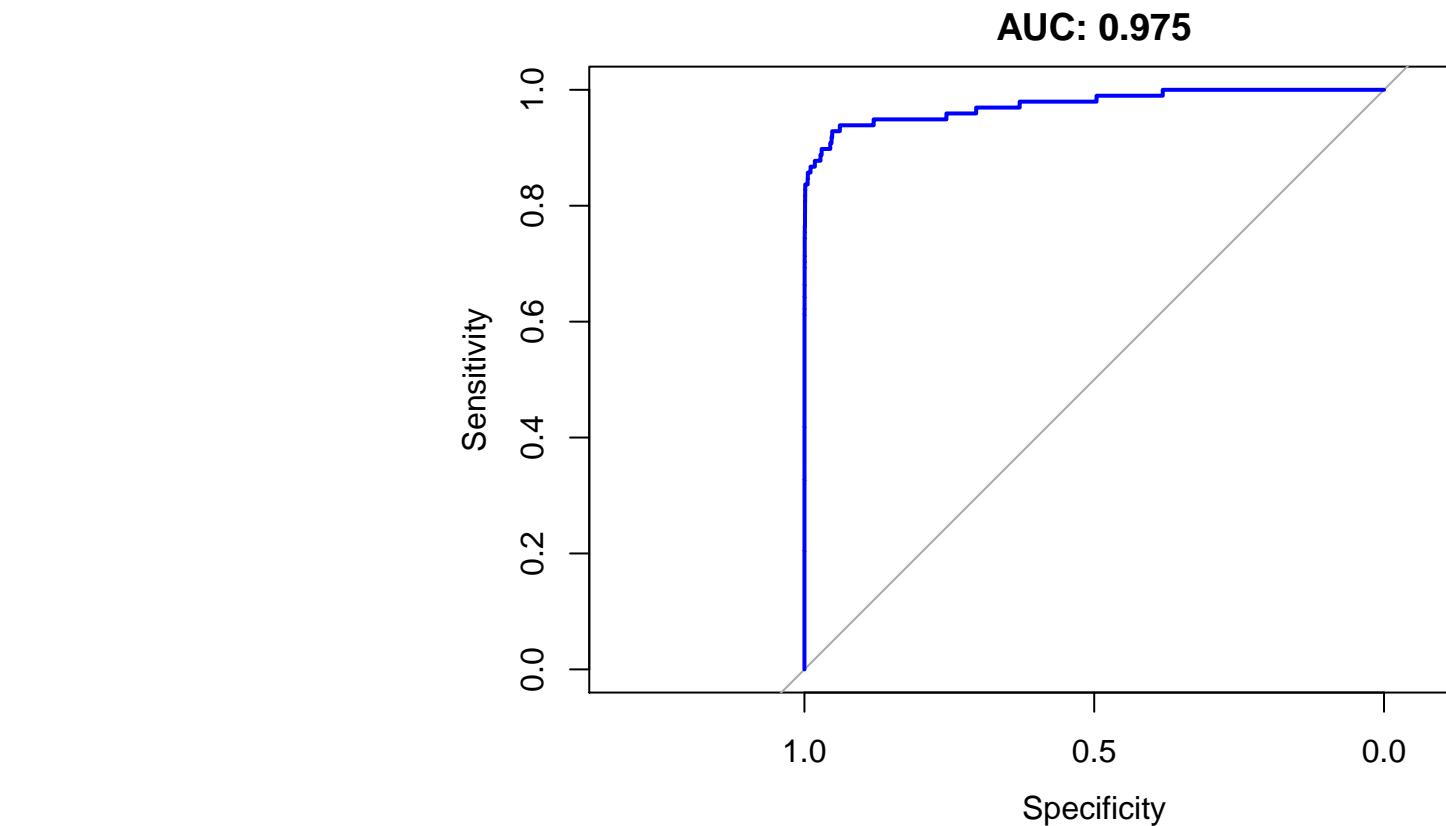
```

lr.predict <- predict(Logistic_Model,train_dataset, probability = TRUE)
auc.gbm <- roc(train_dataset$Class, lr.predict)
plot(auc.gbm, main=paste0("AUC: ", round(pROC::auc(auc.gbm), 3)), col= "blue")

```



```
lr.predict_roc <- predict(Logistic_Model, test_dataset, probability= TRUE)
auc.gbm <- roc(test_dataset$Class, lr.predict_roc)
plot(auc.gbm, main=paste0("AUC: ", round(pROC::auc(auc.gbm), 3)), col = "blue")
```



ROC on unpredicted test data

Fitting a Decision Tree Models : 2

** Note: It will take a little bit time for this Decision Tree model**

We will Implement Decission Tree Algorithm and to plot the out come of decision, these outcomes are basically a consequence through which we can conclude as to what class the object belongs to. We will now implement our decision tree model and will plot it using the rpart.plot() function.

```
decissionTree_model <- rpart(Class~, creditcard_data, method = 'class')
predict_val <- predict(decissionTree_model, creditcard_data, type = 'class')
probability <- predict(decissionTree_model, creditcard_data, type = 'prob')

summary(decissionTree_model)

## Call:
## rpart(formula = Class ~ ., data = creditcard_data, method = "class")
## n= 284807
##
##          CP nsplit rel error      xerror      xstd
## 1 0.47357724      0 1.0000000 1.0000000 0.04504452
## 2 0.05894309      1 0.5264228 0.5365854 0.03300924
## 3 0.05691057      2 0.4674797 0.4959350 0.03173538
## 4 0.02235772      3 0.4105691 0.4247967 0.02937302
## 5 0.02032520      6 0.3434959 0.4065041 0.02873408
## 6 0.01829268      9 0.2804878 0.3760163 0.02763629
```

```

## 7 0.01000000      10 0.2621951 0.3313008 0.02594206
##
## Variable importance
##   V17   V12   V16   V10   V11   V18   V14    V7   V27    V9   V6 Time    V8   V20    V3   V4
##   22    17    13    12    10     9     4     2     1     1     1     1     1     1     1     1     1
##   V21   V26
##   1     1
##
## Node number 1: 284807 observations,      complexity param=0.4735772
##   predicted class=0  expected loss=0.001727486  P(node) =1
##   class counts: 284315   492
##   probabilities: 0.998 0.002
##   left son=2 (284364 obs) right son=3 (443 obs)
## Primary splits:
##   V17 < -2.75394  to the right, improve=514.2412, (0 missing)
##   V12 < -4.562479  to the right, improve=478.1819, (0 missing)
##   V14 < -5.841579  to the right, improve=403.7565, (0 missing)
##   V16 < -4.037668  to the right, improve=361.6741, (0 missing)
##   V10 < -3.833386  to the right, improve=361.0302, (0 missing)
## Surrogate splits:
##   V12 < -4.562479  to the right, agree=0.999, adj=0.628, (0 split)
##   V16 < -4.037668  to the right, agree=0.999, adj=0.578, (0 split)
##   V10 < -5.617552  to the right, agree=0.999, adj=0.427, (0 split)
##   V18 < -3.740074  to the right, agree=0.999, adj=0.402, (0 split)
##   V11 < 4.146304   to the left,  agree=0.999, adj=0.393, (0 split)
##
## Node number 2: 284364 observations,      complexity param=0.05691057
##   predicted class=0  expected loss=0.0005415594  P(node) =0.9984446
##   class counts: 284210   154
##   probabilities: 0.999 0.001
##   left son=4 (284328 obs) right son=5 (36 obs)
## Primary splits:
##   V14 < -8.09768  to the right, improve=56.826780, (0 missing)
##   V12 < -4.839175  to the right, improve=46.246550, (0 missing)
##   V11 < 4.372039   to the left,  improve=23.488920, (0 missing)
##   V10 < -3.84725   to the right, improve=14.658460, (0 missing)
##   V4   < 5.321513   to the left,  improve= 8.063169, (0 missing)
## Surrogate splits:
##   V12 < -5.374364  to the right, agree=1, adj=0.444, (0 split)
##   V11 < 5.426486   to the left,  agree=1, adj=0.278, (0 split)
##
## Node number 3: 443 observations,      complexity param=0.05894309
##   predicted class=1  expected loss=0.2370203  P(node) =0.001555439
##   class counts: 105   338
##   probabilities: 0.237 0.763
##   left son=6 (33 obs) right son=7 (410 obs)
## Primary splits:
##   V12 < -2.18088  to the right, improve=35.18035, (0 missing)
##   V10 < -1.458135  to the right, improve=33.04218, (0 missing)
##   V9   < 3.030988   to the right, improve=29.54316, (0 missing)
##   V7   < 2.484147   to the right, improve=28.99810, (0 missing)
##   V3   < 0.3881728  to the right, improve=24.38649, (0 missing)
## Surrogate splits:
##   V10 < 1.291321   to the right, agree=0.968, adj=0.576, (0 split)

```

```

##      V9 < 3.030988    to the right, agree=0.966, adj=0.545, (0 split)
##      V7 < 3.124222    to the right, agree=0.959, adj=0.455, (0 split)
##      V3 < 0.03134823  to the right, agree=0.957, adj=0.424, (0 split)
##      V4 < -1.983926   to the left,  agree=0.955, adj=0.394, (0 split)
##
## Node number 4: 284328 observations,    complexity param=0.0203252
##   predicted class=0  expected loss=0.0004290819  P(node) =0.9983182
##   class counts: 284206 122
##   probabilities: 1.000 0.000
##   left son=8 (284081 obs) right son=9 (247 obs)
## Primary splits:
##   V14 < -4.661454    to the right, improve=12.8981100, (0 missing)
##   V12 < -4.790606    to the right, improve= 4.5646780, (0 missing)
##   V17 < 5.534993     to the left,  improve= 3.0452870, (0 missing)
##   V10 < -3.213114    to the right, improve= 1.7679740, (0 missing)
##   V11 < 3.503332     to the left,  improve= 0.9499802, (0 missing)
## Surrogate splits:
##   V9 < 7.906346      to the left,  agree=0.999, adj=0.113, (0 split)
##   V18 < 3.197176      to the left,  agree=0.999, adj=0.085, (0 split)
##   V10 < 12.53545     to the left,  agree=0.999, adj=0.040, (0 split)
##   V17 < -2.373553    to the right, agree=0.999, adj=0.040, (0 split)
##   V11 < 4.383243     to the left,  agree=0.999, adj=0.028, (0 split)
##
## Node number 5: 36 observations
##   predicted class=1  expected loss=0.1111111  P(node) =0.0001264014
##   class counts: 4 32
##   probabilities: 0.111 0.889
##
## Node number 6: 33 observations
##   predicted class=0  expected loss=0.06060606  P(node) =0.0001158679
##   class counts: 31 2
##   probabilities: 0.939 0.061
##
## Node number 7: 410 observations,    complexity param=0.02235772
##   predicted class=1  expected loss=0.1804878  P(node) =0.001439571
##   class counts: 74 336
##   probabilities: 0.180 0.820
##   left son=14 (45 obs) right son=15 (365 obs)
## Primary splits:
##   V14 < -3.429828    to the right, improve=15.956910, (0 missing)
##   V26 < -0.2249219   to the left,  improve=15.134070, (0 missing)
##   V4 < 1.169294      to the left,  improve= 9.078085, (0 missing)
##   V9 < -4.031951     to the right, improve= 8.962072, (0 missing)
##   V11 < 0.5930789    to the left,  improve= 7.867805, (0 missing)
## Surrogate splits:
##   V11 < 0.5930789    to the left,  agree=0.910, adj=0.178, (0 split)
##   V7 < 0.1852671     to the right, agree=0.900, adj=0.089, (0 split)
##   V10 < -1.617227    to the right, agree=0.900, adj=0.089, (0 split)
##   V2 < 18.05709      to the right, agree=0.898, adj=0.067, (0 split)
##   V22 < 5.743195     to the right, agree=0.898, adj=0.067, (0 split)
##
## Node number 8: 284081 observations
##   predicted class=0  expected loss=0.0002886501  P(node) =0.9974509
##   class counts: 283999 82

```

```

##      probabilities: 1.000 0.000
##
## Node number 9: 247 observations,    complexity param=0.0203252
##   predicted class=0  expected loss=0.1619433  P(node) =0.000867254
##   class counts:  207    40
##   probabilities: 0.838 0.162
##   left son=18 (195 obs) right son=19 (52 obs)
## Primary splits:
##   V10 < -1.847304  to the right, improve=37.05479, (0 missing)
##   V7  < -0.9118796 to the right, improve=25.64453, (0 missing)
##   V12 < -2.378497  to the right, improve=21.25928, (0 missing)
##   V9  <  0.9459067 to the right, improve=19.76720, (0 missing)
##   V4  <  1.85959   to the left,  improve=18.27138, (0 missing)
## Surrogate splits:
##   V7  < -0.9118796 to the right, agree=0.858, adj=0.327, (0 split)
##   V12 < -2.891134  to the right, agree=0.858, adj=0.327, (0 split)
##   V21 < -0.01407949 to the left,  agree=0.846, adj=0.269, (0 split)
##   V9  < -2.364669  to the right, agree=0.842, adj=0.250, (0 split)
##   V15 < -0.6340641 to the right, agree=0.838, adj=0.231, (0 split)
##
## Node number 14: 45 observations,    complexity param=0.01829268
##   predicted class=0  expected loss=0.4222222  P(node) =0.0001580017
##   class counts:  26    19
##   probabilities: 0.578 0.422
##   left son=28 (34 obs) right son=29 (11 obs)
## Primary splits:
##   V8  < -0.3873742  to the right, improve=6.902080, (0 missing)
##   V20 <  0.3123598  to the left,  improve=5.370302, (0 missing)
##   V4  <  1.400317   to the left,  improve=5.001639, (0 missing)
##   V11 <  1.765276   to the left,  improve=4.595232, (0 missing)
##   V18 < -2.501672   to the right, improve=4.565181, (0 missing)
## Surrogate splits:
##   V21 < -0.6257536  to the right, agree=0.911, adj=0.636, (0 split)
##   V22 <  1.495255   to the left,  agree=0.889, adj=0.545, (0 split)
##   V6  <  0.8007469  to the left,  agree=0.867, adj=0.455, (0 split)
##   V16 < -4.772197   to the right, agree=0.867, adj=0.455, (0 split)
##   V27 < -2.884922   to the right, agree=0.867, adj=0.455, (0 split)
##
## Node number 15: 365 observations,    complexity param=0.02235772
##   predicted class=1  expected loss=0.1315068  P(node) =0.00128157
##   class counts:  48    317
##   probabilities: 0.132 0.868
##   left son=30 (98 obs) right son=31 (267 obs)
## Primary splits:
##   V26 < -0.2644702  to the left,  improve=13.641260, (0 missing)
##   V27 <  1.696946   to the right, improve=11.027410, (0 missing)
##   Time < 31379.5    to the left,  improve=10.829530, (0 missing)
##   V6  < -2.883912   to the left,  improve= 9.930779, (0 missing)
##   V20 <  0.7338922  to the right, improve= 9.524335, (0 missing)
## Surrogate splits:
##   V9  < -0.1000254  to the right, agree=0.753, adj=0.082, (0 split)
##   V4  <  0.3721625  to the left,  agree=0.748, adj=0.061, (0 split)
##   V27 < -4.25345   to the left,  agree=0.748, adj=0.061, (0 split)
##   V7  < -28.1132    to the left,  agree=0.742, adj=0.041, (0 split)

```

```

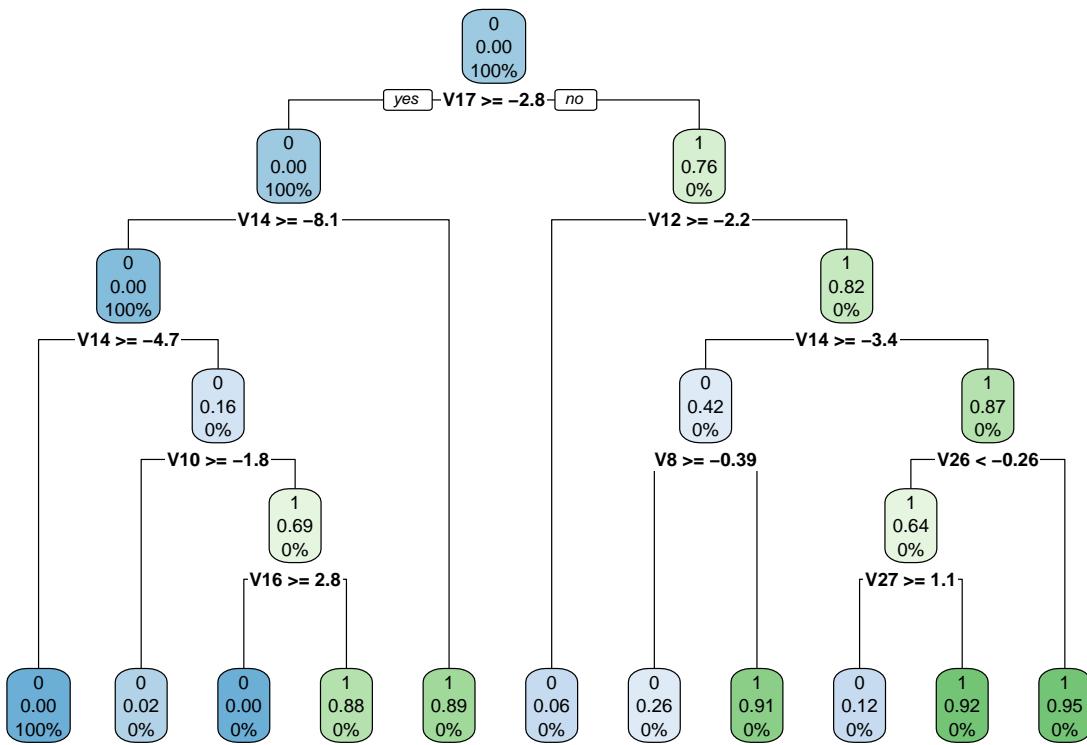
##      V10 < -19.93122  to the left,  agree=0.742, adj=0.041, (0 split)
##
## Node number 18: 195 observations
##   predicted class=0  expected loss=0.02051282  P(node) =0.0006846742
##   class counts:  191     4
##   probabilities: 0.979 0.021
##
## Node number 19: 52 observations,    complexity param=0.0203252
##   predicted class=1  expected loss=0.3076923  P(node) =0.0001825798
##   class counts:  16     36
##   probabilities: 0.308 0.692
##   left son=38 (11 obs) right son=39 (41 obs)
## Primary splits:
##   V16 < 2.754354  to the right, improve=13.373360, (0 missing)
##   V18 < 3.081813  to the right, improve=10.218710, (0 missing)
##   V24 < -0.662693  to the left,  improve= 9.461036, (0 missing)
##   V5  < 1.271888  to the right, improve= 7.809019, (0 missing)
##   V19 < -2.271497  to the left,  improve= 7.095023, (0 missing)
## Surrogate splits:
##   V18  < 3.355946  to the right, agree=0.942, adj=0.727, (0 split)
##   V2   < 3.136648  to the right, agree=0.904, adj=0.545, (0 split)
##   V7   < 1.7515    to the right, agree=0.904, adj=0.545, (0 split)
##   V19 < -2.271497  to the left,  agree=0.865, adj=0.364, (0 split)
##   Time < 161088   to the right, agree=0.846, adj=0.273, (0 split)
##
## Node number 28: 34 observations
##   predicted class=0  expected loss=0.2647059  P(node) =0.0001193791
##   class counts:  25     9
##   probabilities: 0.735 0.265
##
## Node number 29: 11 observations
##   predicted class=1  expected loss=0.09090909  P(node) =3.862265e-05
##   class counts:  1     10
##   probabilities: 0.091 0.909
##
## Node number 30: 98 observations,    complexity param=0.02235772
##   predicted class=1  expected loss=0.3571429  P(node) =0.0003440927
##   class counts:  35     63
##   probabilities: 0.357 0.643
##   left son=60 (34 obs) right son=61 (64 obs)
## Primary splits:
##   V27 < 1.077147  to the right, improve=28.72243, (0 missing)
##   Time < 33013    to the left,  improve=25.48485, (0 missing)
##   V26 < -0.3338474 to the right, improve=23.55094, (0 missing)
##   V6  < -2.446679  to the left,  improve=21.87879, (0 missing)
##   V20 < 0.7582707 to the right, improve=16.52107, (0 missing)
## Surrogate splits:
##   V6  < -2.867273  to the left,  agree=0.939, adj=0.824, (0 split)
##   Time < 33013    to the left,  agree=0.918, adj=0.765, (0 split)
##   V8  < 1.754752  to the right, agree=0.867, adj=0.618, (0 split)
##   V11 < 7.243801  to the right, agree=0.867, adj=0.618, (0 split)
##   V20 < 0.7582707 to the right, agree=0.867, adj=0.618, (0 split)
##
## Node number 31: 267 observations

```

```
##   predicted class=1  expected loss=0.04868914  P(node) =0.000937477
##   class counts:     13    254
##   probabilities:  0.049  0.951
##
## Node number 38: 11 observations
##   predicted class=0  expected loss=0  P(node) =3.862265e-05
##   class counts:     11      0
##   probabilities:  1.000  0.000
##
## Node number 39: 41 observations
##   predicted class=1  expected loss=0.1219512  P(node) =0.0001439571
##   class counts:      5    36
##   probabilities:  0.122  0.878
##
## Node number 60: 34 observations
##   predicted class=0  expected loss=0.1176471  P(node) =0.0001193791
##   class counts:     30      4
##   probabilities:  0.882  0.118
##
## Node number 61: 64 observations
##   predicted class=1  expected loss=0.078125  P(node) =0.0002247136
##   class counts:      5    59
##   probabilities:  0.078  0.922
```

Plot

```
rpart.plot(decissionTree_model)
```



K-Fold with Cross-Validation Model:3

The K-fold cross validation is go-to-method for evaluating the performance of an algorithm on dataset. In terms of how to select k for cross validation, larger values of k are preferable but they will also take much more computational time. For this reason, the choices of k=5 and k=10 are common. we are using SMOTE resampling the data, with 5-fold cv and trains Random forest classifier with roc as a metric.

K-fold cross Validation

```
CV_control <- trainControl(method = "cv",
                             number = 5,
                             verboseIter = T,
                             classProbs = T,
                             sampling = "smote",
                             summaryFunction = twoClassSummary,
                             savePredictions = T)

train_dataset_CV <- train_dataset
train_dataset_CV$Class <- as.factor(train_dataset_CV$Class)
levels(train_dataset_CV$Class) <- make.names(c(0,1))
model_CV <- train(Class~., data = train_dataset_CV, method = "rpart", trControl = CV_control,
                   metric = 'ROC')

## + Fold1: cp=0.05838
## - Fold1: cp=0.05838
```

```

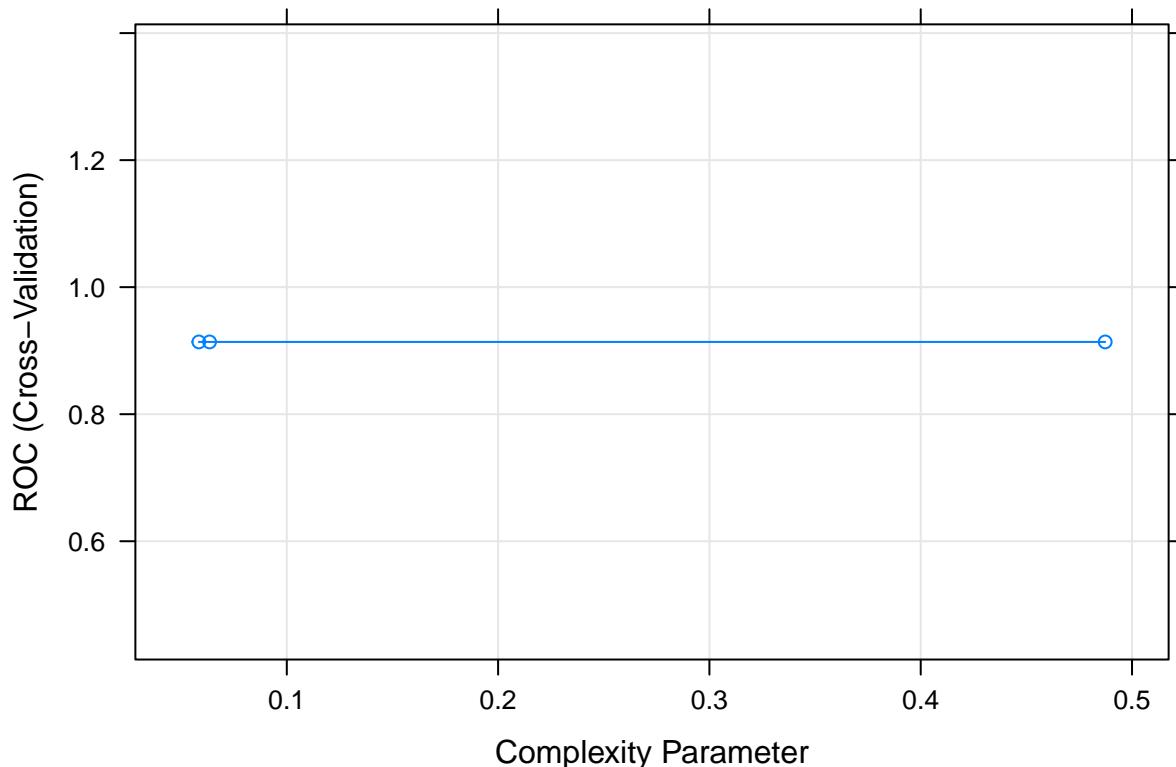
## + Fold2: cp=0.05838
## - Fold2: cp=0.05838
## + Fold3: cp=0.05838
## - Fold3: cp=0.05838
## + Fold4: cp=0.05838
## - Fold4: cp=0.05838
## + Fold5: cp=0.05838
## - Fold5: cp=0.05838
## Aggregating results
## Selecting tuning parameters
## Fitting cp = 0.487 on full training set

print(model_CV)

## CART
##
## 227846 samples
##      29 predictor
##      2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 182277, 182278, 182276, 182276, 182277
## Additional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##     cp          ROC      Sens      Spec
## 0.05837563  0.9137763  0.9875006  0.8400519
## 0.06345178  0.9137763  0.9875006  0.8400519
## 0.48730964  0.9137763  0.9875006  0.8400519
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.4873096.

```

```
plot(model_CV)
```



Now let see the model performing on unpredicted Test data.

Cross Validation does give an unbiased estimation of the algorithms performance on unseen dataset.

```
Model_CV_predict <- predict(model_CV, test_dataset, type = "prob")
Confussion_matrix_CV <- confusionMatrix(table(as.numeric(Model_CV_predict$X1 > 0.5), test_dataset$Class))
print(Confussion_matrix_CV)

## Confusion Matrix and Statistics
##
##
##          0      1
## 0  56286    17
## 1    577    81
##
##          Accuracy : 0.9896
##                 95% CI : (0.9887, 0.9904)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##          Kappa : 0.2119
##
##  Mcnemar's Test P-Value : <2e-16
##
##          Sensitivity : 0.9899
##          Specificity : 0.8265
```

```

##           Pos Pred Value : 0.9997
##           Neg Pred Value : 0.1231
##           Prevalence : 0.9983
##           Detection Rate : 0.9881
##   Detection Prevalence : 0.9884
##           Balanced Accuracy : 0.9082
##
##           'Positive' Class : 0
##

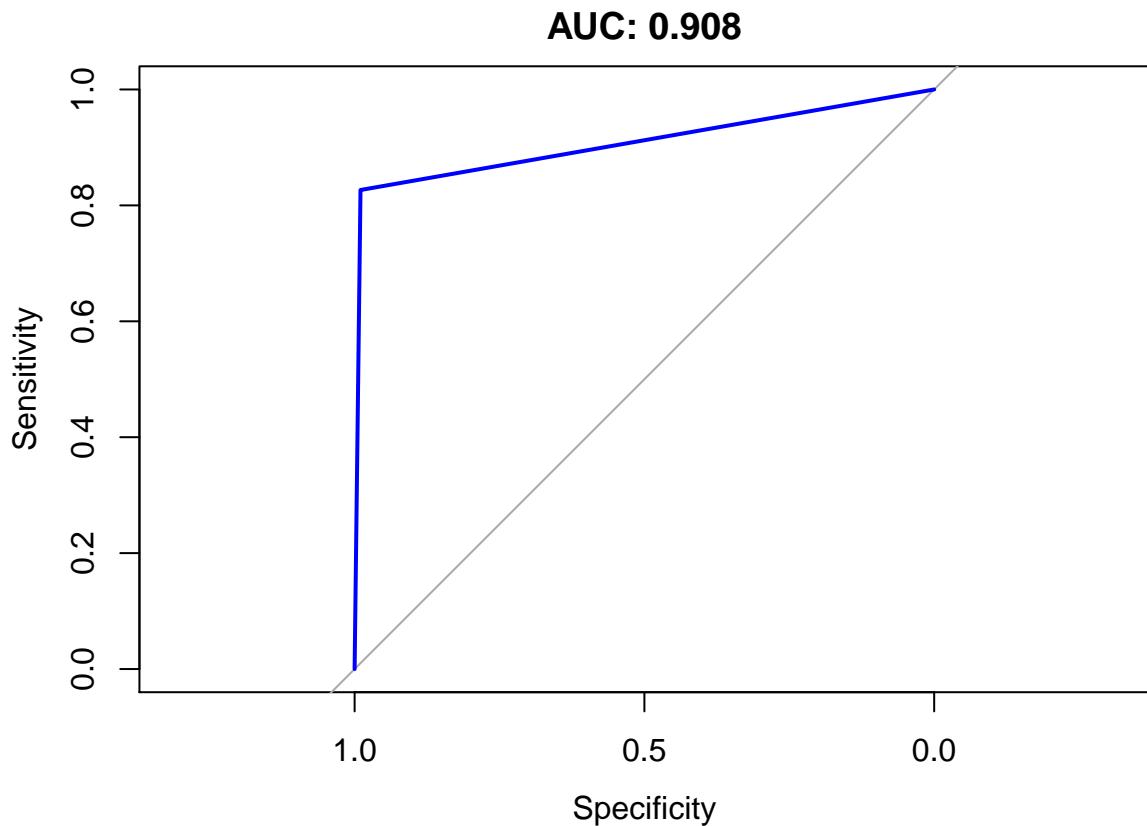
** ROC on unpredicted test data**

model_CV_roc <- roc(test_dataset$Class, predict(model_CV, test_dataset, type = "prob")$X1)
print(model_CV_roc)

##
## Call:
## roc.default(response = test_dataset$Class, predictor = predict(model_CV,      test_dataset, type = "prob")
## 
## Data: predict(model_CV, test_dataset, type = "prob")$X1 in 56863 controls (test_dataset$Class 0) < 9983
## Area under the curve: 0.9082

plot(model_CV_roc, main = paste0("AUC: ", round(pROC::auc(model_CV_roc), 3)), col = "blue")

```



Repeated k-fold Cross Validation Method: 2 The process of splitting the data into k-folds can be repeated a number of times, this is called Repeated k-fold Cross Validation. The final model accuracy is taken as the mean from the number of repeats.

```

CV_repeat <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 3,
                           verboseIter = T,
                           classProbs = T,
                           sampling = "smote",
                           summaryFunction = twoClassSummary,
                           savePredictions = T)
model_repet <- train(Class~., data = train_dataset_CV, trControl = CV_repeat, method = 'rpart2', metric = "Accuracy")

## + Fold1.Rep1: maxdepth=3
## - Fold1.Rep1: maxdepth=3
## + Fold2.Rep1: maxdepth=3
## - Fold2.Rep1: maxdepth=3
## + Fold3.Rep1: maxdepth=3
## - Fold3.Rep1: maxdepth=3
## + Fold4.Rep1: maxdepth=3
## - Fold4.Rep1: maxdepth=3
## + Fold5.Rep1: maxdepth=3
## - Fold5.Rep1: maxdepth=3
## + Fold1.Rep2: maxdepth=3
## - Fold1.Rep2: maxdepth=3
## + Fold2.Rep2: maxdepth=3
## - Fold2.Rep2: maxdepth=3
## + Fold3.Rep2: maxdepth=3
## - Fold3.Rep2: maxdepth=3
## + Fold4.Rep2: maxdepth=3
## - Fold4.Rep2: maxdepth=3
## + Fold5.Rep2: maxdepth=3
## - Fold5.Rep2: maxdepth=3
## + Fold1.Rep3: maxdepth=3
## - Fold1.Rep3: maxdepth=3
## + Fold2.Rep3: maxdepth=3
## - Fold2.Rep3: maxdepth=3
## + Fold3.Rep3: maxdepth=3
## - Fold3.Rep3: maxdepth=3
## + Fold4.Rep3: maxdepth=3
## - Fold4.Rep3: maxdepth=3
## + Fold5.Rep3: maxdepth=3
## - Fold5.Rep3: maxdepth=3
## Aggregating results
## Selecting tuning parameters
## Fitting maxdepth = 2 on full training set

print(model_repet)

## CART
##
## 227846 samples

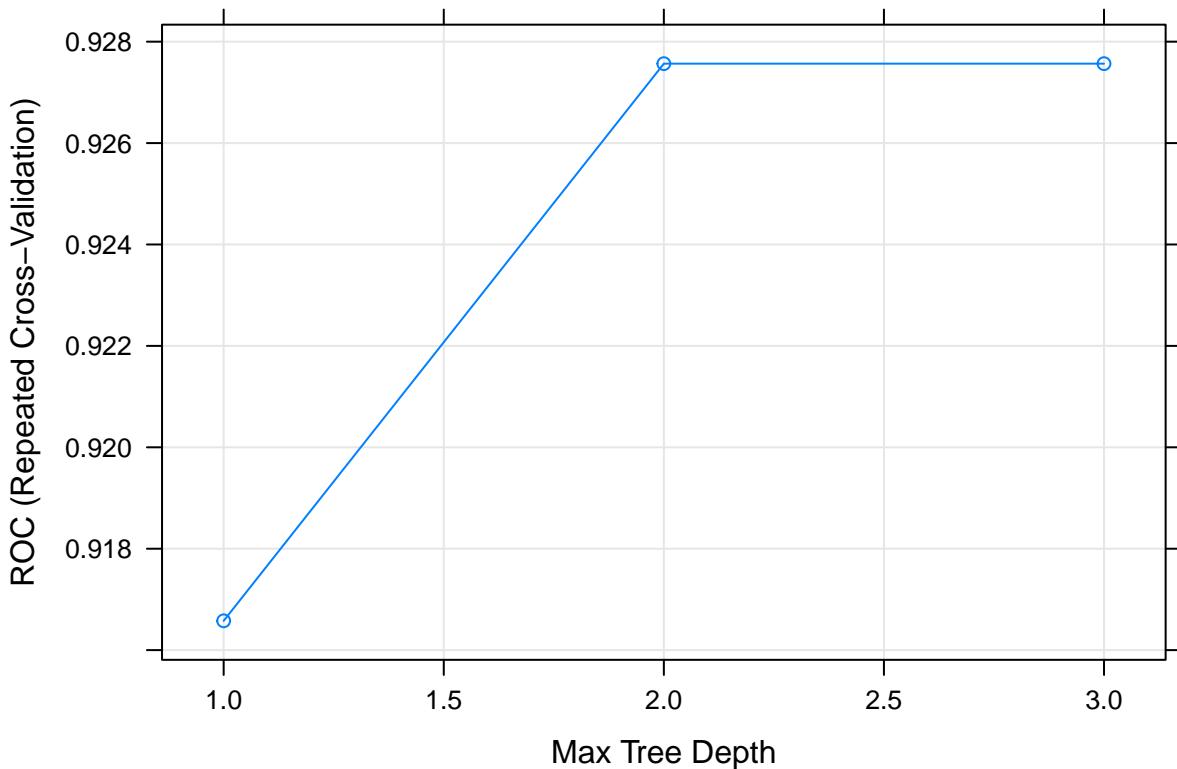
```

```

##      29 predictor
##      2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 182278, 182277, 182276, 182277, 182276, ...
## Additional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   maxdepth    ROC      Sens      Spec
##   1           0.9165777 0.9820680 0.8510873
##   2           0.9275663 0.9724294 0.8629233
##   3           0.9275663 0.9724294 0.8629233
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 2.

```

```
plot(model_repet)
```



Now let see the model performing on unpredicted Test data.

```

CV_repeat_pred <- predict(model_repet, test_dataset, type = 'prob')
confussion_Matrix_CVRepet <- confusionMatrix(table(as.numeric(CV_repeat_pred$X1 >0.5),test_dataset$Class))
print(confussion_Matrix_CVRepet)

```

```

## Confusion Matrix and Statistics
##
##
##          0      1
## 0 56158    17
## 1    705    81
##
##          Accuracy : 0.9873
##                 95% CI : (0.9864, 0.9882)
## No Information Rate : 0.9983
## P-Value [Acc > NIR] : 1
##
##          Kappa : 0.1808
##
## McNemar's Test P-Value : <2e-16
##
##          Sensitivity : 0.9876
##          Specificity : 0.8265
## Pos Pred Value : 0.9997
## Neg Pred Value : 0.1031
##          Prevalence : 0.9983
##          Detection Rate : 0.9859
## Detection Prevalence : 0.9862
## Balanced Accuracy : 0.9071
##
## 'Positive' Class : 0
##

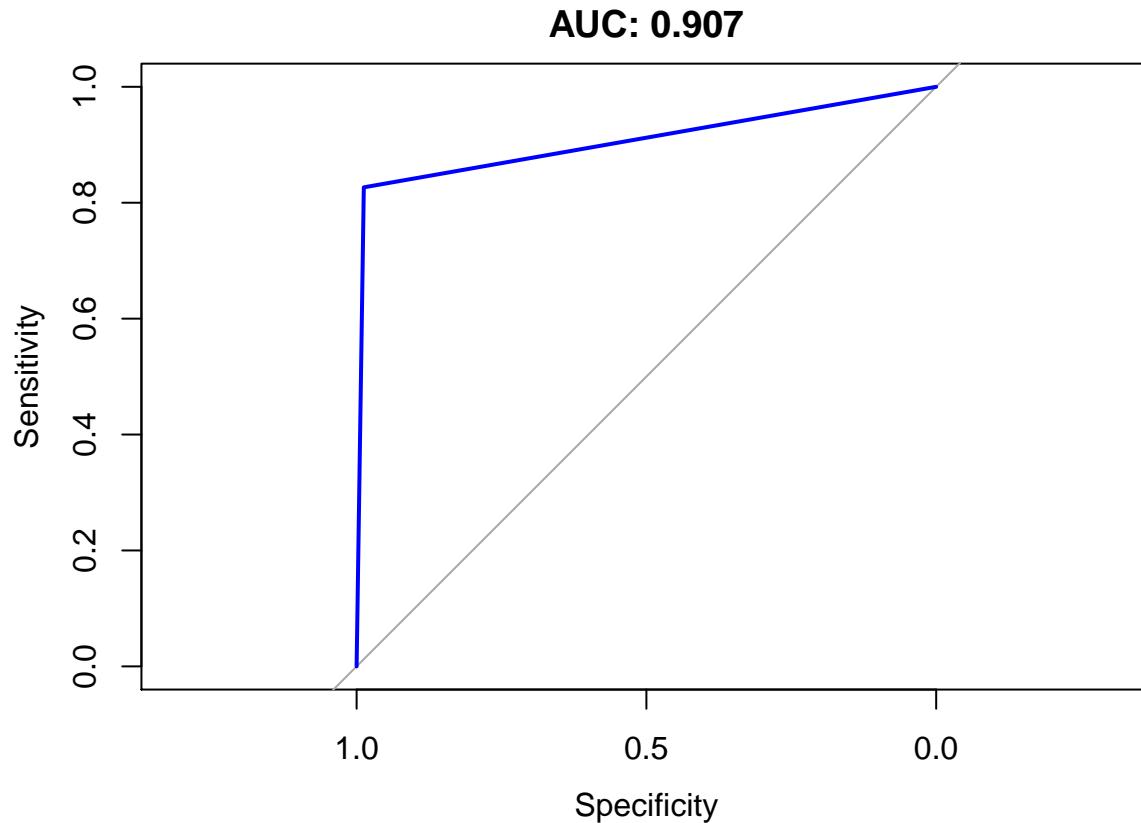
** ROC on unpredicted test data **

CV_repet_roc <- roc(test_dataset$Class, predict(model_repet, test_dataset, type = "prob")$X1)
print(CV_repet_roc)

##
## Call:
## roc.default(response = test_dataset$Class, predictor = predict(model_repet, test_dataset, type =
## 
## Data: predict(model_repet, test_dataset, type = "prob")$X1 in 56863 controls (test_dataset$Class 0) ...
## Area under the curve: 0.9071

plot(CV_repet_roc, main = paste0("AUC: ", round(pROC::auc(CV_repet_roc), 3)), col ="blue")

```



Random forest Model: 4

the code below uses the SMOTE to resample the data with 5 fold cv perfomance and trains a Random Forest Classifier using ROC as a metric.

```
Model_rf <- trainControl(method = "cv",
                           number = 5,
                           verboseIter = T,
                           classProbs = T,
                           sampling = "smote",
                           summaryFunction = twoClassSummary,
                           savePredictions = T)

train_dataset_rf <- train_dataset
train_dataset_rf$Class <- as.factor(train_dataset_rf$Class)
levels(train_dataset_rf$Class) <- make.names(c(0,1))
Model_rf <- train(Class~, data = train_dataset_rf, method = 'rf', trControl = Model_rf, metric = 'ROC')

## + Fold1: mtry= 2
## - Fold1: mtry= 2
## + Fold1: mtry=15
## - Fold1: mtry=15
## + Fold1: mtry=29
## - Fold1: mtry=29
## + Fold2: mtry= 2
```

```

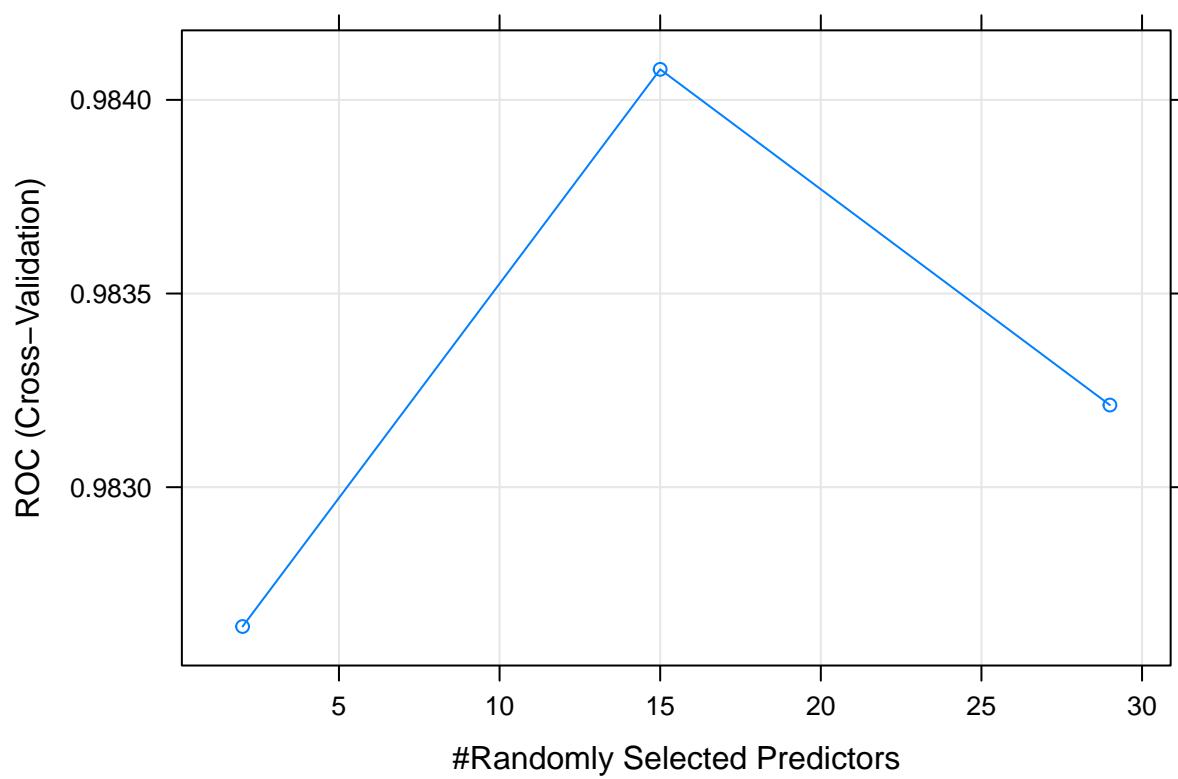
## - Fold2: mtry= 2
## + Fold2: mtry=15
## - Fold2: mtry=15
## + Fold2: mtry=29
## - Fold2: mtry=29
## + Fold3: mtry= 2
## - Fold3: mtry= 2
## + Fold3: mtry=15
## - Fold3: mtry=15
## + Fold3: mtry=29
## - Fold3: mtry=29
## + Fold4: mtry= 2
## - Fold4: mtry= 2
## + Fold4: mtry=15
## - Fold4: mtry=15
## + Fold4: mtry=29
## - Fold4: mtry=29
## + Fold5: mtry= 2
## - Fold5: mtry= 2
## + Fold5: mtry=15
## - Fold5: mtry=15
## + Fold5: mtry=29
## - Fold5: mtry=29
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 15 on full training set

print(Model_rf)

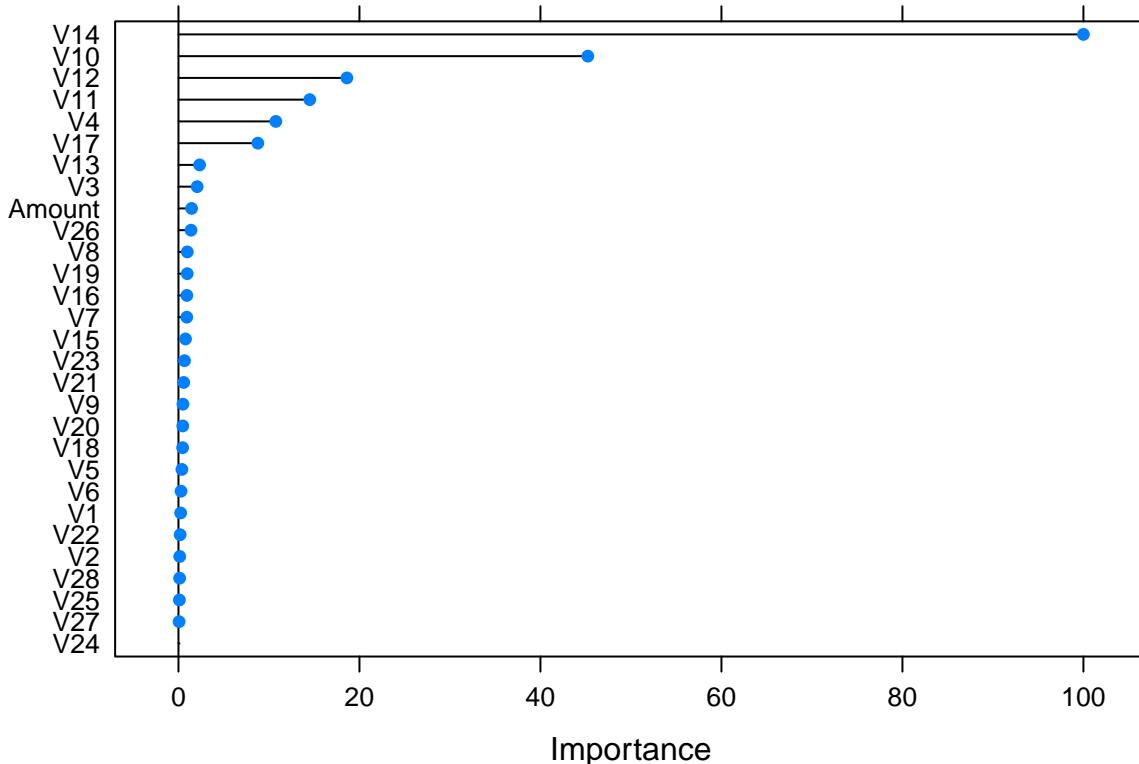
## Random Forest
##
## 227846 samples
##      29 predictor
##      2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 182276, 182277, 182277, 182277, 182277
## Additional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   mtry    ROC      Sens      Spec
##   2       0.9826403 0.9950275 0.8655631
##   15      0.9840788 0.9888328 0.8858812
##   29      0.9832121 0.9831217 0.8757546
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 15.

plot(Model_rf)

```



```
plot(varImp(Model_rf))
```



** Now let see the model performing on unpredicted Test data.**

```

rf_pred <- predict(Model_rf, test_dataset, type = "prob")
Confusion_matrix_rf <- confusionMatrix(table(as.numeric(rf_pred$X1 > 0.5), test_dataset$Class))
print(Confusion_matrix_rf)

## Confusion Matrix and Statistics
##
##
##          0     1
## 0 56504    12
## 1   359    86
##
##                  Accuracy : 0.9935
##                         95% CI : (0.9928, 0.9941)
##      No Information Rate : 0.9983
##      P-Value [Acc > NIR] : 1
##
##                  Kappa : 0.3148
##
##      Mcnemar's Test P-Value : <2e-16
##
##                  Sensitivity : 0.9937
##                  Specificity  : 0.8776
##      Pos Pred Value : 0.9998
##      Neg Pred Value : 0.1933

```

```

##           Prevalence : 0.9983
##           Detection Rate : 0.9920
##   Detection Prevalence : 0.9922
##           Balanced Accuracy : 0.9356
##
##           'Positive' Class : 0
##
```

** Now we will look ROC on unpredicted test data.**

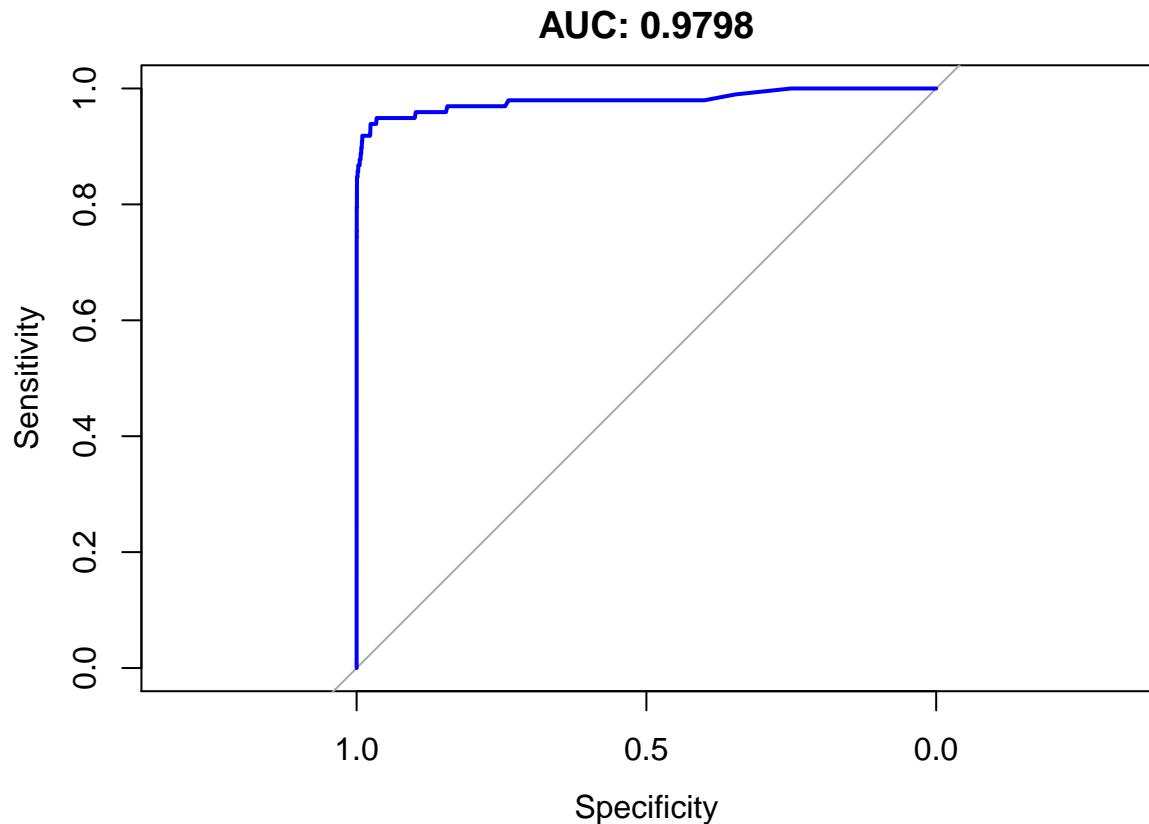
```

rf_roc <- roc(test_dataset$Class, predict(Model_rf, test_dataset, type = "prob")$X1)
print(rf_roc)
```

```

##
## Call:
## roc.default(response = test_dataset$Class, predictor = predict(Model_rf,      test_dataset, type = "p
## 
## Data: predict(Model_rf, test_dataset, type = "prob")$X1 in 56863 controls (test_dataset$Class 0) < 9
## Area under the curve: 0.9798
```

```
plot(rf_roc, main = paste0("AUC: ", round(pROC::auc(rf_roc), 4)), col ="blue")
```



Model XG Boost: Gradient Boosted Tress: 5

Finally we will use XGboost which is based on Gradient Boosted Tress, which is a most powerful model when compared to the other models discussed above.

```

train_XGB <- xgb.DMatrix(data = as.matrix(train_dataset[, -c("Class")]), label = as.numeric(train_dataset$Class))
Model_xgb <- xgboost(data = train_XGB, nrounds = 100, gamma = 0.1, max_depth = 10, objective = "binary:logistic")

## [1]  train-error:0.000391
## [2]  train-error:0.000356
## [3]  train-error:0.000342
## [4]  train-error:0.000320
## [5]  train-error:0.000285
## [6]  train-error:0.000281
## [7]  train-error:0.000281
## [8]  train-error:0.000272
## [9]  train-error:0.000263
## [10] train-error:0.000263
## [11] train-error:0.000263
## [12] train-error:0.000259
## [13] train-error:0.000259
## [14] train-error:0.000259
## [15] train-error:0.000259
## [16] train-error:0.000250
## [17] train-error:0.000250
## [18] train-error:0.000246
## [19] train-error:0.000233
## [20] train-error:0.000228
## [21] train-error:0.000224
## [22] train-error:0.000202
## [23] train-error:0.000189
## [24] train-error:0.000180
## [25] train-error:0.000162
## [26] train-error:0.000158
## [27] train-error:0.000140
## [28] train-error:0.000132
## [29] train-error:0.000127
## [30] train-error:0.000114
## [31] train-error:0.000105
## [32] train-error:0.000097
## [33] train-error:0.000088
## [34] train-error:0.000075
## [35] train-error:0.000057
## [36] train-error:0.000044
## [37] train-error:0.000040
## [38] train-error:0.000026
## [39] train-error:0.000022
## [40] train-error:0.000022
## [41] train-error:0.000018
## [42] train-error:0.000013
## [43] train-error:0.000004
## [44] train-error:0.000004
## [45] train-error:0.000000
## [46] train-error:0.000000
## [47] train-error:0.000000
## [48] train-error:0.000000
## [49] train-error:0.000000
## [50] train-error:0.000000

```

```
## [51] train-error:0.000000
## [52] train-error:0.000000
## [53] train-error:0.000000
## [54] train-error:0.000000
## [55] train-error:0.000000
## [56] train-error:0.000000
## [57] train-error:0.000000
## [58] train-error:0.000000
## [59] train-error:0.000000
## [60] train-error:0.000000
## [61] train-error:0.000000
## [62] train-error:0.000000
## [63] train-error:0.000000
## [64] train-error:0.000000
## [65] train-error:0.000000
## [66] train-error:0.000000
## [67] train-error:0.000000
## [68] train-error:0.000000
## [69] train-error:0.000000
## [70] train-error:0.000000
## [71] train-error:0.000000
## [72] train-error:0.000000
## [73] train-error:0.000000
## [74] train-error:0.000000
## [75] train-error:0.000000
## [76] train-error:0.000000
## [77] train-error:0.000000
## [78] train-error:0.000000
## [79] train-error:0.000000
## [80] train-error:0.000000
## [81] train-error:0.000000
## [82] train-error:0.000000
## [83] train-error:0.000000
## [84] train-error:0.000000
## [85] train-error:0.000000
## [86] train-error:0.000000
## [87] train-error:0.000000
## [88] train-error:0.000000
## [89] train-error:0.000000
## [90] train-error:0.000000
## [91] train-error:0.000000
## [92] train-error:0.000000
## [93] train-error:0.000000
## [94] train-error:0.000000
## [95] train-error:0.000000
## [96] train-error:0.000000
## [97] train-error:0.000000
## [98] train-error:0.000000
## [99] train-error:0.000000
## [100]    train-error:0.000000
```

** Let's see into unseen test test**

```

test_XBG <- xgb.DMatrix(data = as.matrix(test_dataset[, -c("Class")]), label = as.numeric(test_dataset$Class))
predit_xgb <- predict(Model_xgb,test_XBG)
confusionMatrix(table(as.numeric(predit_xgb > 0.5), test_dataset$Class))

## Confusion Matrix and Statistics
##
##
##          0      1
## 0 56857    22
## 1      6    76
##
##          Accuracy : 0.9995
##                 95% CI : (0.9993, 0.9997)
## No Information Rate : 0.9983
## P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.8442
##
## McNemar's Test P-Value : 0.004586
##
##          Sensitivity : 0.9999
##          Specificity : 0.7755
## Pos Pred Value : 0.9996
## Neg Pred Value : 0.9268
## Prevalence : 0.9983
## Detection Rate : 0.9982
## Detection Prevalence : 0.9986
## Balanced Accuracy : 0.8877
##
## 'Positive' Class : 0
## 

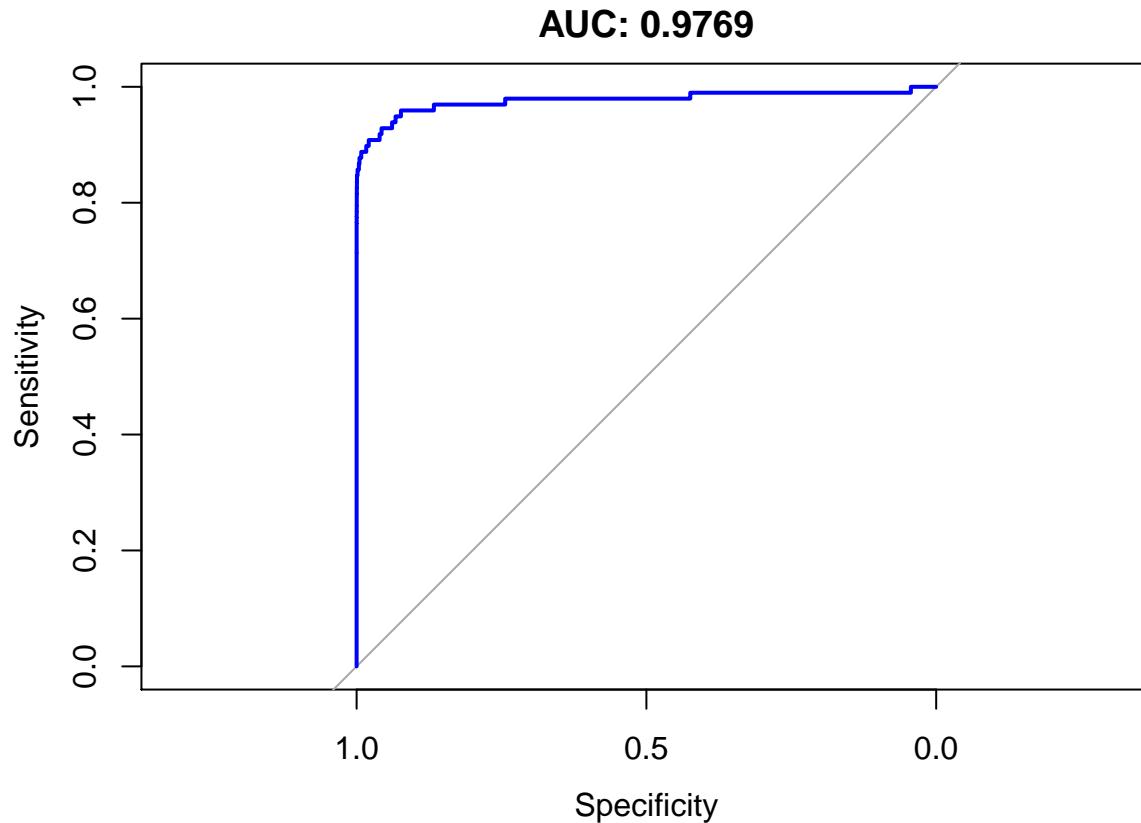
** Let's see ROC for XGBoost**

roc_xgb <- roc(test_dataset$Class, predit_xgb)
print(roc_xgb)

##
## Call:
## roc.default(response = test_dataset$Class, predictor = predit_xgb)
##
## Data: predit_xgb in 56863 controls (test_dataset$Class 0) < 98 cases (test_dataset$Class 1).
## Area under the curve: 0.9769

plot(roc_xgb, main = paste0("AUC: ", round(pROC::auc(roc_xgb), 4)), col = "blue")

```



Summary:

In the credit Card Fraudulent detection project we have applied some of common and popular Machine Learning Models / algorithms to detect fraudulent.

Based on over exploration it has shown that even a very simple logistic regression model can achieve good recall. while the other models improve in term of Logistic Regression in terms of AUC.

As we see the best model: XGBoost (0.9769), Random Forest (0.9798) with a very marginal difference when comparative with Logistic Regression in terms of AUC.

Appendix - Environment

```
version
## 
## platform      x86_64-w64-mingw32
## arch         x86_64
## os           mingw32
## system       x86_64, mingw32
## status        
## major        3
```

```
## minor      6.3
## year       2020
## month      02
## day        29
## svn rev    77875
## language   R
## version.string R version 3.6.3 (2020-02-29)
## nickname   Holding the Windsock
```