APPENDIX 1

Credit Card Fraud Detection with Machine Learning

A Project Report of Capstone Project - 2

Submitted by

VISHAL THAKUR

(1613101842/16SCSE101251)

in partial fulfillment for the award of the degree

of

Bachelor of Technology

IN

Computer Science and Engineering

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of

Dr. Kuldeep Singh Kaswan

Professor

APRIL / MAY- 2020

APPENDIX 2

SCHOOL OF COMPUTING AND SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report "**Credit Card Fraud Detection with Machine Learning**" is the bonafide work of "**VISHAL THAKUR (1613101842**)" who carried out the project work under my supervision.

SIGNATURE OF HEAD

Dr. MUNISH SHABARWAL,

Professor & Dean,

School of Computing Science & Engineering

SIGNATURE OF SUPERVISOR

Dr. KULDEEP SINGH KASWAN

Professor,

School of Computing Science & Engineering

APPENDIX 3

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1.	**Abstract**	**5**
2.	**Introduction**	**6**
2.1	Overall Description	6
3	**Methodology Adopted**	**7**
3.1	Importing The Dataset	7
3.2	Data Exploration	7-10
3.3	Data Manipulation	10-12
3.4	Data Modeling	12-13
3.5	Fitting Logistic Regression Model	13-18
3.6	Fitting a Decision Tree Model	18-19
3.7	Artificial Neural Network	19-21
3.8	Gradient Boosting (GSM)	21-25
4.	**Software Specification Requirement**	**25**
4.1	Software Requirements	25
4.2	Hardware Requirements	25
5.	**Future Work**	**25**

6.	**Problem Statement**	**25-26**
7.	**Observations**	**26**
8.	**Acknowledgement**	**26**
9.	**Conclusion**	**26**
10.	**References**	**27**

Abstract

Credit cards play a very important role in today's economy. It becomes an unavoidable part of household, business and global activities. Although using credit cards provides enormous benefits when used carefully and responsibly, significant credit and financial damages may be caused by fraudulent activities. Many techniques have been proposed to confront the growth in credit card fraud. However, all of these techniques have the same goal of avoiding credit card fraud; each one has its own drawbacks, advantages and characteristics. The advantages and disadvantages of fraud detection methods are enumerated and compared.Furthermore, a classification of mentioned techniques into two main fraud detection approaches, namely, misuses (supervised) and anomaly detection (unsupervised) is presented. Again, a classification of techniques is proposed based on capability to process the numerical and categorical datasets. Different datasets used in literature are then described and grouped into real and synthesized data and the effective and common attributes are extracted for further usage.Moreover, evaluation employed criterions in literature are collected and discussed.Consequently, open issues for credit card fraud detection are explained as guidelines for new researchers.

1. Introduction

Credit card fraud detection has drawn a lot of research interest and a number of techniques, with special emphasis on neural networks, data mining and distributed data mining have been suggested. Ghosh and Reilly[1] have proposed credit card fraud detection with a neural network. They have built a detection system, which is trained on a large sample of labeled credit card account transactions. These transactions contain example fraud cases due to lost cards, stolen cards, application fraud, counterfeit fraud, mail-order fraud, and non-received issue (NRI) fraud. Recently, Syeda et al. [2] have used parallel granular neural networks (PGNNs) for improving the speed of data mining and knowledge discovery process in credit card fraud detection. A complete system has been implemented for this purpose. Stolfo et al. [3] suggest a credit card fraud detection system (FDS) using meta learning techniques to learn models of fraudulent credit card transactions. Meta learning is a general strategy that provides a means for combining and integrating a number of separately built classifiers or models. A Meta classifier is thus trained on the correlation of the predictions of the base classifiers. The same group has also worked on a cost-based model for fraud and intrusion detection. They use Java agents for Meta learning (JAM), which is a distributed data mining system for credit card fraud detection A number of important performance metrics like True Positive—False Positive (TP-FP) spread and accuracy have been defined by them. Alekerov et al. [4] present CARDWATCH, a database mining system used for credit card fraud detection. The system, based on a neural learning module, provides an interface to a variety of commercial databases.

2. Methodology Adopted

2.1 Importing the Datasets

We are importing the datasets that contain transactions made by credit cards.

Code:

1. library(ranger)
2. library(caret)
3. library(data.table)
4. creditcard_data<- read.csv("/home/dataflair/data/Credit Card/creditcard.csv")

Input Screenshot:

```
library(ranger)
library(caret)
```

```
## Loading required package: lattice
```

```
library(data.table)
creditcard_data <- read.csv("/home/dataflair/data/Credit Card/creditcard.csv")
```

2.2 Data Exploration

In this section of the fraud detection ML project, we will explore the data that is contained in the creditcard_datadataframe. We will proceed by displaying the creditcard_data using the head () function as well as the tail () function. We will then proceed to explore the other components of this data frame.

Code:

1. **dim**(creditcard_data)
2. **head**(creditcard_data,6)

Output Screenshot:

```
dim(creditcard_data)
```

```
## [1] 284807     31
```

```
head(creditcard_data,6)
```

```
##   Time         V1          V2        V3         V4          V5          V6
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            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
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##          V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
```

Code:

1. **tail**(creditcard_data,6)

Output Screenshot:

```
tail(creditcard_data,6)
```

```
##          Time          V1          V2         V3         V4          V5
## 284802 172785   0.1203164  0.93100513 -0.5460121 -0.7450968  1.13031398
## 284803 172786 -11.8811179 10.07178497 -9.8347835 -2.0666557 -5.36447278
## 284804 172787  -0.7327887 -0.05508049  2.0350297 -0.7385886  0.86822940
## 284805 172788   1.9195650 -0.30125385 -3.2496398 -0.5578281  2.63051512
## 284806 172788  -0.2404400  0.53048251  0.7025102  0.6897992 -0.37796113
## 284807 172792  -0.5334125 -0.18973334  0.7033374 -0.5062712 -0.01254568
##                V6         V7         V8         V9        V10        V11
## 284802 -0.2359732  0.8127221  0.1150929 -0.2040635 -0.6574221  0.6448373
## 284803 -2.6068373 -4.9182154  7.3053340  1.9144283  4.3561704 -1.5931053
## 284804  1.0584153  0.0243297  0.2948687  0.5848000 -0.9759261 -0.1501888
## 284805  3.0312601 -0.2968265  0.7084172  0.4324540 -0.4847818  0.4116137
## 284806  0.6237077 -0.6861800  0.6791455  0.3920867 -0.3991257 -1.9338488
## 284807 -0.6496167  1.5770063 -0.4146504  0.4861795 -0.9154266 -1.0404583
##                V12        V13         V14         V15        V16
## 284802  0.19091623 -0.5463289 -0.73170658 -0.80803553  0.5996281
## 284803  2.71194079 -0.6892556  4.62694203 -0.92445871  1.1076406
## 284804  0.91580191  1.2147558 -0.67514296  1.16493091 -0.7117573
## 284805  0.06311886 -0.1836987 -0.51060184  1.32928351  0.1407160
## 284806 -0.96288614 -1.0420817  0.44962444  1.96256312 -0.6085771
## 284807 -0.03151305 -0.1880929 -0.08431647  0.04133346 -0.3026201
```

Code:

1. **table**(creditcard_data$Class)
2. **summary**(creditcard_data$Amount)
3. **names**(creditcard_data)
4. **var**(creditcard_data$Amount)

Output Screenshot:

```
table(creditcard_data$Class)
```

```
##
##      0      1
## 284315    492
```

```
summary(creditcard_data$Amount)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     0.00     5.60    22.00    88.35    77.17 25691.16
```

```
names(creditcard_data)
```

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

```
var(creditcard_data$Amount)
```

```
## [1] 62560.07
```

Code:

1. **sd**(creditcard_data$Amount)

Output Screenshot:

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

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

2.3 Data Manipulation

In this section of the R data science project, 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.

Code:

```
1. head(creditcard_data)
```

Output Screenshot:

```
head(creditcard_data)
```

```
##   Time         V1          V2        V3         V4          V5          V6
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            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
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##          V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##           V19         V20          V21          V22         V23
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767
```

Code:

```
1. creditcard_data$Amount=scale(creditcard_data$Amount)
2. NewData=creditcard_data[,-c(1)]
3. head(NewData)
```

Output Screenshot:

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

```
##           V1          V2        V3         V4          V5          V6
## 1 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            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
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##          V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##           V19         V20          V21          V22         V23
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767
```

2.4 Data Modeling

After we have standardized our entire dataset, we will split our dataset into training set as well as test set with a split ratio of 0.80. This means that 80% of our data will be attributed to the train_data whereas 20% will be attributed to the test data.

Code:

```
1. library(caTools)
2. set.seed(123)
3. data_sample = sample.split(NewData$Class,SplitRatio=0.80)
4. train_data = subset(NewData,data_sample==TRUE)
5. test_data = subset(NewData,data_sample==FALSE)
6. dim(train_data)
```

7. **dim**(test_data)

Output Screenshot:

```
library(caTools)
set.seed(123)
data_sample = sample.split(NewData$Class,SplitRatio=0.80)
train_data = subset(NewData,data_sample==TRUE)
test_data = subset(NewData,data_sample==FALSE)
dim(train_data)
```

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

```
dim(test_data)
```

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

```
  Logistic_Model=glm(Class~.,test_data,family=binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
  summary(Logistic_Model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = test_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.9019  -0.0254  -0.0156  -0.0078   4.0877
```

2.5 Fitting Logistic Regression Model

In this section of credit card fraud detection project, we will fit our first model. We will begin with logistic regression. A logistic regression is used for modeling the outcome probability of a class such as pass/fail, positive/negative and in our case – fraud/not fraud.

Code:

1. Logistic_Model=**glm**(Class~.,test_data,family=**binomial**())
2. **summary**(Logistic_Model)

Output Screenshot:

```
Logistic_Model=glm(Class~.,test_data,family=binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(Logistic_Model)
```

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = test_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.9019  -0.0254  -0.0156  -0.0078   4.0877
```

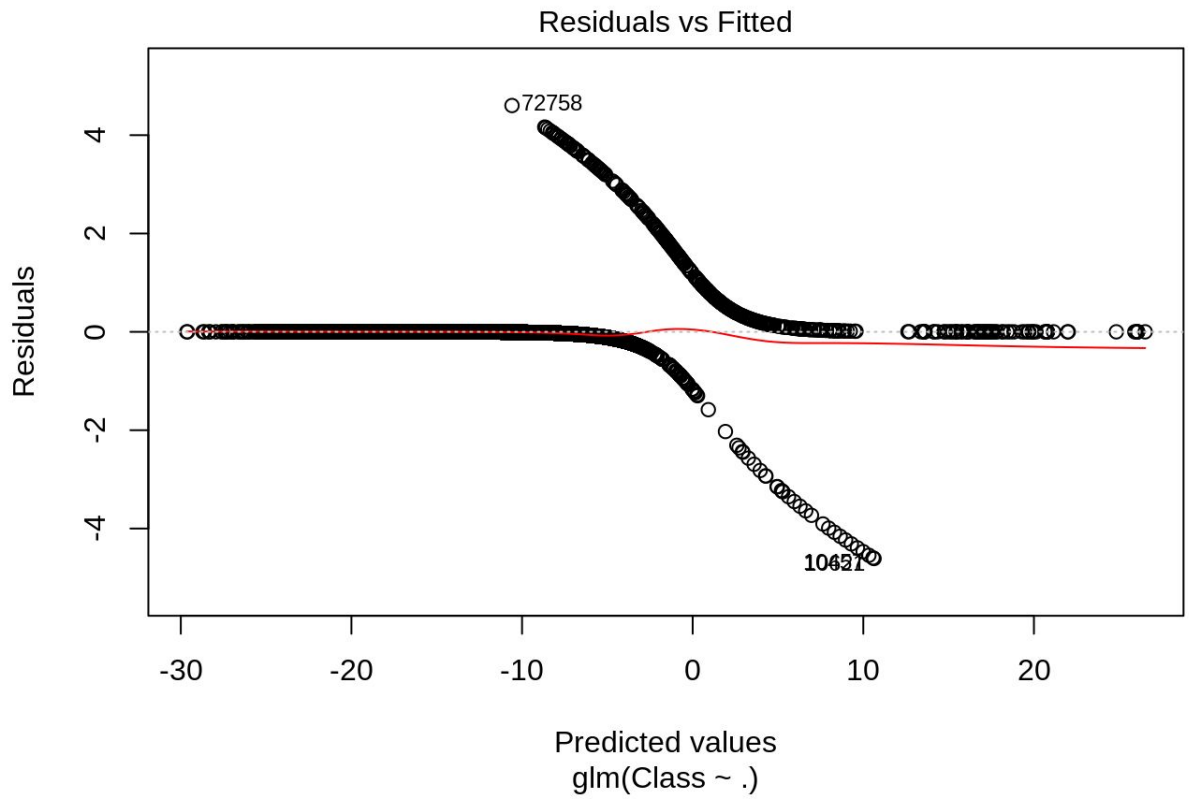
Code:

1. **plot**(Logistic_Model)

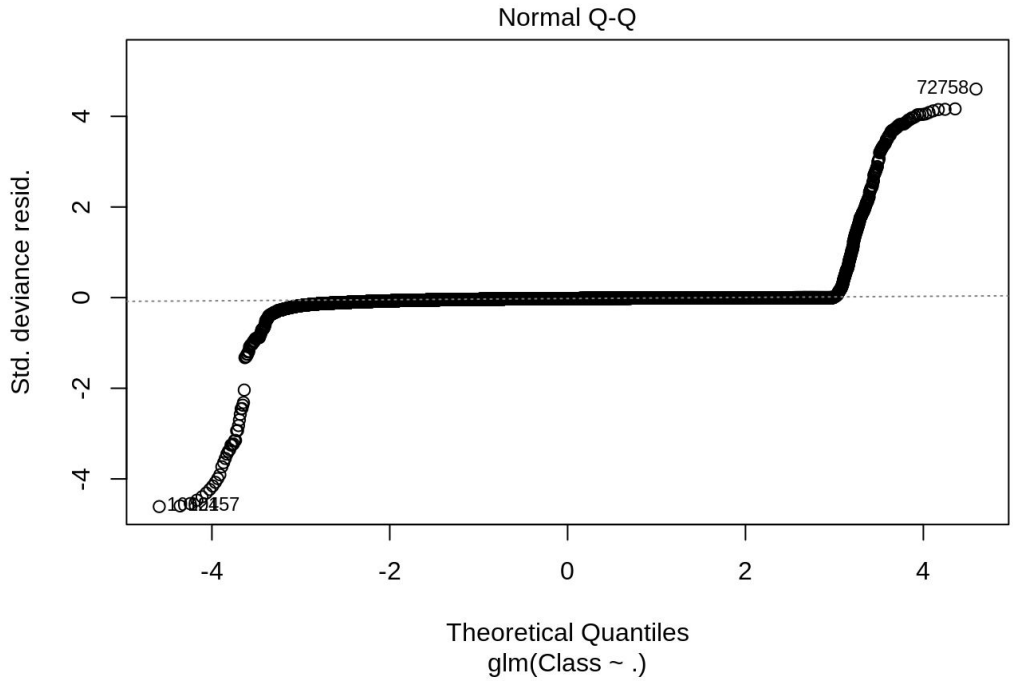
Input Screenshot:

```
plot(Logistic_Model)
```

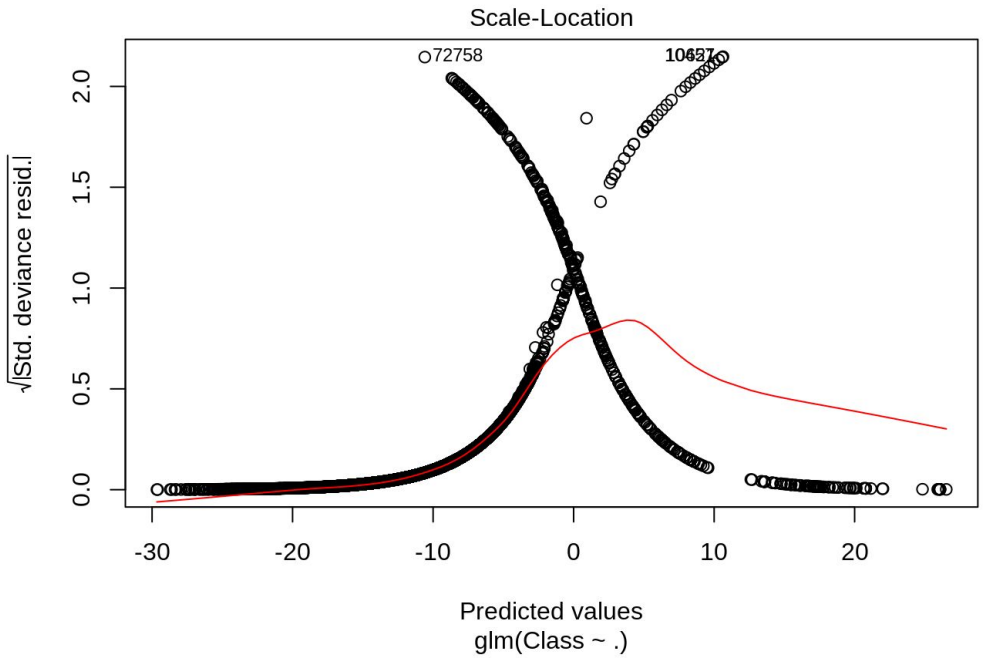
Output:


Residuals vs Fitted
72758
Residuals
4
2
0
-2
-4
-30
-20
-10
0
10
20
Predicted values
glm(Class ~ .)

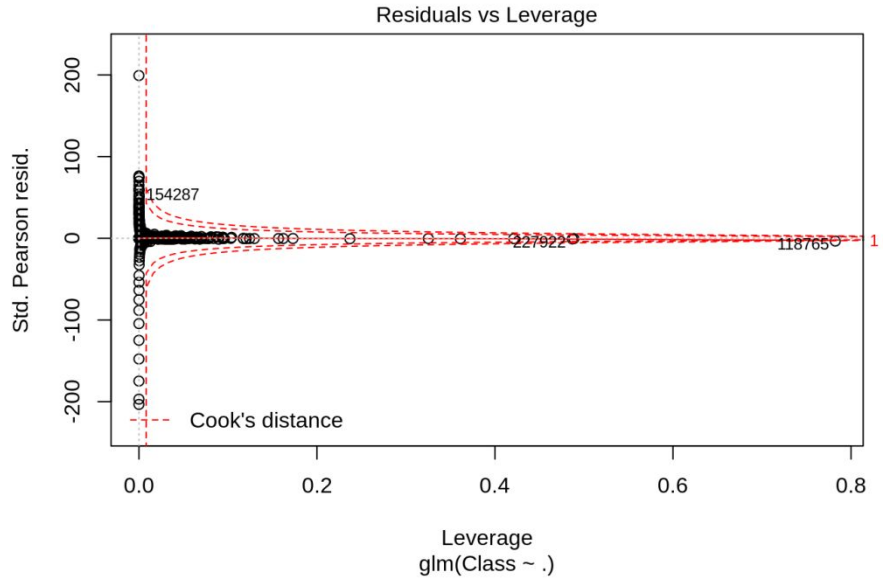
Output:


Normal Q-Q
72758
Std. deviance resid.
4
2
0
-2
-4
-4
-2
0
2
4
Theoretical Quantiles
glm(Class ~ .)

Output:



Output:



In order to assess the performance of our model, we will delineate the ROC curve. ROC is also known as Receiver Optimistic Characteristics. For this, we will first import the ROC package and then plot our ROC curve to analyze its performance.

Code:

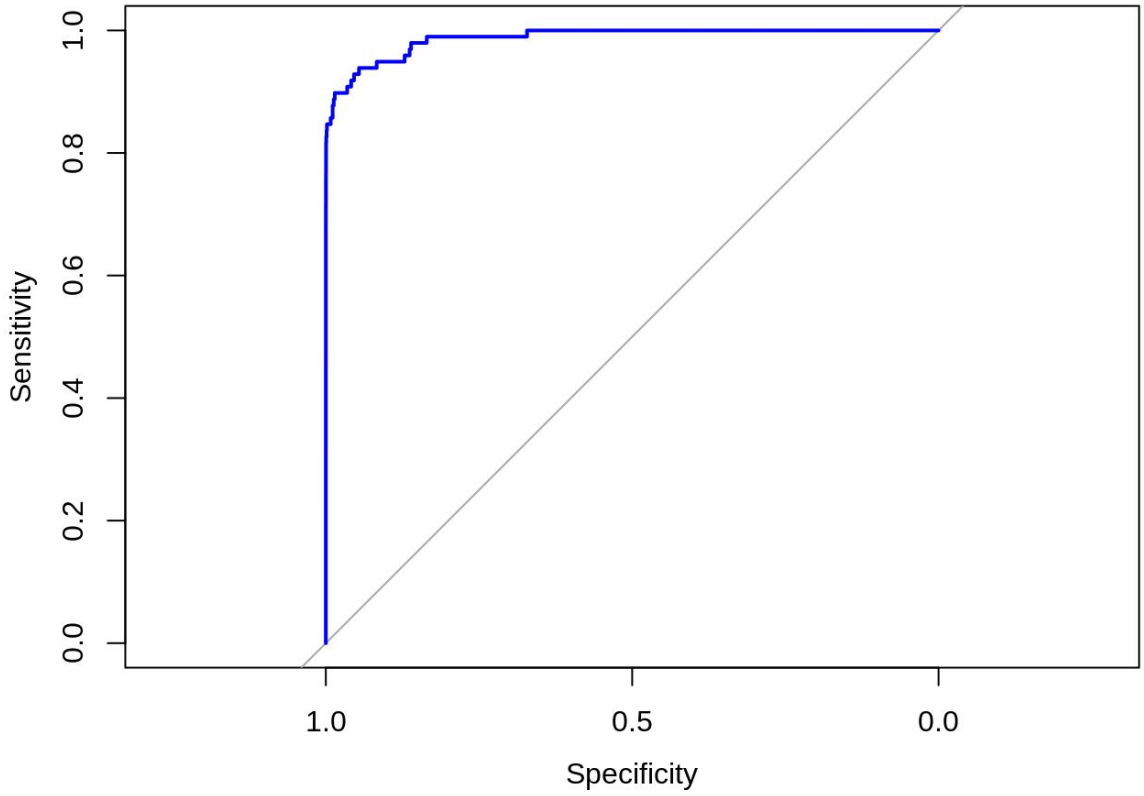
```
1. library(pROC)
2. lr.predict<- predict(Logistic_Model,train_data, probability = TRUE)
3. auc.gbm = roc(test_data$Class, lr.predict, plot = TRUE, col = "blue")
```

Output Screenshot:

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

```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = train_data)
##
## 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
```

Output:



2.6 Fitting a Decision Tree Model

In this section, we will implement a decision tree algorithm. Decision Trees to plot the outcomes of a 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.

Code:

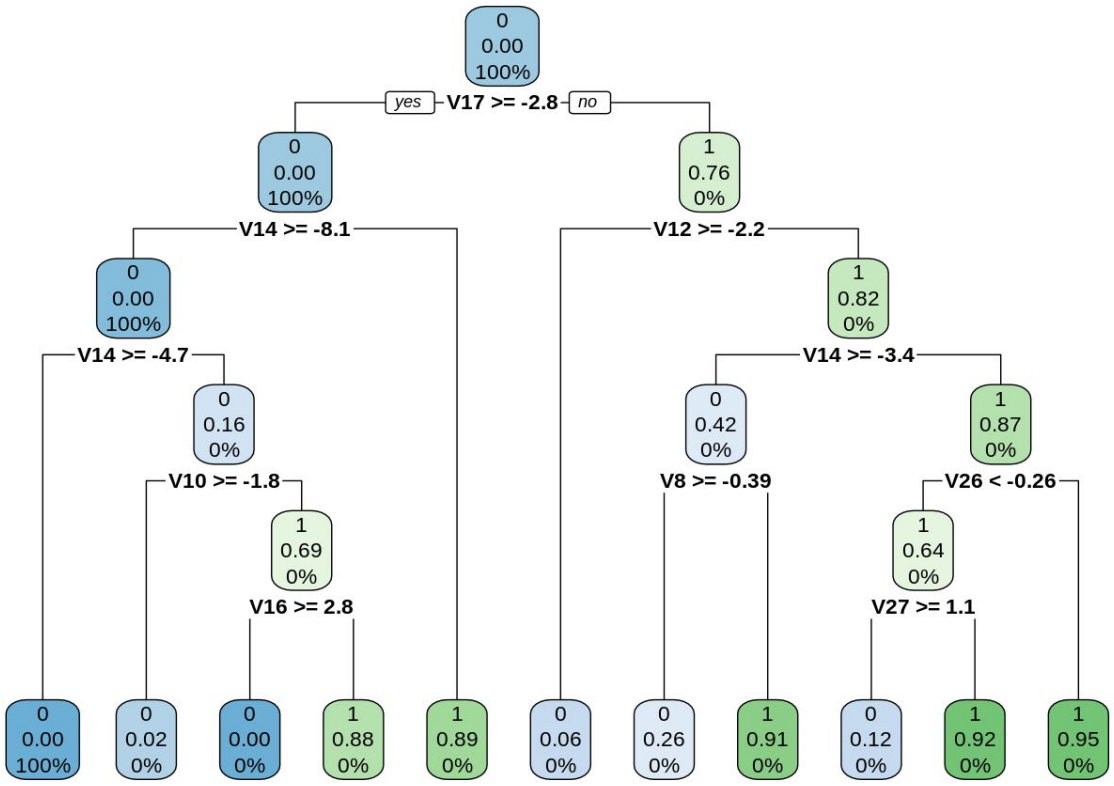
```
1. library(rpart)
2. library(rpart.plot)
3. decisionTree_model<- rpart(Class ~ . , creditcard_data, method = 'class')
4. predicted_val<- predict(decisionTree_model, creditcard_data, type = 'class')
5. probability <- predict(decisionTree_model, creditcard_data, type = 'prob')
6. rpart.plot(decisionTree_model)
```

Input Screenshot:

```
library(rpart)
library(rpart.plot)
decisionTree_model <- rpart(Class ~ . , creditcard_data, method = 'class')
predicted_val <- predict(decisionTree_model, creditcard_data, type = 'class')
probability <- predict(decisionTree_model, creditcard_data, type = 'prob')

rpart.plot(decisionTree_model)
```

Output:



2.7 Artificial Neural Network

Artificial Neural Networks are a type of machine learning algorithm that is modeled after the human nervous system. The ANN models are able to learn the patterns using the historical data and are able to perform classification on the input data. We import the neuralnet package that would allow us to implement our ANNs. Then we proceeded to plot it using the plot() function. Now, in the case of Artificial Neural Networks, there is a

range of values that is between 1 and 0. We set a threshold as 0.5, that is, values above 0.5 will correspond to 1 and the rest will be 0.

Code:

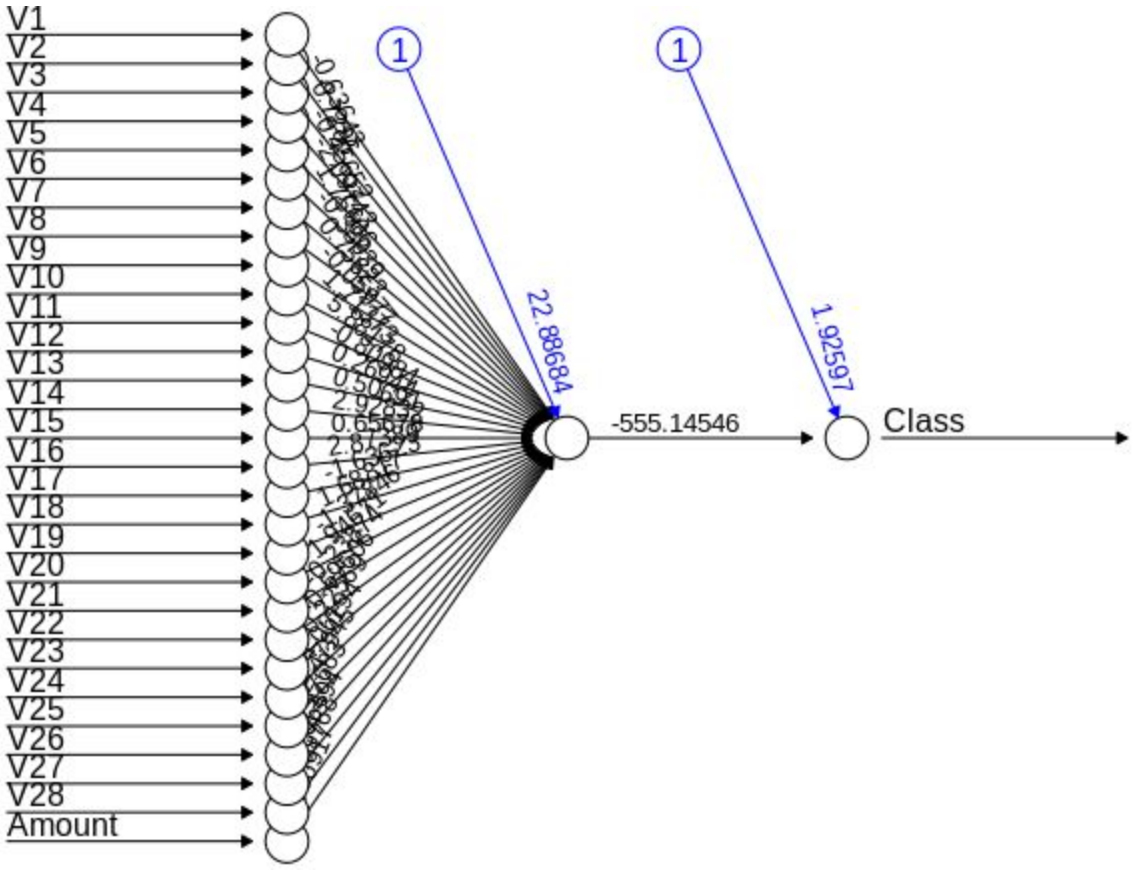
```
1. library(neuralnet)
2. ANN_model =neuralnet(Class~.,train_data,linear.output=FALSE)
3. plot(ANN_model)
4.
5. predANN=compute(ANN_model,test_data)
6. resultANN=predANN$net.result
7. resultANN=ifelse(resultANN>0.5,1,0)
```

Input Screenshot:

```
library(neuralnet)
ANN_model =neuralnet (Class~.,train_data,linear.output=FALSE)
plot(ANN_model)
```

```
predANN=compute(ANN_model,test_data)
resultANN=predANN$net.result
resultANN=ifelse(resultANN>0.5,1,0)
```

Output:



2.8 Gradient Boosting (GSM)

Gradient Boosting is a popular machine learning algorithm that is used to perform classification and regression tasks. This model comprises of several underlying ensemble models like weak decision trees. These decision trees combine together to form a strong model of gradient boosting.

Code:

```
1.  library(gbm, quietly=TRUE)
2.
3.  # Get the time to train the GBM model
4.  system.time(
5.  model_gbm<- gbm(Class ~ .
6.  , distribution = "bernoulli"
7.  , data = rbind(train_data, test_data)
8.  , n.trees = 500
9.  , interaction.depth = 3
10. , n.minobsinnode = 100
```

```
11. , shrinkage = 0.01
12. , bag.fraction = 0.5
13. , train.fraction = nrow(train_data) / (nrow(train_data) + nrow(test_data))
14. )
15. )
16. # Determine best iteration based on test data
17. gbm.iter = gbm.perf(model_gbm, method = "test")
```

Input Screenshot:

```
library(gbm, quietly=TRUE)
```

```
## Loaded gbm 2.1.5
```

```
# Get the time to train the GBM model
system.time(
    model_gbm <- gbm(Class ~ .
        , distribution = "bernoulli"
        , data = rbind(train_data, test_data)
        , n.trees = 500
        , interaction.depth = 3
        , n.minobsinnode = 100
        , shrinkage = 0.01
        , bag.fraction = 0.5
        , train.fraction = nrow(train_data) / (nrow(train_data) + nrow(test_data))
        )
)
```

```
##    user  system elapsed
## 345.781   0.144 345.971
```

```
# Determine best iteration based on test data
gbm.iter = gbm.perf(model_gbm, method = "test")
```

Code:

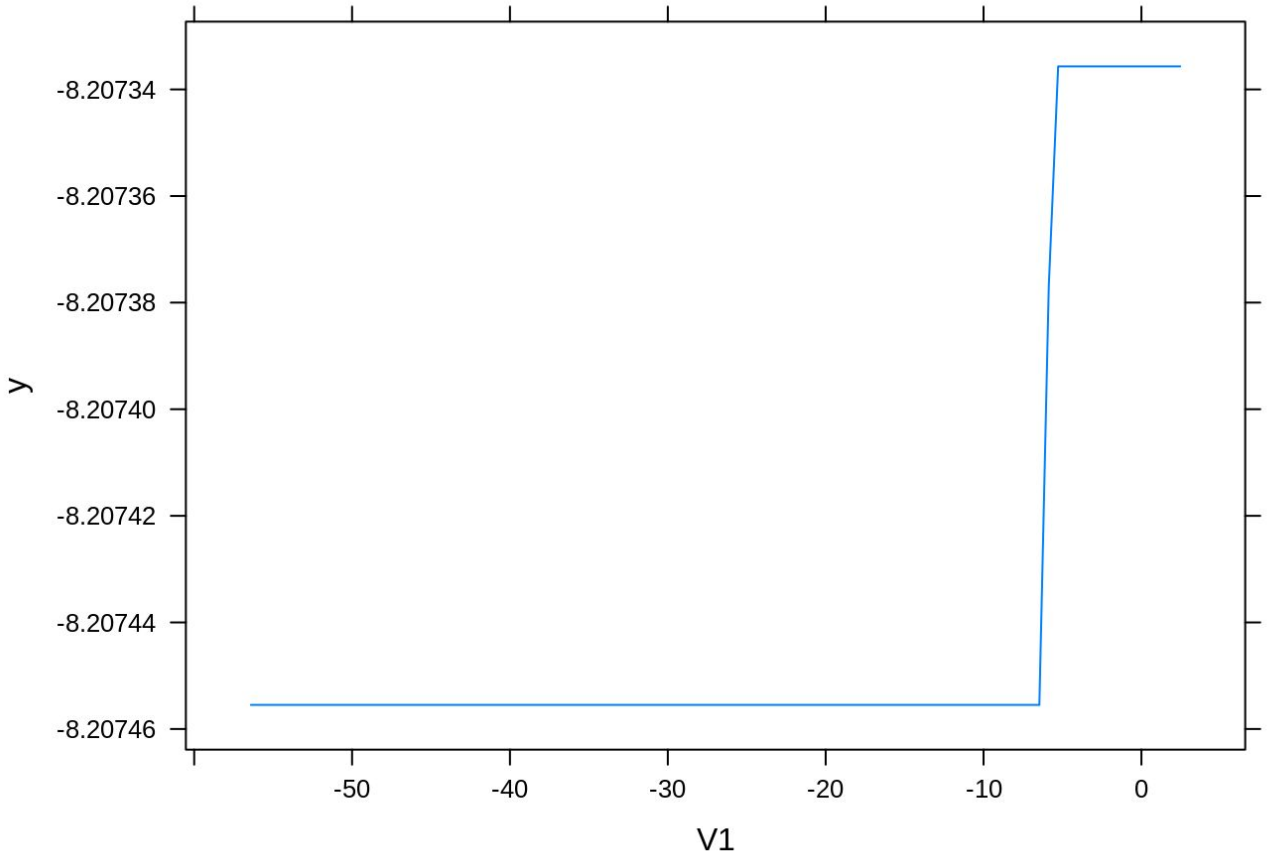
```
1. model.influence = relative.influence(model_gbm, n.trees = gbm.iter, sort. = TRUE)
2. #Plot the gbm model
3.
4. plot(model_gbm)
```

Input Screenshot:

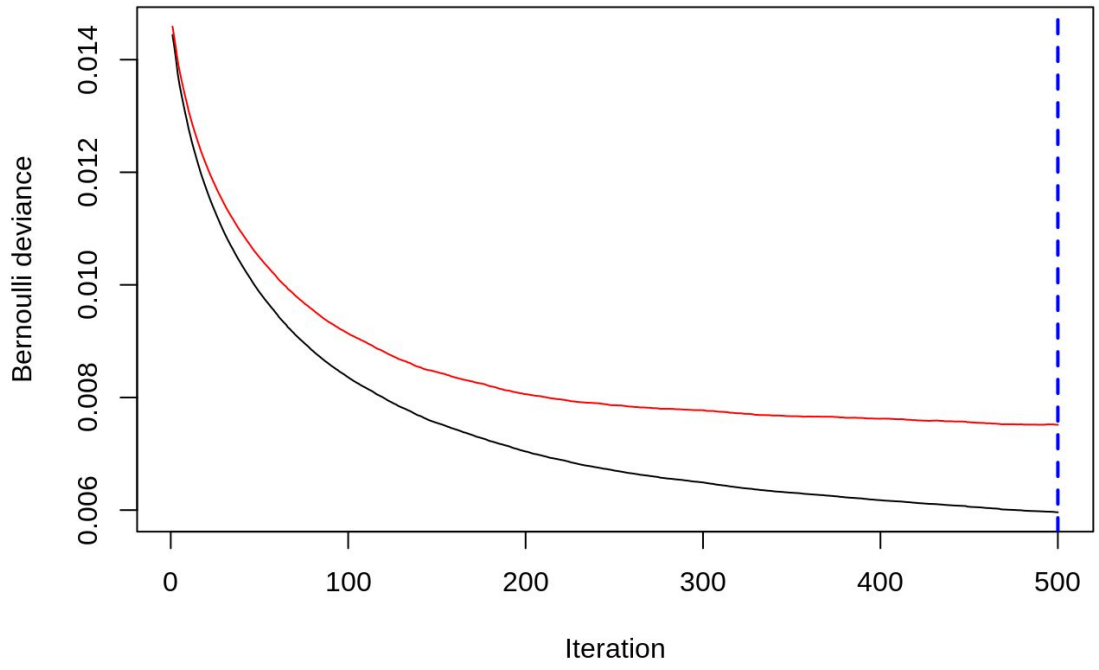
```
model.influence = relative.influence(model_gbm, n.trees = gbm.iter, sort. = TRUE)
#Plot the gbm model

plot(model_gbm)
```

Output:


-8.20734
-8.20736
-8.20738
-8.20740
-8.20742
-8.20744
-8.20746
y
-50
-40
-30
-20
-10
0
V1

Output:



Code:

```
1. # Plot and calculate AUC on test data
2. gbm_test = predict(model_gbm, newdata = test_data, n.trees = gbm.iter)
3. gbm_auc = roc(test_data$Class, gbm_test, plot = TRUE, col = "red")
```

Output Screenshot:

```
# Plot and calculate AUC on test data
gbm_test = predict(model_gbm, newdata = test_data, n.trees = gbm.iter)
gbm_auc = roc(test_data$Class, gbm_test, plot = TRUE, col = "red")
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

Code:

```
1. print(gbm_auc)
```

Output Screenshot:

```
print(gbm_auc)
```

```
##
## Call:
## roc.default(response = test_data$Class, predictor = gbm_test,     plot = TRUE, col = "red")
##
## Data: gbm_test in 56863 controls (test_data$Class 0) < 98 cases (test_data$Class 1).
## Area under the curve: 0.9555
```

3. Software Specification Requirement

3.1 Software Requirements:

- R-Studio

3.2 Hardware Requirements:

- Processor: Preferably 1.0 GHz or Greater.
- RAM : 2 GB or Greater.

4. Future Work - Fraud detection is a complex issue that requires a substantial amount of planning before throwing machine learning algorithms at it. Nonetheless, it is also an application of data science and machine learning for the good, which makes sure that the customer's money is safe and not easily tampered with. Future work will include a comprehensive tuning of the Random Forest algorithm I talked about earlier. Having a data set with non-anonymized features would make this particularly interesting as outputting the feature importance would enable one to see what specific factors are most important for detecting fraudulent transactions.

5. Problem Statement

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify

whether a new transaction is fraudulent or not. Our aim here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

6. Observations-

The data set is highly skewed, consisting of 492 frauds in a total of 284,807 observations. This resulted in only 0.172% fraud cases. This skewed set is justified by the low number of fraudulent transactions.The dataset consists of numerical values from the 28 'Principal Component Analysis (PCA)' transformed features, namely V1 to V28. Furthermore, there is no metadata about the original features provided, so pre-analysis or feature study could not be done.

- The 'Time' and 'Amount' features are not transformed data.
- There is no missing value in the dataset.

7. Acknowledgement-

I would like to acknowledge my guide Dr. Kuldeep Singh Kaswan for providing me the necessary guidance and valuable support throughout this research project. Learning from their knowledge helped me to become passionate about my research topic

8. Conclusion -

Concluding our R Data Science project, I learnt how to develop our credit card fraud detection model using machine learning. I used a variety of ML algorithms to implement this model and also plotted the respective performance curves for the models. I learnt how data can be analyzed and visualized to discern fraudulent transactions from other types of data.

9. References-

1. R. J. Bolton and D. J. Hand. Unsupervised profiling methods for fraud detection. In conference of Credit Scoring and Credit Connol VII, Edinburgh. UK, Sept 5-7,2001.
2. KhyatiChaudhary, JyotiYadav, BhawnaMallick, ―A review of Fraud Detection Techniques: Credit Card‖ , International Journal of Computer Applications (0975 – 8887) Volume 45– No.1, May 2012.
3. K. C. Cox, S. G. Eick, G. J. Wills, and R. J. Brachman. Visual data mining: Recognizing telephone calling fraud.J Data Mining and Knowledge Discover, 1(2):22>231, 1997.
4. Hollman and Jaakko. PmbabilisticAppmaches to Fraud Detection, Licentiate's thesis. Helsinki University of Technology, Department of Computer Science and Engineering, 1999.
5. https://rpubs.com/slazien/fraud_detection