By: PAWAN KUMAR AMAT

(IIT BOMBAY)

Data Science Project – Detect Credit Card Fraud with Machine Learning in R

Machine Learning Project – How to Detect Credit Card Fraud

The aim of this R project is to build a classifier that can detect credit card fraudulent transactions. We will use a variety of ***machine learning algorithms*** that will be able to discern fraudulent from non-fraudulent one. By the end of this machine learning project, you will learn how to implement machine learning algorithms to perform classification.

1. Importing the Datasets

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

```
install.packages("lattice")
library(ranger)
library(caret)
library(data.table) creditcard <- read.csv("C:/Users/PAWAN/Downloads/Credi
t-Card-Dataset/creditcard.csv")
> View(creditcard)
```

2. Data Exploration

In this section of the fraud detection ML project, we will explore the data that is contained in the creditcard_data dataframe. 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 dataframe –

```
> dim(creditcard)
[1] 284807     31
> head(creditcard,6)
  Time         V1          V2        V3         V4          V5          V6
V7          V8
1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
0.23959855  0.09869790
2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
-0.07880298  0.08510165
```

```
3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
0.79146096  0.24767579
4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
0.23760894  0.37743587
5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
0.59294075 -0.27053268
6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
0.47620095  0.26031433
          V9         V10        V11         V12        V13        V14
V15        V16
1  0.3637870  0.09079417 -0.5515995 -0.61780086 -0.9913898 -0.3111694  1.4
681770 -0.4704005
2 -0.2554251 -0.16697441  1.6127267  1.06523531  0.4890950 -0.1437723  0.6
355581  0.4639170
3 -1.5146543  0.20764287  0.6245015  0.06608369  0.7172927 -0.1659459  2.3
458649 -2.8900832
4 -1.3870241 -0.05495192 -0.2264873  0.17822823  0.5077569 -0.2879237 -0.6
314181 -1.0596472
5  0.8177393  0.75307443 -0.8228429  0.53819555  1.3458516 -1.1196698  0.1
751211 -0.4514492
6 -0.5686714 -0.37140720  1.3412620  0.35989384 -0.3580907 -0.1371337  0.5
176168  0.4017259
          V17         V18         V19         V20          V21          V2
2         V23         V24
1  0.20797124  0.02579058  0.40399296  0.25141210 -0.018306778  0.27783757
6 -0.11047391  0.06692807
2 -0.11480466 -0.18336127 -0.14578304 -0.06908314 -0.225775248 -0.63867195
3  0.10128802 -0.33984648
3  1.10996938 -0.12135931 -2.26185710  0.52497973  0.247998153  0.77167940
2  0.90941226 -0.68928096
4 -0.68409279  1.96577500 -1.23262197 -0.20803778 -0.108300452  0.00527359
7 -0.19032052 -1.17557533
5 -0.23703324 -0.03819479  0.80348692  0.40854236 -0.009430697  0.79827849
5 -0.13745808  0.14126698
6 -0.05813282  0.06865315 -0.03319379  0.08496767 -0.208253515 -0.55982479
6 -0.02639767 -0.37142658
         V25        V26          V27         V28 Amount Class
1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
6 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0
> tail(creditcard,6)
         Time          V1          V2         V3         V4          V5
V6         V7
284802 172785   0.1203164  0.93100513 -0.5460121 -0.7450968  1.13031398 -0
.2359732  0.8127221
284803 172786 -11.8811179 10.07178497 -9.8347835 -2.0666557 -5.36447278 -2
.6068373 -4.9182154
284804 172787  -0.7327887 -0.05508049  2.0350297 -0.7385886  0.86822940  1
.0584153  0.0243297
284805 172788   1.9195650 -0.30125385 -3.2496398 -0.5578281  2.63051512  3
.0312601 -0.2968265
284806 172788  -0.2404400  0.53048251  0.7025102  0.6897992 -0.37796113  0
.6237077 -0.6861800
284807 172792  -0.5334125 -0.18973334  0.7033374 -0.5062712 -0.01254568 -0
.6496167  1.5770063
               V8         V9        V10        V11         V12        V13
V14          V15
284802  0.1150929 -0.2040635 -0.6574221  0.6448373  0.19091623 -0.5463289
-0.73170658 -0.80803553
284803  7.3053340  1.9144283  4.3561704 -1.5931053  2.71194079 -0.6892556
4.62694203 -0.92445871
284804  0.2948687  0.5848000 -0.9759261 -0.1501888  0.91580191  1.2147558
-0.67514296  1.16493091
284805  0.7084172  0.4324540 -0.4847818  0.4116137  0.06311886 -0.1836987
-0.51060184  1.32928351
```

```
284806  0.6791455  0.3920867 -0.3991257 -1.9338488 -0.96288614 -1.0420817
0.44962444  1.96256312
284807 -0.4146504  0.4861795 -0.9154266 -1.0404583 -0.03151305 -0.1880929
-0.08431647  0.04133346
              V16         V17        V18        V19          V20        V2
1          V22         V23
284802  0.5996281  0.07044075  0.3731103  0.1289038 0.0006758329 -0.314204
6 -0.8085204  0.05034266
284803  1.1076406  1.99169111  0.5106323 -0.6829197 1.4758291347  0.213454
1  0.1118637  1.01447990
284804 -0.7117573 -0.02569286 -1.2211789 -1.5455561 0.0596158999  0.214205
3  0.9243836  0.01246304
284805  0.1407160  0.31350179  0.3956525 -0.5772518 0.0013959703  0.232045
0  0.5782290 -0.03750086
284806 -0.6085771  0.50992846  1.1139806  2.8978488 0.1274335158  0.265244
9  0.8000487 -0.16329794
284807 -0.3026201 -0.66037665  0.1674299 -0.2561169 0.3829481049  0.261057
3  0.6430784  0.37677701
                V24        V25        V26          V27         V28 Amount
Class
284802  0.102799590 -0.4358701  0.1240789  0.217939865  0.06880333   2.69
0
284803 -0.509348453  1.4368069  0.2500343  0.943651172  0.82373096   0.77
0
284804 -1.016225669 -0.6066240 -0.3952551  0.068472470 -0.05352739  24.79
0
284805  0.640133881  0.2657455 -0.0873706  0.004454772 -0.02656083  67.88
0
284806  0.123205244 -0.5691589  0.5466685  0.108820735  0.10453282  10.00
0
284807  0.008797379 -0.4736487 -0.8182671 -0.002415309  0.01364891 217.00
0
> table(creditcard$Class)

     0      1
284315    492
> summary(creditcard$Amount)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
    0.00     5.60    22.00    88.35    77.17 25691.16
> names(creditcard)
 [1] "Time"     "V1"      "V2"      "V3"      "V4"      "V5"      "V6"      "V7"
"V8"      "V9"
[11] "V10"     "V11"     "V12"     "V13"     "V14"     "V15"     "V16"     "V17"
"V18"     "V19"
[21] "V20"     "V21"     "V22"     "V23"     "V24"     "V25"     "V26"     "V27"
"V28"     "Amount"
[31] "Class"
> var(creditcard$Amount)
[1] 62560.07
> sd(creditcard$Amount)
[1] 250.1201
```

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. We will carry this out as follows:

```
> head(creditcard)
```

```
  Time         V1          V2        V3         V4          V5          V6
V7          V8
1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
0.23959855  0.09869790
2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
-0.07880298  0.08510165
3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
0.79146096  0.24767579
4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
0.23760894  0.37743587
5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
0.59294075 -0.27053268
6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
0.47620095  0.26031433
          V9         V10        V11         V12        V13        V14
V15        V16
1  0.3637870  0.09079417 -0.5515995 -0.61780086 -0.9913898 -0.3111694  1.4
681770 -0.4704005
2 -0.2554251 -0.16697441  1.6127267  1.06523531  0.4890950 -0.1437723  0.6
355581  0.4639170
3 -1.5146543  0.20764287  0.6245015  0.06608369  0.7172927 -0.1659459  2.3
458649 -2.8900832
4 -1.3870241 -0.05495192 -0.2264873  0.17822823  0.5077569 -0.2879237 -0.6
314181 -1.0596472
5  0.8177393  0.75307443 -0.8228429  0.53819555  1.3458516 -1.1196698  0.1
751211 -0.4514492
6 -0.5686714 -0.37140720  1.3412620  0.35989384 -0.3580907 -0.1371337  0.5
176168  0.4017259
          V17         V18         V19         V20          V21          V2
2         V23         V24
1  0.20797124  0.02579058  0.40399296  0.25141210 -0.018306778  0.27783757
6 -0.11047391  0.06692807
2 -0.11480466 -0.18336127 -0.14578304 -0.06908314 -0.225775248 -0.63867195
3  0.10128802 -0.33984648
3  1.10996938 -0.12135931 -2.26185710  0.52497973  0.247998153  0.77167940
2  0.90941226 -0.68928096
4 -0.68409279  1.96577500 -1.23262197 -0.20803778 -0.108300452  0.00527359
7 -0.19032052 -1.17557533
5 -0.23703324 -0.03819479  0.80348692  0.40854236 -0.009430697  0.79827849
5 -0.13745808  0.14126698
6 -0.05813282  0.06865315 -0.03319379  0.08496767 -0.208253515 -0.55982479
6 -0.02639767 -0.37142658
         V25        V26          V27         V28 Amount Class
1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
6 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0
> creditcard$Amount=scale(creditcard$Amount)
> NewData=creditcard[,-c(1)]
> head(NewData)
          V1          V2        V3         V4          V5          V6
V7          V8
1 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778  0.2
3959855  0.09869790
2  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081 -0.0
7880298  0.08510165
3 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938  0.7
9146096  0.24767579
4 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317  0.2
3760894  0.37743587
5 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146  0.5
9294075 -0.27053268
6 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755  0.4
7620095  0.26031433
          V9         V10        V11         V12        V13        V14
V15        V16
```

```
1  0.3637870  0.09079417 -0.5515995 -0.61780086 -0.9913898 -0.3111694  1.4
681770 -0.4704005
2 -0.2554251 -0.16697441  1.6127267  1.06523531  0.4890950 -0.1437723  0.6
355581  0.4639170
3 -1.5146543  0.20764287  0.6245015  0.06608369  0.7172927 -0.1659459  2.3
458649 -2.8900832
4 -1.3870241 -0.05495192 -0.2264873  0.17822823  0.5077569 -0.2879237 -0.6
314181 -1.0596472
5  0.8177393  0.75307443 -0.8228429  0.53819555  1.3458516 -1.1196698  0.1
751211 -0.4514492
6 -0.5686714 -0.37140720  1.3412620  0.35989384 -0.3580907 -0.1371337  0.5
176168  0.4017259
          V17         V18         V19         V20          V21          V2
2         V23         V24
1  0.20797124  0.02579058  0.40399296  0.25141210 -0.018306778  0.27783757
6 -0.11047391  0.06692807
2 -0.11480466 -0.18336127 -0.14578304 -0.06908314 -0.225775248 -0.63867195
3  0.10128802 -0.33984648
3  1.10996938 -0.12135931 -2.26185710  0.52497973  0.247998153  0.77167940
2  0.90941226 -0.68928096
4 -0.68409279  1.96577500 -1.23262197 -0.20803778 -0.108300452  0.00527359
7 -0.19032052 -1.17557533
5 -0.23703324 -0.03819479  0.80348692  0.40854236 -0.009430697  0.79827849
5 -0.13745808  0.14126698
6 -0.05813282  0.06865315 -0.03319379  0.08496767 -0.208253515 -0.55982479
6 -0.02639767 -0.37142658
         V25        V26          V27         V28      Amount Class
1  0.1285394 -0.1891148  0.133558377 -0.02105305  0.24496383     0
2  0.1671704  0.1258945 -0.008983099  0.01472417 -0.34247394     0
3 -0.3276418 -0.1390966 -0.055352794 -0.05975184  1.16068389     0
4  0.6473760 -0.2219288  0.062722849  0.06145763  0.14053401     0
5 -0.2060096  0.5022922  0.219422230  0.21515315 -0.07340321     0
6 -0.2327938  0.1059148  0.253844225  0.08108026 -0.33855582     0
```

4. Data Modeling

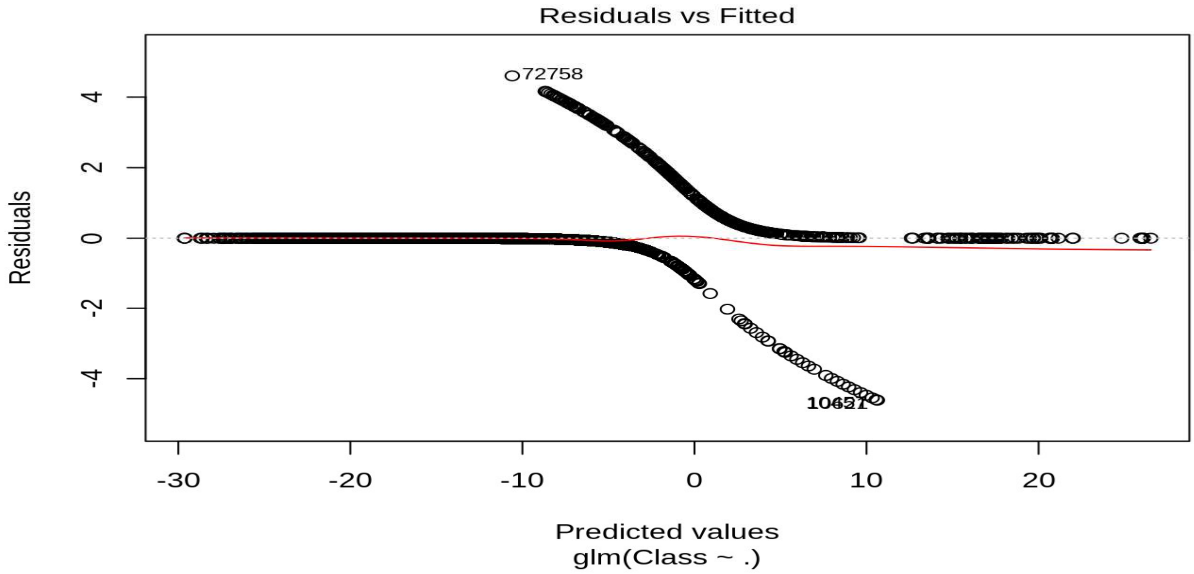
After we have standardized our entire dataset, we will split our dataset into training set as well as test set with a split ratio of 0.80. This means that 80% of our data will be attributed to the train_data whereas 20% will be attributed to the test data. We will then find the dimensions using the dim() function –

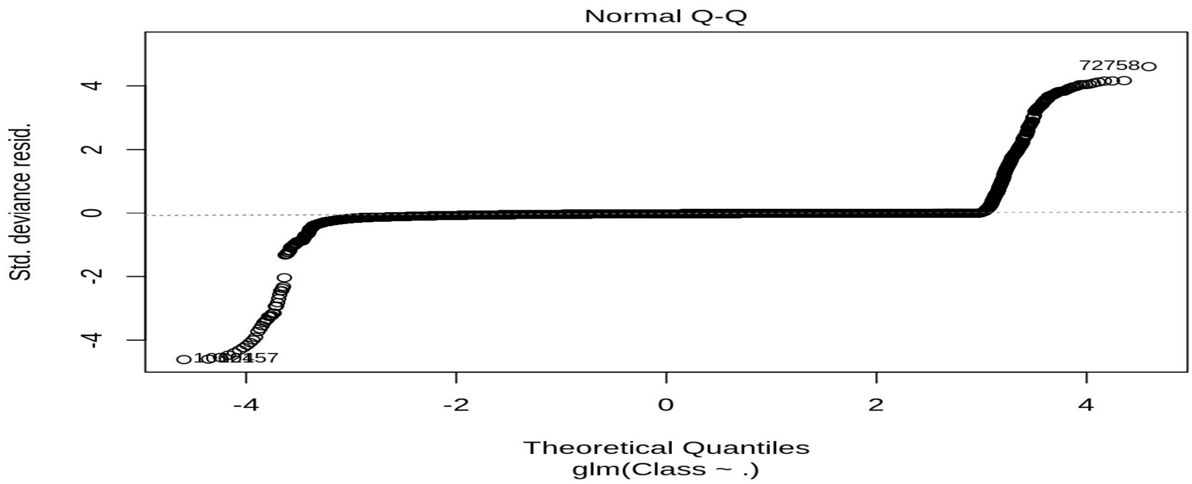
```
> library(caTools)
> set.seed(123)
> data_sample = sample.split(NewData$Class,SplitRatio=0.80)
> train_data = subset(NewData,data_sample==TRUE)
> test_data = subset(NewData,data_sample==FALSE)
> dim(train_data)
[1] 227846     30
> dim(test_data)
[1] 56961    30
```

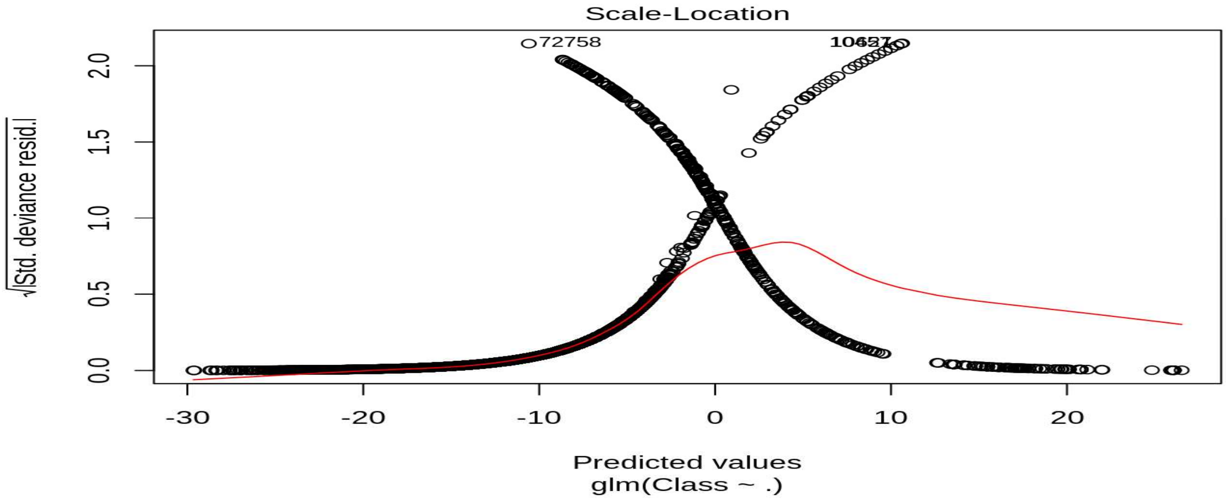
5. Fitting Logistic Regression Model

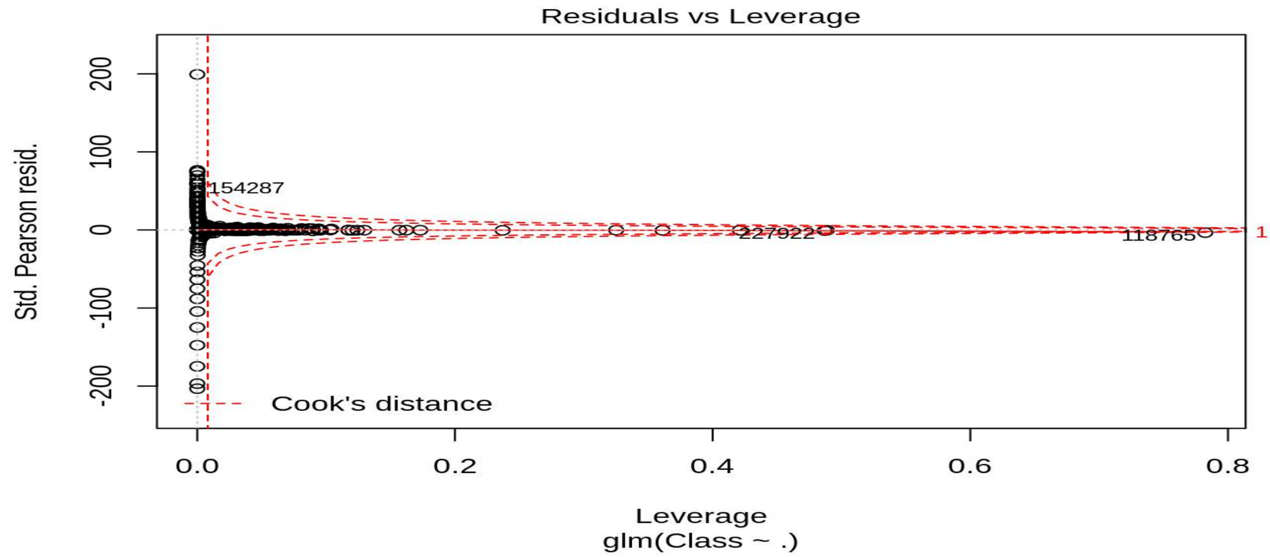
In this section of credit card fraud detection project, we will fit our first model. We will begin with logistic regression. A logistic regression is used for modeling the outcome probability of a class such as pass/fail, positive/negative and in our case – fraud/not fraud. We proceed to implement this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

```
> Logistic_Model=glm(Class~.,test_data,family=binomial())
Warning message:
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

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -12.52800   10.30537  -1.216   0.2241
V1           -0.17299    1.27381  -0.136   0.8920
V2            1.44512    4.23062   0.342   0.7327
V3            0.17897    0.24058   0.744   0.4569
V4            3.13593    7.17768   0.437   0.6622
V5            1.49014    3.80369   0.392   0.6952
V6           -0.12428    0.22202  -0.560   0.5756
V7            1.40903    4.22644   0.333   0.7388
V8           -0.35254    0.17462  -2.019   0.0435 *
V9            3.02176    8.67262   0.348   0.7275
V10          -2.89571    6.62383  -0.437   0.6620
V11          -0.09769    0.28270  -0.346   0.7297
V12           1.97992    6.56699   0.301   0.7630
V13          -0.71674    1.25649  -0.570   0.5684
V14           0.19316    3.28868   0.059   0.9532
V15           1.03868    2.89256   0.359   0.7195
V16          -2.98194    7.11391  -0.419   0.6751
V17          -1.81809    4.99764  -0.364   0.7160
V18           2.74772    8.13188   0.338   0.7354
V19          -1.63246    4.77228  -0.342   0.7323
V20          -0.69925    1.15114  -0.607   0.5436
V21          -0.45082    1.99182  -0.226   0.8209
V22          -1.40395    5.18980  -0.271   0.7868
V23           0.19026    0.61195   0.311   0.7559
V24          -0.12889    0.44701  -0.288   0.7731
V25          -0.57835    1.94988  -0.297   0.7668
V26           2.65938    9.34957   0.284   0.7761
V27          -0.45396    0.81502  -0.557   0.5775
V28          -0.06639    0.35730  -0.186   0.8526
Amount        0.22576    0.71892   0.314   0.7535
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1443.40  on 56960  degrees of freedom
Residual deviance:  378.59  on 56931  degrees of freedom
AIC: 438.59

Number of Fisher Scoring iterations: 17

plot(Logistic_Model)
```

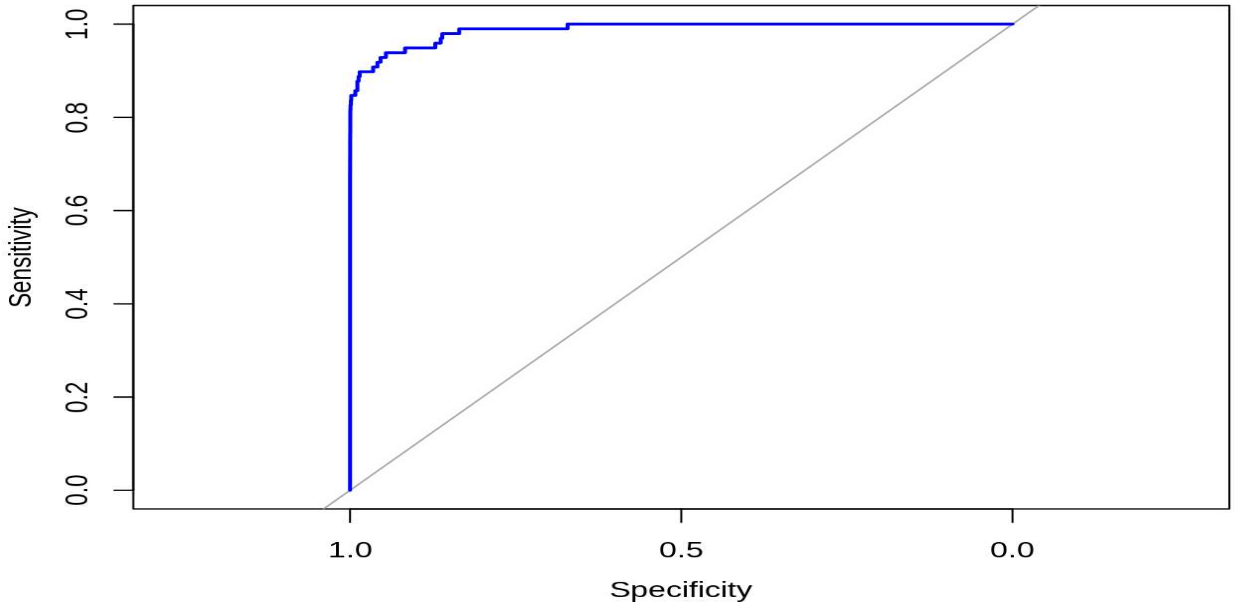

Residuals vs Fitted
Residuals
Predicted values
glm(Class ~ .)
72758
10621
4
2
0
-2
-4
-30
-20
-10
0
10
20


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





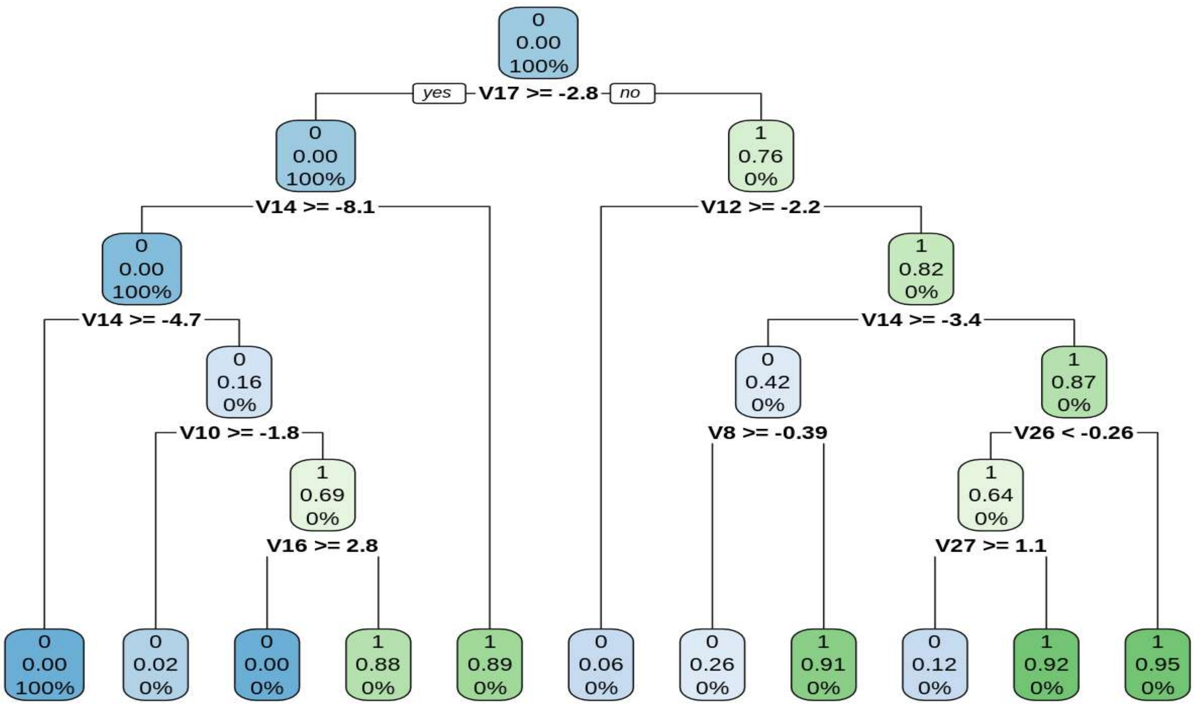

```
library(rpart.plot)
lr.predict <- predict(Logistic_Model,train_data, probability = TRUE)
auc.gbm = roc_ (test_data$Class, lr.predict, plot = TRUE, col = "blue")
```



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. We will specifically use the recursive parting to plot the decision tree.

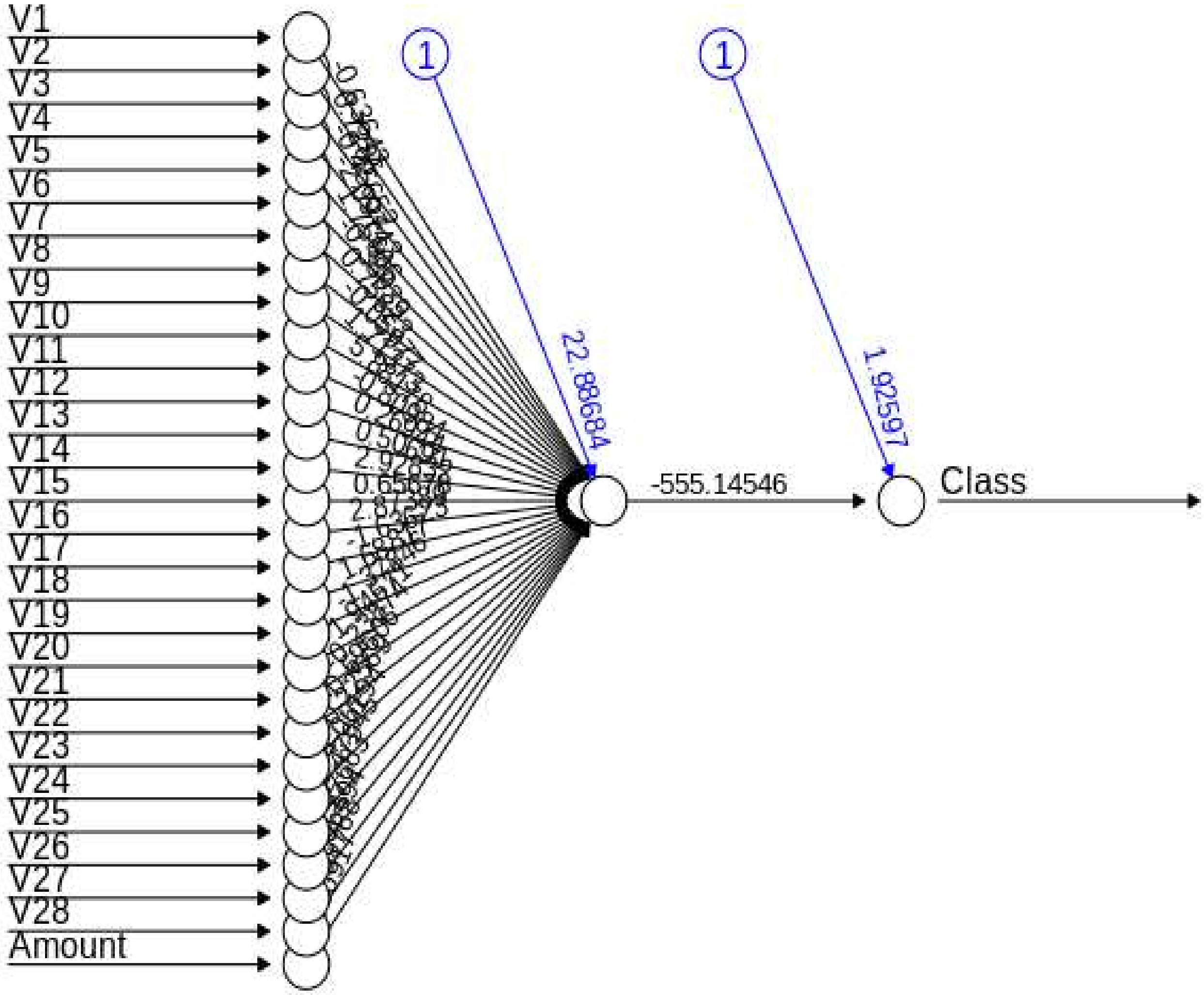
```
library(rpart)
library(rpart.plot)
decisionTree_model <- rpart(Class ~ . , creditcard, method = 'class')
predicted_val <- predict(decisionTree_model, creditcard, type = 'class')
probability <- predict(decisionTree_model, creditcard, type = 'prob')
rpart.plot(decisionTree_model)
```



7. Artificial Neural Network

Artificial Neural Networks are a type of machine learning algorithm that are 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. We implement this as follows –

```
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)
```



8. Gradient Boosting (GBM)

__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. We will implement gradient descent algorithm in our model as follows –

```
library(gbm, quietly=TRUE)

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

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

```
## Loaded gbm 2.1.5
```

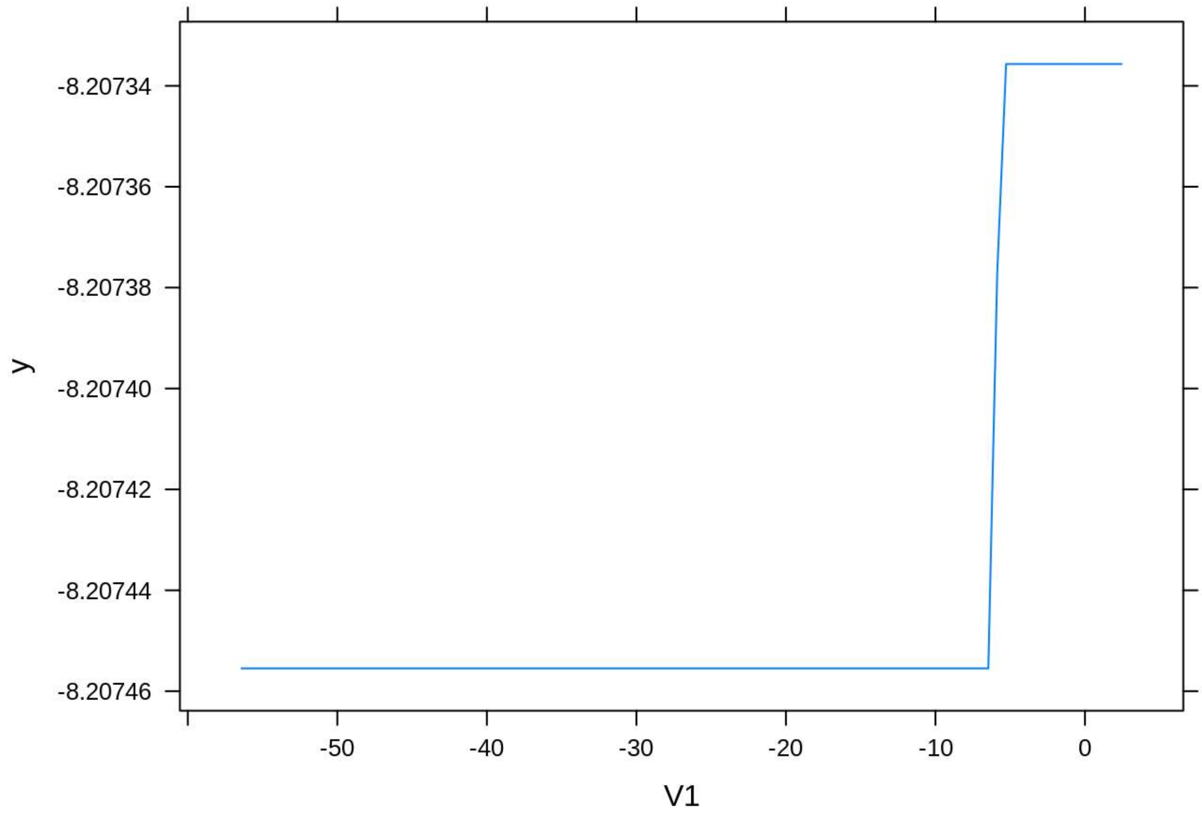
```r
# Get the time to train the GBM model
system.time(
    model_gbm <- gbm(Class ~ .
        , distribution = "bernoulli"
        , data = rbind(train_data, test_data)
        , n.trees = 500
        , interaction.depth = 3
        , n.minobsinnode = 100
        , shrinkage = 0.01
        , bag.fraction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

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

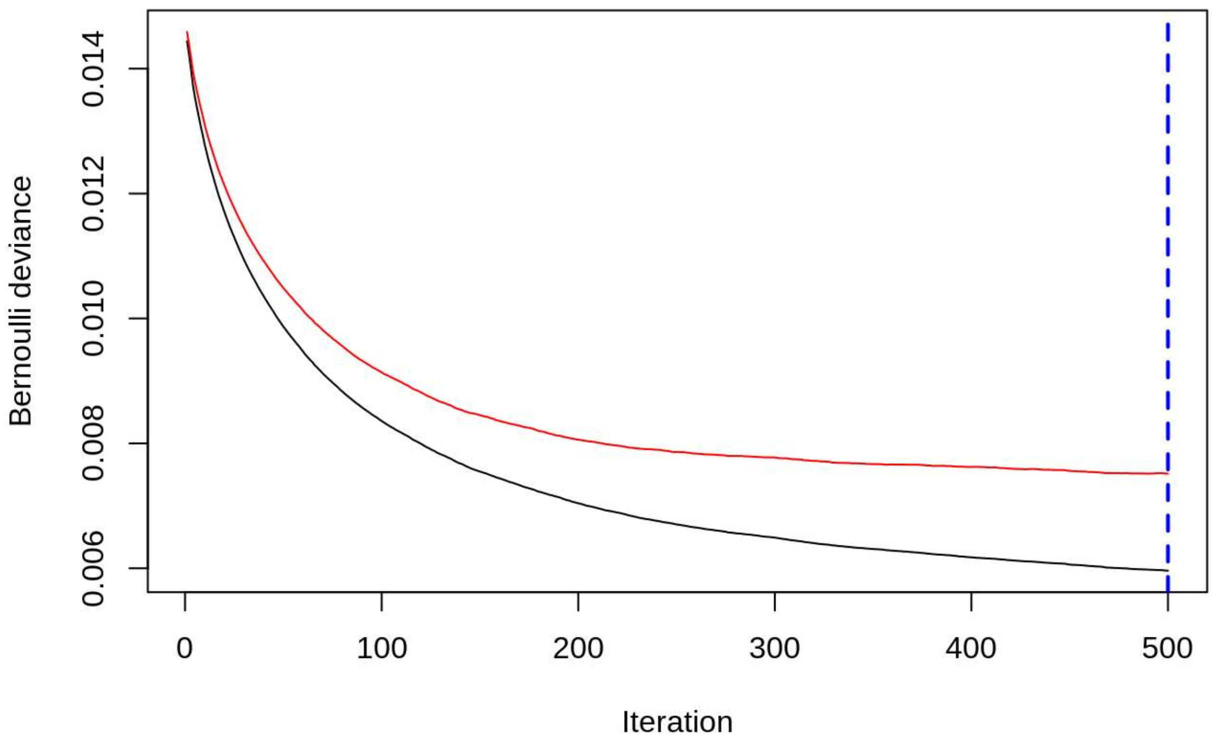
```r
# Determine best iteration based on test data
gbm.iter = gbm.perf(model_gbm, method = "test")
```

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

plot(model_gbm)



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



0.014
0.012
0.010
0.008
0.006
Bernoulli deviance
0
100
200
300
400
500
Iteration

# 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")

```
# 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
```

print(gbm_auc)

```
print(gbm_auc)
```

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