HarvardX: PH125.9x Data Science Professional Certificate Program

Choose Your Own Project

Abraham Mateos

September 19, 2020

Index

1 Overview......**1**
1.1 Introduction......2
1.2 Aim of the project......2
1.3 Dataset......3

2 Project 1 (first approach)......**4**
2.1 Data post-processing......4
2.2 User defined functions......7
2.3 Data exploration......12
2.4 Correlation......13
2.5 Model......14
2.6 Data visualization......15
2.7 Prediction......16
2.8 Conclusion Project 1......18

3 Project 2 (second approach)......**19**
3.1 Data exploration......19
3.2 Data wrangling......22
3.3 Data modeling......23
3.4 Logistic Regression Model......25
3.5 Decision Tree model method......29
3.6 Artificial Neural Network......29
3.7 Gradient Boosting Model......31
3.8 Conclusion Project 2......33

4 Resume......**33**

1 Overview

This project is the solution for the "Choose Your Own" Project requirement, the second part of the Module 9 - Capstone course, inside the HarvardX Data Science Professional Certificate Program.
The main target is to devise a credit card fraud detection system, based on a machine learning algorithm, which has been approached by two methods below.

Nowadays, we are absolutely exposed to possible fraud in the Internet through e-commerce and online purchases. For this reason, several international institutions, such as the International Monetary Fund, have set up policy procedures to ensure and ease up bank card fraud detection through online account machine learning code.

So, in sum, this project may be a sample / resume of these proceeds carried out by the main world organizations.

For this task, I have used all the techniques and resources learnt throughout all this program materials and courses.

1.1 Introduction

The credit card fraud detection systems might be one of the main systems any banking entity sould have established inside its own software structure.

Under certain recent American and British universities research analysis, fraud is one of the major ethical issues in the credit card industry. The main aims are, firstly, to identify the different types of credit card fraud, and, in second place, to review alternative techniques that have been employed in fraud detection.

The secondary target is to present, compare and analyze recently published discoveries in credit card fraud detection. This article defines common terms in credit card fraud and highlights key statistics and figures in this field. Depending on the type of fraud banks or credit card firms might face, several measures should be adopted and implemented. The proposals made in multiple documents are likely to have beneficial results and perks in terms of cost savings and time efficiency.

The relevance of the application of the techniques reviewed here strikes on shrinking credit card fraud crimes volume. However, there are still ethical issues when genuine credit card clients are misclassified as fraudulent.

1.2 Aim of the project

The target in this project is to devise a machine learning algorithm approached by two ways or methods.
The first one is based on the computation of 2D coordinates, which have been obtained running the t-SNE function. Afterwards, the coordinates are merged and then we can plot the results into hexagonal figures to distribute fraud commitment percentages.

Later, we calculate ROC, AUC and the cost function, in order to set up users features as variables. Since we get the matrix where all in the info is disordered, I have opted in this site to carry out data exploration and then create correlations among the variables chosen and users features, and training a final model into this training set (evaluation into a validation set, as taught by HarvardX staff in the previous project).

The second one, the second approach to this problem, is based on a quite different approach, since I have devised the linear regression model and then opted by the decision tree method. Later, there is a quite unique feature I hope will be welcome by the staff, the artificial neural network. This is a tool used generally to create the links among different features and variables straight forward into a machine learning training set, and quite easy to visualize. So then, the result is a gradient boosting machine learning model to train, under the Bernouilli distribution of fraud / not fraud, as in the previous approach.

And finally, after getting the final model to work through iterations, we plot the final model and come up with the AUC using the own GBM. It's quite different, as you may appreciate.

1.3 Dataset

I have used a dataset everyone can easy download at the link provided, as in the EdX patform as at my GitHub repository.

You may find out two different links which drive you to two different datasets, but actually the main dataset used is the first one from Kaggle. The second one is just a dataset I have uploaded to Drive and is based on the first one from Kaggle; its size is somehow smaller for I have deleted some variables I thought they were not relevant to my analysis, such as car data- house data- other properties data related to a person's credit card. We here are absolutely focused on the data related to credit cards and people features.
Nevertheless, hereby I share both of them again, for any problem you may search out.

https://www.kaggle.com/mlg-ulb/creditcardfraud/download

https://drive.google.com/file/d/1CTAlmlREFRaEN3NoHHitewpqAtWS5cVQ/view

2 PROJECT 1 (first approach)

First of all, I have decided to use in first place the SNE, for it is a great tool in order to gather a great amount of high-dimensional data in a low representation space, since we want to carry out non-linear dimension reduction.

We run t-SNE* (Stochastic Neighbor Embedding) to get the 2D coordinates:

```
rtsne_out <- Rtsne(as.matrix(select(data_sub, -id)),

                   pca = FALSE,

                   verbose = TRUE,

                   theta = 0.3,

                   max_iter = 1300,

                   Y_init = NULL)
```

("Class", the target, is not used to compute the 2D coordinates)

* Mathematically, given a set of N high-dimensional objects `$x_1, \ldots, x_N$', t-SNE first computes probabilities p_{ij} that are proportional to the similarity of objects x_i and x_j , as follows:

For $i \neq j$, define

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum\limits_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

And set $p_{i,i} = 0$. Note that $\sum p_{j,i} = 1$.

2.1 Data post-processing

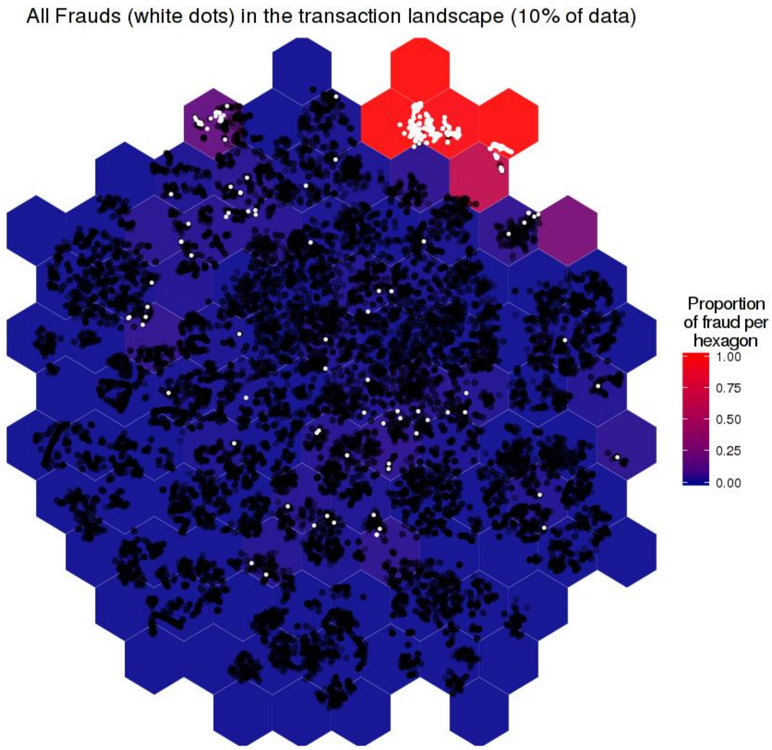
Now we proceed to merge 2D coordinates with original features:

```
tsne_coord <- as.data.frame(rtsne_out$Y) %>%

  cbind(select(data_sub, id)) %>%

  left_join(data, by = 'id')
```

We plot the map and its hexagonal figure background, due to its optimal node distribution:

```
gg <- ggplot() +

  labs(title = "All Frauds (white dots) in the transaction landscape (10% of
data)") +

  scale_fill_gradient(low = 'darkblue',

                      high = 'red',

                      name="Proportion\nof fraud per\nhexagon") +

  coord_fixed(ratio = 1) +

  theme_void() +

  stat_summary_hex(data = tsne_coord,

                   aes(x = V1, y = V2, z = Class),

                   bins=10,

                   fun = mean,

                   alpha = 0.9) +

  geom_point(data = filter(tsne_coord, Class == 0),

             aes(x = V1, y = V2),

             alpha = 0.3,

             size = 1,

             col = 'black') +

  geom_point(data = filter(tsne_coord, Class == 1),

             aes(x = V1, y = V2),

             alpha = 0.9,

             size = 0.3,

             col = 'white') +

  theme(plot.title = element_text(hjust = 0.5,

                                  family = 'Calibri'),

        legend.title.align = 0.5)
```

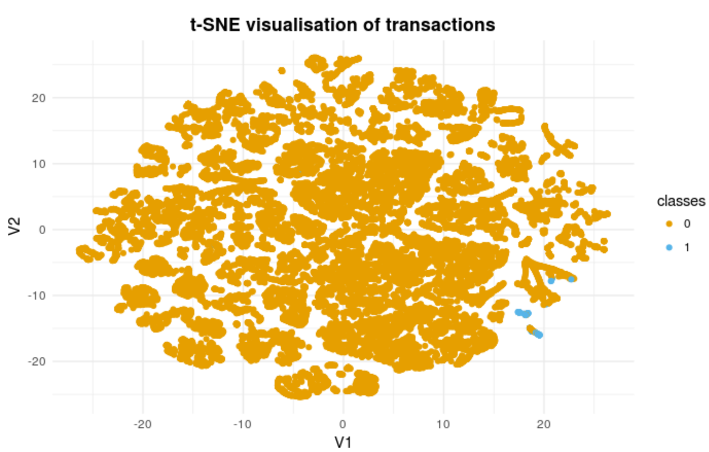
(On about 10% of the data)



The hexagons show the local density of fraudulent transactions (white points).
Red colors mean high density of fraud (typically > 75% of points included in the hexagon) whereas blue colors are associated with a small fraction of fraud.

Tip: To carry out just the t-SNE visualization of transactions, we may just write

```
classes <- as.factor(data$Class[tsne_subset])

tsne_mat <- as.data.frame(tsne$Y)

ggplot(tsne_mat, aes(x = V1, y = V2)) + geom_point(aes(color = classes)) +
theme_minimal() + common_theme + ggtitle("t-SNE visualisation of transactions") +
scale_color_manual(values = c("#E69F00", "#56B4E9"))
```



2.2 User defined functions

We calculate now the ROC (Receiver Optimistic Characteristics):

```
calculate_roc <- function(verset, cost_of_fp, cost_of_fn, n=100) {


  tp <- function(verset, threshold) {

    sum(verset$predicted >= threshold & verset$Class == 1)

  }


  fp <- function(verset, threshold) {

    sum(verset$predicted >= threshold & verset$Class == 0)

  }


  tn <- function(verset, threshold) {

    sum(verset$predicted < threshold & verset$Class == 0)

  }


  fn <- function(verset, threshold) {

    sum(verset$predicted < threshold & verset$Class == 1)

  }


  tpr <- function(verset, threshold) {

    sum(verset$predicted >= threshold & verset$Class == 1) / sum(verset$Class == 1)

  }


  fpr <- function(verset, threshold) {

    sum(verset$predicted >= threshold & verset$Class == 0) / sum(verset$Class == 0)

  }
```

```
  cost <- function(verset, threshold, cost_of_fp, cost_of_fn) {

    sum(verset$predicted >= threshold & verset$Class == 0) * cost_of_fp +

      sum(verset$predicted < threshold & verset$Class == 1) * cost_of_fn

  }

  fpr <- function(verset, threshold) {

    sum(verset$predicted >= threshold & verset$Class == 0) / sum(verset$Class == 0)

  }


  threshold_round <- function(value, threshold)

  {

    return (as.integer(!(value < threshold)))

  }
```

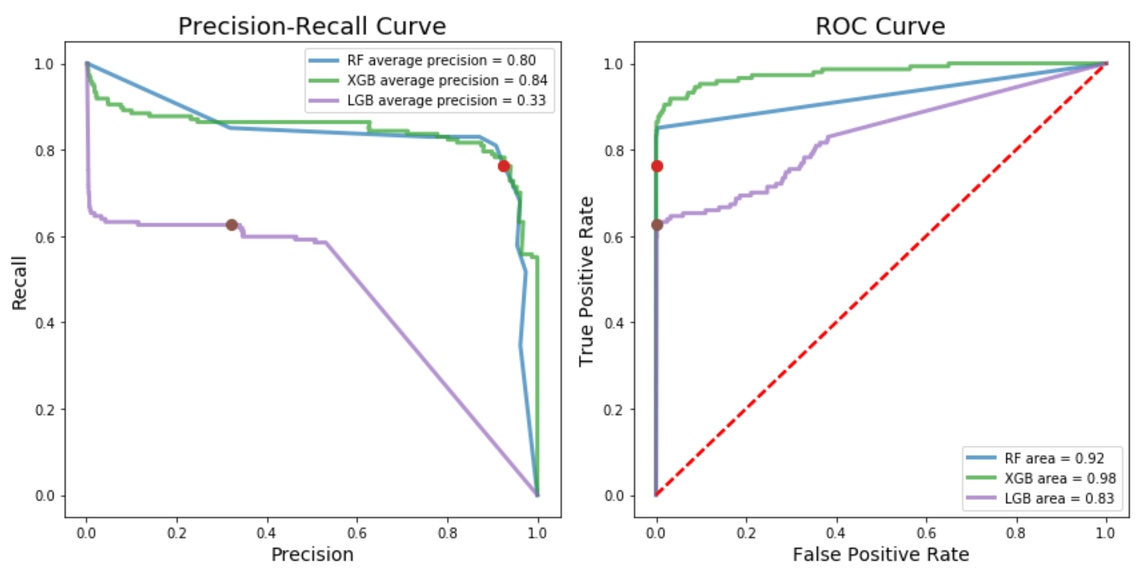
And then, to calculate the AUC, we proceed to apply iterations through spply with all the previous variables:

```
  auc_ <- function(verset, threshold) {

    auc(verset$Class, threshold_round(verset$predicted,threshold))

  }

  roc <- data.frame(threshold = seq(0,1,length.out=n), tpr=NA, fpr=NA)

  roc$tp <- sapply(roc$threshold, function(th) tp(verset, th))

  roc$fp <- sapply(roc$threshold, function(th) fp(verset, th))

  roc$tn <- sapply(roc$threshold, function(th) tn(verset, th))

  roc$fn <- sapply(roc$threshold, function(th) fn(verset, th))

  roc$tpr <- sapply(roc$threshold, function(th) tpr(verset, th))

  roc$fpr <- sapply(roc$threshold, function(th) fpr(verset, th))

  roc$cost <- sapply(roc$threshold, function(th) cost(verset, th, cost_of_fp,
cost_of_fn))

  roc$auc <-  sapply(roc$threshold, function(th) auc_(verset, th))


return(roc)}
```

Graphical representation for ROC, AUC (Area Under Curve) and cost function related to the users features definition:

```
plot_roc <- function(roc, threshold, cost_of_fp, cost_of_fn) {
  library(gridExtra)

  norm_vec <- function(v) (v - min(v))/diff(range(v))

  idx_threshold = which.min(abs(roc$threshold-threshold))

  col_ramp <- colorRampPalette(c("green", "orange", "red", "black"))(100)

  col_by_cost <- col_ramp[ceiling(norm_vec(roc$cost) * 99) + 1]

  p_roc <- ggplot(roc, aes(fpr, tpr)) +
    geom_line(color = rgb(0, 0, 1, alpha = 0.3)) +
    geom_point(color = col_by_cost,
               size = 2,
               alpha = 0.5) +
    labs(title = sprintf("ROC")) + xlab("FPR") + ylab("TPR") +
    geom_hline(yintercept = roc[idx_threshold, "tpr"],
               alpha = 0.5,
               linetype = "dashed") +
    geom_vline(xintercept = roc[idx_threshold, "fpr"],
               alpha = 0.5,
               linetype = "dashed")

  p_auc <- ggplot(roc, aes(threshold, auc)) +
    geom_line(color = rgb(0, 0, 1, alpha = 0.3)) +
    geom_point(color = col_by_cost,
```

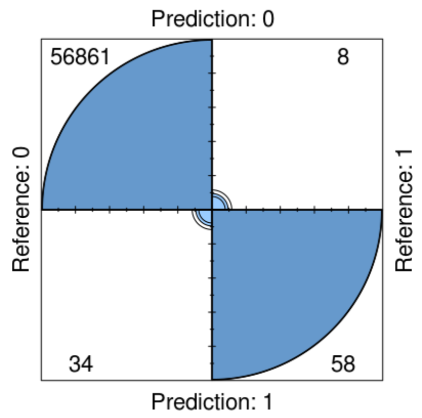
```
              size = 2,
              alpha = 0.5) +
  labs(title = sprintf("AUC")) +
  geom_vline(xintercept = threshold,
             alpha = 0.5,
             linetype = "dashed")

p_cost <- ggplot(roc, aes(threshold, cost)) +
  geom_line(color = rgb(0, 0, 1, alpha = 0.3)) +
  geom_point(color = col_by_cost,
             size = 2,
             alpha = 0.5) +
  labs(title = sprintf("cost function")) +
  geom_vline(xintercept = threshold,
             alpha = 0.5,
             linetype = "dashed")

sub_title <- sprintf("threshold at %.2f - cost of FP = %d, cost of FN = %d", threshold, cost_of_fp, cost_of_fn)
```



```
  grid.arrange(

    p_roc,

    p_auc,

    p_cost,

    ncol = 2,

    sub = textGrob(sub_title, gp = gpar(cex = 1), just = "bottom")
})
```

Function and plot in order to demonstrate or handle out the ‘confusion matrix’ as a result:

```
plot_confusion_matrix <- function(verset, sSubtitle) {

  tst <- data.frame(round(verset$predicted,0), verset$Class)

  opts <-  c("Predicted", "True")

  names(tst) <- opts

  cf <- plyr::count(tst)

  cf[opts][cf[opts]==0] <- "Not Fraud"

  cf[opts][cf[opts]==1] <- "Fraud"


  ggplot(data =  cf, mapping = aes(x = True, y = Predicted)) +

    labs(title = "Confusion matrix", subtitle = sSubtitle) +

    geom_tile(aes(fill = freq), colour = "grey") +

    geom_text(aes(label = sprintf("%1.0f", freq)), vjust = 1) +

    scale_fill_gradient(low = "lightblue", high = "blue") +

    theme_bw() +

    theme(legend.position = "none")


}
```



2.3 Data exploration

Exploring the data though columns, rows, summary and table formats

```
nrow(credit_data)

ncol(credit_data)

summary(credit_data)

str(credit_data)

head(credit_data, 10) %>%

  kable( "html",

         escape=F,

         align="c") %>%

  kable_styling(bootstrap_options = "striped",

                full_width = F,

                position = "center")


boxplot(credit_data$Amount)

hist(credit_data$Amount)
```

There are totally 31 columns in the data. One column, `Class` is the target value; it is a binary value, can have either `0` (not fraud) or `1` (fraud) value. Another two columns have clear meaning: `Amount` is the amount of the transaction; `Time` is the time of the transaction. The rest of the features (28), anonymized, are named from `V1` to `V28`.

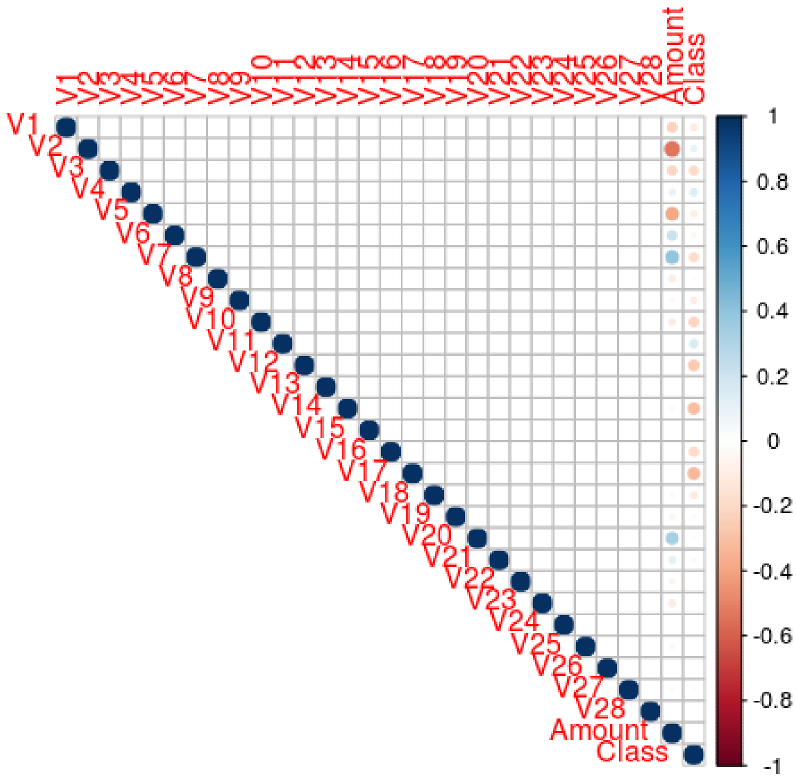
The data is highly unbalanced with respect of `Class` variable values. There are only ``r nrow(credit_data[credit_data$Class==1,])/nrow(credit_data)*100`` % of the rows with value `Class = 1`.

Typically, in such cases, we can either choose to preserve the data unbalancing or use a oversampling (of the data with minority value of target variable) or undersampling (of the data with majority value of the target variable).

Here we will just preserve the unbalancing of the data. In terms of validation of the result, we will see that usual matrix, using a confusion matrix or accuracy are not the most relevant and will be preferred alternative solutions using AUC.

2.4 Correlations

```
correlations <- cor(credit_data, method = "pearson")
corrplot(
  correlations,
  number.cex = .9,
  method = "circle",
  type = "full",
  tl.cex = 0.8,
  tl.col = "black"
)
```



We can observe that most of the data features are not correlated. This is because before publishing, most of the features were presented to a Principal Component Analysis (PCA) algorithm.
The features `V1` to `V28` are most probably the Principal Components resulted after propagating the real features through PCA. We do not know if the numbering of the features reflects the importance of the Principal Components. This information might be checked partially using the Variable Importance from Random Forest.

2.5 Model

After we split the data in a training and test set, we create the RF model using the training set.

```
nrows <- nrow(credit_data)

set.seed(314)

indexT <- sample(1:nrow(credit_data), 0.7 * nrows)
```

For the future work, as always, we separate the train and validation sets:

```
trainset = credit_data[indexT, ]

verset = credit_data[-indexT, ]
```

And we insert our training set inside the Random Forest model as follows:

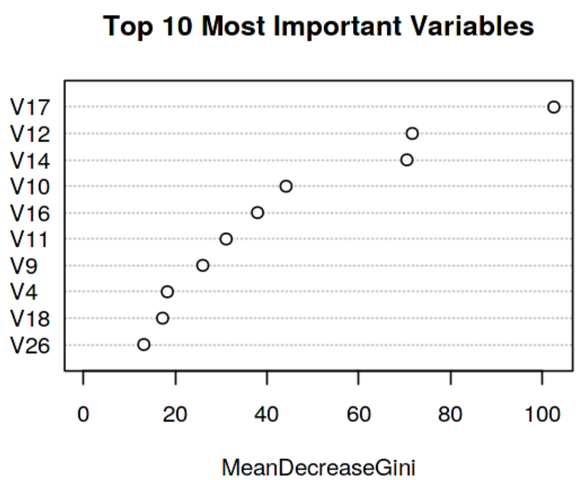
```
n <- names(trainset)

rf.form <- as.formula(paste("Class ~", paste(n[!n %in% "Class"], collapse = " +
")))


trainset.rf <- randomForest(rf.form,

                            trainset,

                            ntree = 100,

                            importance = T)
```

2.6 Data visualization

Now, let's visualize the variable importance arranging a grid by plotting, coordinating and structuring all the data in three main "variables" (vi):

```
varimp <- data.frame(trainset.rf$importance)



vi1 <- ggplot(varimp, aes(x = reorder(rownames(varimp), IncNodePurity), y =
IncNodePurity)) +

  geom_bar(stat = "identity",

           fill = "tomato",

           colour = "black") +

  coord_flip() +

  theme_bw(base_size = 8) +

  labs(title = "Prediction using RandomForest with 100 trees",

       subtitle = "Variable importance (IncNodePurity)",

       x = "Variable",

       y = "Variable importance (IncNodePurity)")


vi2 <- ggplot(varimp, aes(x = reorder(rownames(varimp), X.IncMSE), y = X.IncMSE)) +

  geom_bar(stat = "identity",

           fill = "lightblue",
```

```
          colour = "black") +
  coord_flip() + theme_bw(base_size = 8) +
  labs(title = "Prediction using RandomForest with 100 trees",
       subtitle = "Variable importance (%IncMSE)",
       x = "Variable",
       y = "Variable importance (%IncMSE)")
grid.arrange(vi1, vi2, ncol = 2)
```



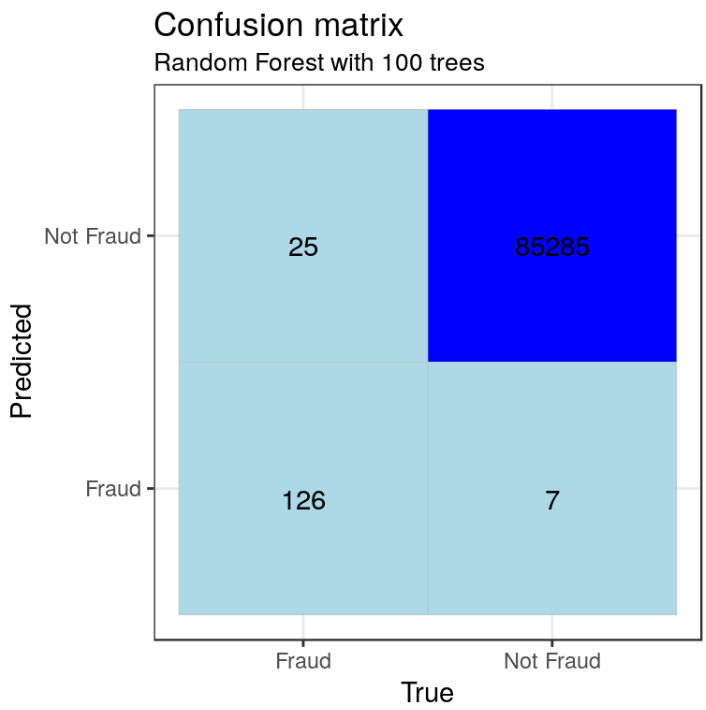
2.7 Prediction

Let's use the model we have just trained for predictions on the Fraud/Not Fraud Class for our test set.

```
verset$predicted <- predict(trainset.rf ,verset)
```

For the threshold at 0.5, let's represent the Confusion matrix.

```
plot_confusion_matrix(verset, "Random Forest with 100 trees")
```



For such a problem, where the number of TP is very small in comparison with the number of TN, the Confusion Matrix is less useful; it is important to use a metric that include evaluation of FP and FN as well.

It is important to minimize as much as possible the number of FN (Predicted: Not Fraud and True: Fraud) since their cost could be very large. Tipically AUC is used for such cases.

Let's calculate the TP, FP, TN, FN, ROC, AUC and cost for threshold with values between 0 and 1 (100 values equaly distributed) and cost 1 for TN and 10 for FN.

```
roc <- calculate_roc(verset, 1, 10, n = 100)


mincost <- min(roc$cost)

roc %>%

  mutate(auc = ifelse(cost == mincost,

                      cell_spec(sprintf("%.5f", auc),

                                "html",

                                color = "green",

                                background = "lightblue",

                                bold = T),

                      cell_spec(sprintf("%.5f", auc),
```

```
                                    "html",
                                    color = "black",
                                    bold = F))) %>%
  kable("html", escape = F, align = "c") %>%
  kable_styling(bootstrap_options = "striped",
                full_width = F,
                position = "center") %>%
```

Finally, we plot the ROC, AUC and cost functions for a ref. threshold of 0.3.

```
threshold = 0.3
renderPlot({
d<-get(input$roc, threshold, 1, 10)
plot(d)
})
```

2.8 Conclusion Project 1

The calculated accuracy is not very relevant in the conditions where there is a very large unbalance between the number of `fraud` and `non-fraud` events in the dataset. In such cases, we can see a very large accuracy.
More relevant is the value of ROC-AUC (Area Under Prevision-Recall Curve for the Receiver Operator Characteristic). The value obtained (0.93) is relatively good, considering that we did not performed any tuning, working with default RandomForest algorithm parameters.

3 PROJECT 2 (second approach)

3.1 Data exploration

We start by carrying out data exploration / cleaning, through the functions learnt through this program courses, as always:

```
dim(creditcard_data)

head(creditcard_data, 6)
```

```
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
```

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

```
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
```

table(creditcard_data$Class)

```
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
```

summary(creditcard_data$Amount)

```
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
```

names(creditcard_data)


```
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
```

```
var(creditcard_data$Amount)

sd(creditcard_data$Amount)
```

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

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

3.2 Data wrangling

Later, we proceed to data wrangling techniques, using mainly head and scale functions which will let me scale or grow up the chosen amount to a more realistic sample to analyze:

```
head(creditcard_data)
```

```
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
```

```
creditcard_data$Amount=scale(creditcard_data$Amount)

NewData=creditcard_data[,-c(1)]

head(NewData)
```

```
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
```

3.3 Data modeling

We now model the data, so that we can come up with the data sample, and therefore, the train (test) set and the test (validation) set, as follows:

```
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)
```

```r
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
```

```r
dim(test_data)
```

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

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

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

```r
 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
```

dim(test_data)


```r
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
```

```r
dim(test_data)
```

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

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

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

```r
 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
```

3.4 Logistic Regression model

We proceed to carry out the logistic regression model, making use of the class and test data and the binomial distribution specification.
We then use the library proper for the ROC feature, and later we make the prediction and include this into our validation set (test set) and its visualization with roc function.

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

```
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
```

We summarize this model through ‘summary(Logistic_Model)’ as follows:

```
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
```

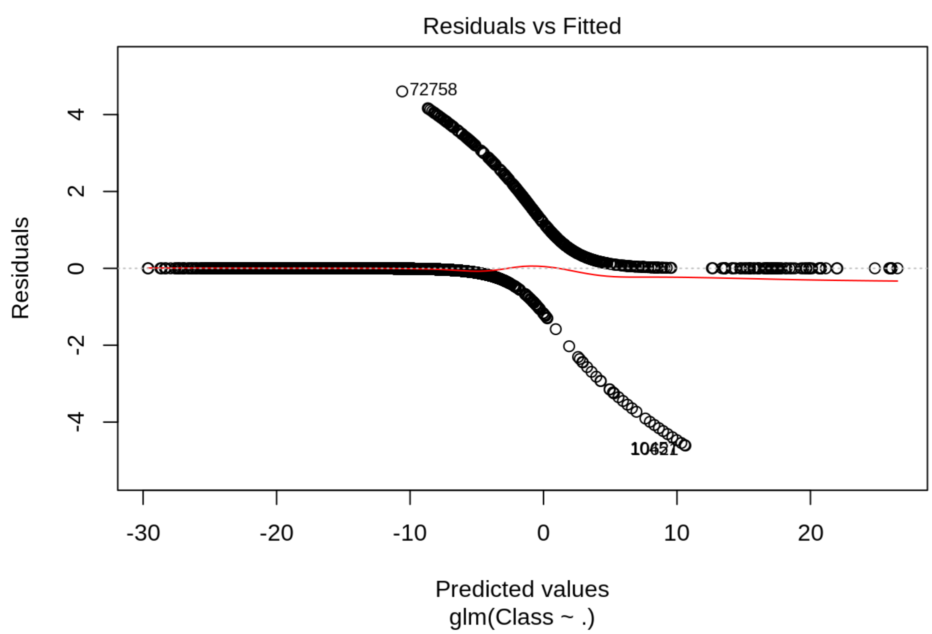
```
  Logistic_Model=glm(Class~.,test_data,family=binomial())
```

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

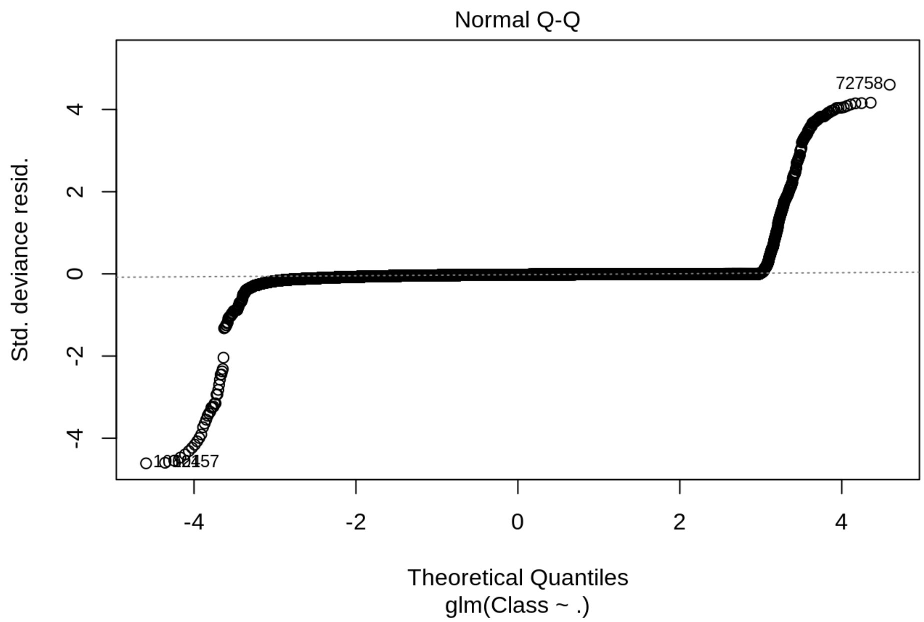
```
  summary(Logistic_Model)
```

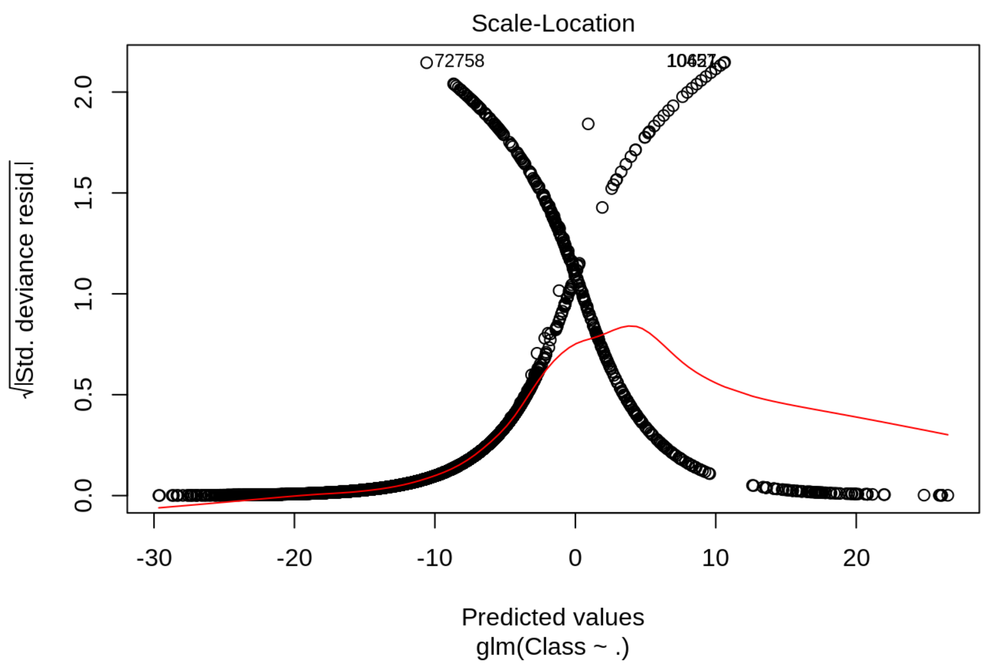
```
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = test_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.9019  -0.0254  -0.0156  -0.0078   4.0877
```

And then we get the following plots:

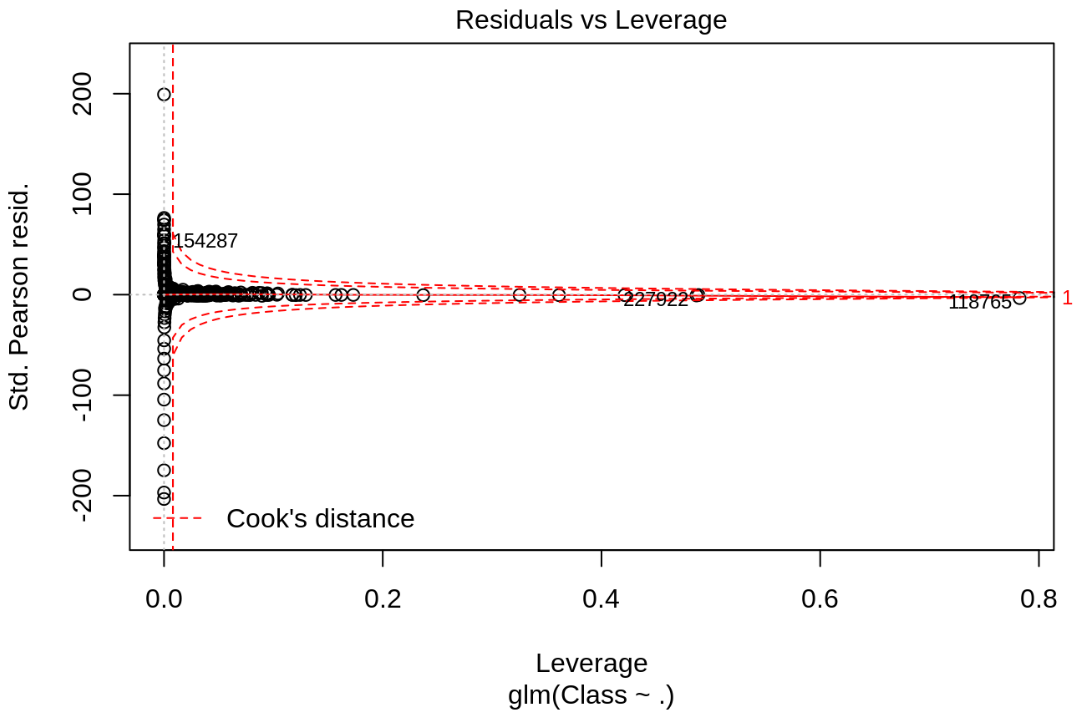


We can appreciate easily the fitted probabilities occurred among values 0 – 1 by the red line.



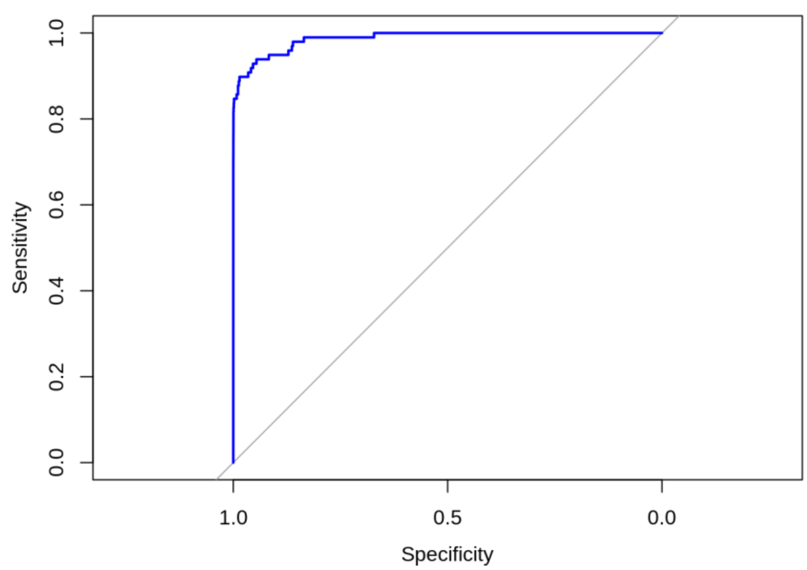


The squared root of the standard deviance residuals lets us appreciate the point where our predicted values through our logistic model, depending on the scale-location analyzed, gets matched (crossed) and how it may be continued in progress.



Assessing the performance of our model will imply delineate or define the ROC curve:

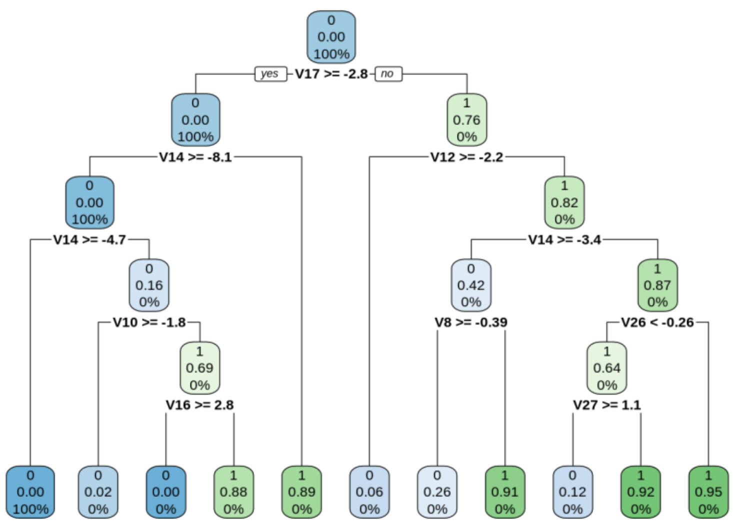
```
library(pROC)

lr.predict <- predict(Logistic_Model,train_data, probability = TRUE)

auc.gbm = roc(test_data$Class, lr.predict, plot = TRUE, col = "blue")
```



3.5 Decision Tree Model

We use again the Decision Tree model method, where, as we have been taught through the program, we can partition the credit card dataset, and solve the linear regression, as we are managing continuous input and output data.

```
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)
```



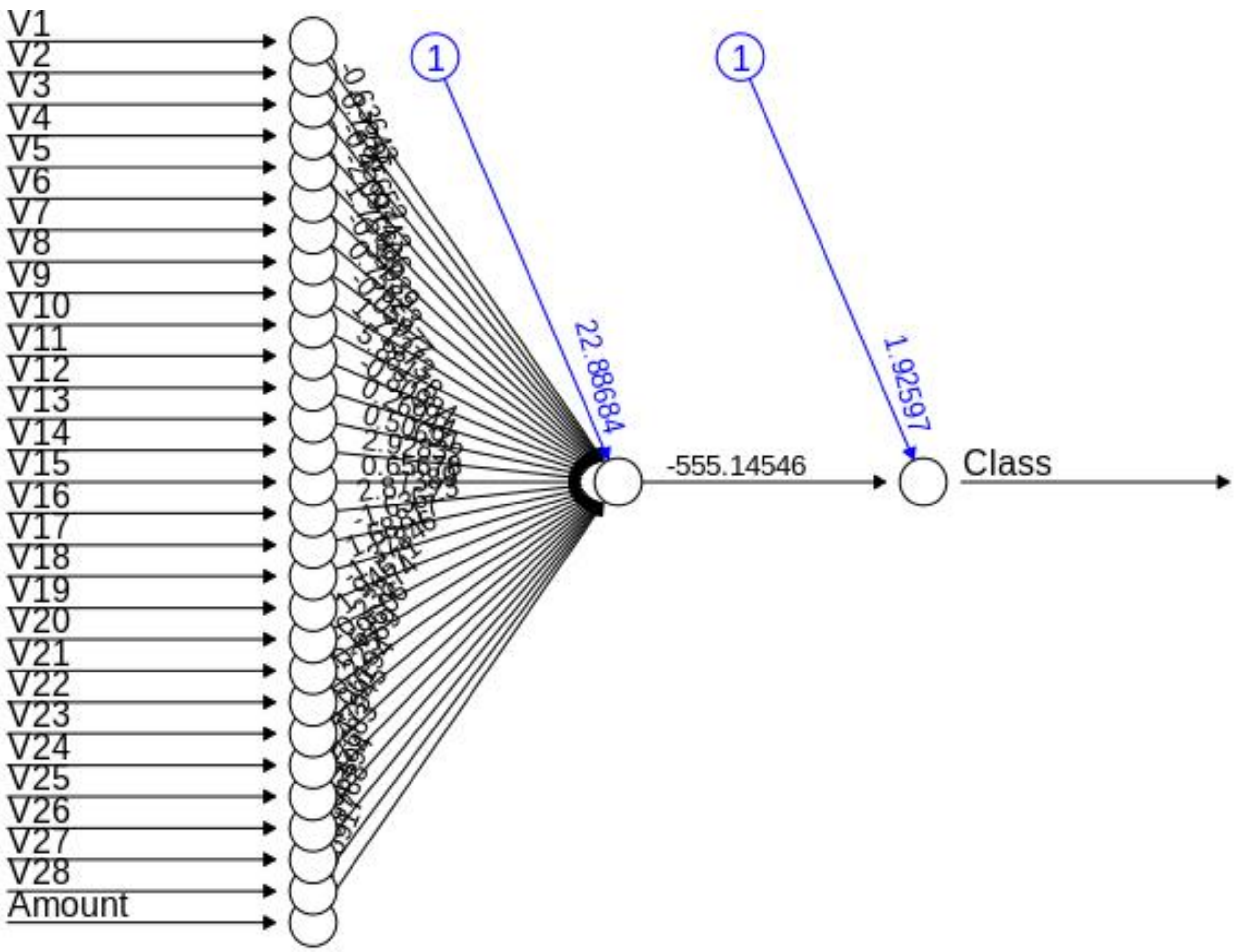
3.6 Artificial Neural Network

Now, we get to the Artificial Neural Network stage, where we should analyze our test set (training set) into a neural model, in order to create a result which would fit the human mind. I have achieved this point by not reading the data into a linear description, but in a network way, linking and making sense in info nods with other data in the same variable. Later, that training result is set in our validation set (test set), giving away a result in a “default case” among 0.5 and 1. So,

```
library(neuralnet)

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

plot(ANN_model)
```

And then, we take up the prediction computation on the devised ANN_model

```
predANN=compute(ANN_model,test_data)

resultANN=predANN$net.result

resultANN=ifelse(resultANN>0.5,1,0)
```



3.7 Gradient boosting

We proceed to take up gradient boosting, downloading the proper library.

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

```
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")
```

Training the Gradient Boosting machine model, we can apply different values to the different parameters settled down into our model, as follows:

```
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))
                       )
  )
```

```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")
```

Then, we try to define the best iteration based on the validation data set:

```
gbm.iter = gbm.perf(model_gbm, method = "test")

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

In the end, we set up the plot of the gbm model, to later calculate AUC on the test (validation set) data.

Tip:

```
renderPlot({

d<-get(input$model_gbm)

plot(d)

})
```

This is just a *Shiny render function*, in order to represent an interactive motion of the chart depicted by the code below:

```
gbm_test = predict(model_gbm, newdata = test_data, n.trees = gbm.iter)
```

```
# 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
```

```
gbm_auc = roc(test_data$Class, gbm_test, plot = TRUE, col = "red")

print(gbm_auc)
```


```
# 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
```


3.8 Conclusion Project 2

As we can appreciate for this second approach, we have settled down the same principles but have arranged the tree model method to then create an artificial neural network in order to connect human mind features to credit cards data, gathering different data into each "variable classification". To sum up, we carry out gradient boosting, in order to apply different values to the different parameters included in our final training model, to achieve then the best iteration into our evaluation (test) set.
PD: As you may observe as well, I have included some Shiny functions in order to create an interactive explanation of both projects. Sincerely, I hope it will also work for you without any problems.

4 Resume

Ending up this whole project, we can state that credit card fraud detection is a crucial point to take into account, in order to set up policies and security measures in every private firm or public institution.
As we have appreciated, these are two ways of so many others figured out to solve this so important and common issue in our society. Detection systems sometimes may create default messages due to several reasons: not accurate code configuration, rough software configuration, Wi-Fi connection at portable systems such as mobile phone or Laptops... But focusing on our work, one of the main problems of AI/machine learning (and quite more closely related to these projects) is selecting a confident and secure dataset. This is essential (along with the right connection to the respective URL or file inside our R or Python workspace), for it will be the correct start of our coding, and therefore, our program will be based on realistic and safe sources and will help in a better way to people.