Credit Card Fraud Detection

Introduction

The subject of the project is fraud detection in transactions with credit cards. Detecting fraud can save lots of money and prevent loses. To do that two different methods are utilized. The first one is Recursive Partitioning and the second approach is Random Forest.

creditcardfraud dataset is used in this project which is a public dataset and can be downloaded by the following link:

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

Following libraries are used in this project:

```r
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
if(!require(tidyverse))  install.packages("tidyverse")
```

```
## Loading required package: tidyverse

## Warning: package 'tidyverse' was built under R version 3.6.3

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used

## Warning: package 'tibble' was built under R version 3.6.3

## Warning: package 'tidyr' was built under R version 3.6.3

## Warning: package 'readr' was built under R version 3.6.3

## Warning: package 'purrr' was built under R version 3.6.3

## Warning: package 'dplyr' was built under R version 3.6.3
## Warning: package 'forcats' was built under R version 3.6.3

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(randomForest))  install.packages("randomForest")
```

```
## Loading required package: randomForest

## Warning: package 'randomForest' was built under R version 3.6.3

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
if(!require(imbalance))  install.packages("imbalance")
```

```
## Loading required package: imbalance

## Warning: package 'imbalance' was built under R version 3.6.3
```

```r
if(!require(caret))  install.packages("caret")
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.6.3

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.6.3

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(e1071))  install.packages("e1071")
```

```
## Loading required package: e1071

## Warning: package 'e1071' was built under R version 3.6.3
```

```r
if(!require(Metrics))  install.packages("Metrics")
```

```
## Loading required package: Metrics

## Warning: package 'Metrics' was built under R version 3.6.3

##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
library(tidyverse)
library(randomForest)
library(imbalance)
library(caret)
library(e1071)
library(Metrics)
```

Preparing the data

From the given link the dataset in form of csv file is downloaded (*creditcard.csv*) and saved in the same folder as the script file and .rmd file. Now we can use the following piece of code to have a look at the dataset.

```r
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2194655 117.3    4000280 213.7  4000280 213.7
## Vcells 3672761  28.1    8388608  64.0  5989392  45.7
```

```r
credit_card_data <- read.table(file = "creditcard.csv", sep = ",", header=TRUE)
glimpse(credit_card_data)
```

```
## Rows: 284,807
## Columns: 31
## $ Time   <dbl> 0, 0, 1, 1, 2, 2, 4, 7, 7, 9, 10, 10, 10, 11, 12, 12, 12, 13, 1~
## $ V1     <dbl> -1.3598071, 1.1918571, -1.3583541, -0.9662717, -1.1582331, -0.4~
## $ V2     <dbl> -0.07278117, 0.26615071, -1.34016307, -0.18522601, 0.87773675, ~
## $ V3     <dbl> 2.53634674, 0.16648011, 1.77320934, 1.79299334, 1.54871785, 1.1~
## $ V4     <dbl> 1.37815522, 0.44815408, 0.37977959, -0.86329128, 0.40303393, -0~
## $ V5     <dbl> -0.33832077, 0.06001765, -0.50319813, -0.01030888, -0.40719338,~
## $ V6     <dbl> 0.46238778, -0.08236081, 1.80049938, 1.24720317, 0.09592146, -0~
## $ V7     <dbl> 0.239598554, -0.078802983, 0.791460956, 0.237608940, 0.59294074~
## $ V8     <dbl> 0.098697901, 0.085101655, 0.247675787, 0.377435875, -0.27053267~
## $ V9     <dbl> 0.3637870, -0.2554251, -1.5146543, -1.3870241, 0.8177393, -0.56~
## $ V10    <dbl> 0.09079417, -0.16697441, 0.20764287, -0.05495192, 0.75307443, -~
## $ V11    <dbl> -0.55159953, 1.61272666, 0.62450146, -0.22648726, -0.82284288, ~
## $ V12    <dbl> -0.61780086, 1.06523531, 0.06608369, 0.17822823, 0.53819555, 0.~
## $ V13    <dbl> -0.99138985, 0.48909502, 0.71729273, 0.50775687, 1.34585159, -0~
## $ V14    <dbl> -0.31116935, -0.14377230, -0.16594592, -0.28792375, -1.11966983~
## $ V15    <dbl> 1.468176972, 0.635558093, 2.345864949, -0.631418118, 0.17512113~
## $ V16    <dbl> -0.47040053, 0.46391704, -2.89008319, -1.05964725, -0.45144918,~
## $ V17    <dbl> 0.207971242, -0.114804663, 1.109969379, -0.684092786, -0.237033~
## $ V18    <dbl> 0.02579058, -0.18336127, -0.12135931, 1.96577500, -0.03819479, ~
## $ V19    <dbl> 0.40399296, -0.14578304, -2.26185710, -1.23262197, 0.80348692, ~
## $ V20    <dbl> 0.25141210, -0.06908314, 0.52497973, -0.20803778, 0.40854236, 0~
## $ V21    <dbl> -0.018306778, -0.225775248, 0.247998153, -0.108300452, -0.00943~
## $ V22    <dbl> 0.277837576, -0.638671953, 0.771679402, 0.005273597, 0.79827849~
## $ V23    <dbl> -0.110473910, 0.101288021, 0.909412262, -0.190320519, -0.137458~
## $ V24    <dbl> 0.06692807, -0.33984648, -0.68928096, -1.17557533, 0.14126698, ~
## $ V25    <dbl> 0.12853936, 0.16717040, -0.32764183, 0.64737603, -0.20600959, -~
## $ V26    <dbl> -0.18911484, 0.12589453, -0.13909657, -0.22192884, 0.50229222, ~
## $ V27    <dbl> 0.133558377, -0.008983099, -0.055352794, 0.062722849, 0.2194222~
## $ V28    <dbl> -0.021053053, 0.014724169, -0.059751841, 0.061457629, 0.2151531~
## $ Amount <dbl> 149.62, 2.69, 378.66, 123.50, 69.99, 3.67, 4.99, 40.80, 93.20, ~
## $ Class  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

As we can see there are in total 31 columns in the dataset. The time is not considered to be relevant to our analysis. *Class* shows whether the transaction is a fraud (Class = 1) or a non-fraud (Class = 0). We might guess that the number of observations which belong to Class 0 is by far more than those which belong to Class 1. We can verify this by the following code.

```r
n_fraud_not_fraud <- credit_card_data %>% count(Class)
print(n_fraud_not_fraud)
```

```
##   Class      n
## 1     0 284315
## 2     1    492
```

Since time column is not relevant to our model, we remove it from the dataset. Now we have 29 features(*V1-V28* and *Amount*) and the goal is to predict the *Class* of the transaction.

As we see, fraud transactions are only a tiny fraction (0.17%) of all observations. This means, if we predict all transaction are non-fraud, we still have a accuracy of 99.8% which obviously is not correct.

To solve this issue we need to *balance* the dataset. To do that there are two techniques. The first one is to undersample the dominant class (Class 0) and the second one is to oversample the class which has less members (Class 1). Here, we use *pdfos* function from *imbalance* library to do oversampling.

```r
credit_card_data_noTime <- subset (credit_card_data, select = -Time)
new_fraud_data <- pdfos(credit_card_data_noTime, numInstances =
                          n_fraud_not_fraud$n[1] - n_fraud_not_fraud$n[2])
new_credit_card_data <- rbind(credit_card_data_noTime, new_fraud_data)
new_n_fraud_not_fraud <- new_credit_card_data %>% count(Class)
rm(credit_card_data, credit_card_data_noTime, new_fraud_data)
glimpse(new_credit_card_data)
```

```
## Rows: 568,630
## Columns: 30
## $ V1     <dbl> -1.3598071, 1.1918571, -1.3583541, -0.9662717, -1.1582331, -0.4~
## $ V2     <dbl> -0.07278117, 0.26615071, -1.34016307, -0.18522601, 0.87773675, ~
## $ V3     <dbl> 2.53634674, 0.16648011, 1.77320934, 1.79299334, 1.54871785, 1.1~
## $ V4     <dbl> 1.37815522, 0.44815408, 0.37977959, -0.86329128, 0.40303393, -0~
## $ V5     <dbl> -0.33832077, 0.06001765, -0.50319813, -0.01030888, -0.40719338,~
## $ V6     <dbl> 0.46238778, -0.08236081, 1.80049938, 1.24720317, 0.09592146, -0~
## $ V7     <dbl> 0.239598554, -0.078802983, 0.791460956, 0.237608940, 0.59294074~
## $ V8     <dbl> 0.098697901, 0.085101655, 0.247675787, 0.377435875, -0.27053267~
## $ V9     <dbl> 0.3637870, -0.2554251, -1.5146543, -1.3870241, 0.8177393, -0.56~
## $ V10    <dbl> 0.09079417, -0.16697441, 0.20764287, -0.05495192, 0.75307443, -~
## $ V11    <dbl> -0.55159953, 1.61272666, 0.62450146, -0.22648726, -0.82284288, ~
## $ V12    <dbl> -0.61780086, 1.06523531, 0.06608369, 0.17822823, 0.53819555, 0.~
## $ V13    <dbl> -0.99138985, 0.48909502, 0.71729273, 0.50775687, 1.34585159, -0~
## $ V14    <dbl> -0.31116935, -0.14377230, -0.16594592, -0.28792375, -1.11966983~
## $ V15    <dbl> 1.468176972, 0.635558093, 2.345864949, -0.631418118, 0.17512113~
## $ V16    <dbl> -0.47040053, 0.46391704, -2.89008319, -1.05964725, -0.45144918,~
## $ V17    <dbl> 0.207971242, -0.114804663, 1.109969379, -0.684092786, -0.237033~
## $ V18    <dbl> 0.02579058, -0.18336127, -0.12135931, 1.96577500, -0.03819479, ~
## $ V19    <dbl> 0.40399296, -0.14578304, -2.26185710, -1.23262197, 0.80348692, ~
## $ V20    <dbl> 0.25141210, -0.06908314, 0.52497973, -0.20803778, 0.40854236, 0~
## $ V21    <dbl> -0.018306778, -0.225775248, 0.247998153, -0.108300452, -0.00943~
## $ V22    <dbl> 0.277837576, -0.638671953, 0.771679402, 0.005273597, 0.79827849~
## $ V23    <dbl> -0.110473910, 0.101288021, 0.909412262, -0.190320519, -0.137458~
## $ V24    <dbl> 0.06692807, -0.33984648, -0.68928096, -1.17557533, 0.14126698, ~
## $ V25    <dbl> 0.12853936, 0.16717040, -0.32764183, 0.64737603, -0.20600959, -~
## $ V26    <dbl> -0.18911484, 0.12589453, -0.13909657, -0.22192884, 0.50229222, ~
## $ V27    <dbl> 0.133558377, -0.008983099, -0.055352794, 0.062722849, 0.2194222~
## $ V28    <dbl> -0.021053053, 0.014724169, -0.059751841, 0.061457629, 0.2151531~
## $ Amount <dbl> 149.62, 2.69, 378.66, 123.50, 69.99, 3.67, 4.99, 40.80, 93.20, ~
## $ Class  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

As we see, the number of rows in each class is the same in this new dataset. We also need to convert *Class* column to factor type.

```
new_credit_card_data$Class <- factor(new_credit_card_data$Class)
```

Now we have a dataset which we can use for training and testing our machine learning models. Training and testing datasets are created by following lines of codes.

```
test_index <- createDataPartition(y = new_credit_card_data$Class,
                                  times = 1, p = 0.1, list = FALSE)
training_data <- new_credit_card_data[-test_index,]
test_data <- new_credit_card_data[test_index,]
rm(new_credit_card_data)
```

Methods

Two different models are used in this project. The first one uses *Recursive Partitioning* and the second approach uses Random Forest.

Recursive Partitioning

In this approach we use *rpart* library.

```
if(!require(rpart))  install.packages("rpart")
```

```
## Loading required package: rpart
```

```
## Warning: package 'rpart' was built under R version 3.6.3
```

```
library(rpart)
```

We want to predict *Class* by using the given 29 features.

```
fit <- rpart(Class~., data = training_data, method = 'class')
```

We can visualize our model by *fancyRpartPlot* function as shown in the following code.

```
if(!require(rattle))  install.packages("rattle")
```

```
## Loading required package: rattle
```

```
## Warning: package 'rattle' was built under R version 3.6.3
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

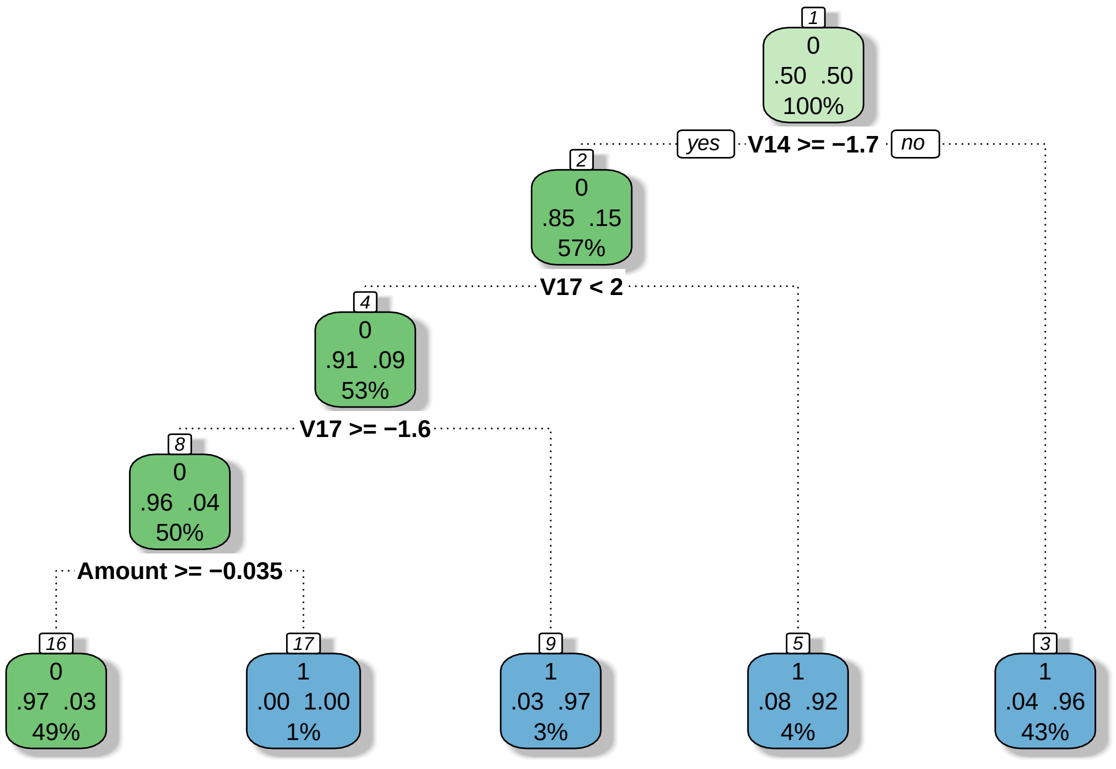
```
if(!require(rpart.plot))  install.packages("rpart.plot")
```

```
## Loading required package: rpart.plot
```

```
if(!require(RColorBrewer))  install.packages("RColorBrewer")
```

```
## Loading required package: RColorBrewer
```

```
library(rattle)
library(rpart.plot)
library(RColorBrewer)
fancyRpartPlot(fit, caption = NULL)
```



To see how our model perform, we do prediction on the test data as follows. The *Confusion Matrix* is used to evaluate the predicted results.

```
pred_DT <- predict(object= fit, newdata = test_data, type = 'class')
conf_DT <- confusionMatrix(pred_DT,test_data[['Class']])
overall_accuracy <- conf_DT$overall[['Accuracy']]
sensitivity <- conf_DT$byClass[['Sensitivity']]
specificity <- conf_DT$byClass[['Specificity']]

print(conf_DT)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 27303   799
##          1  1129 27633
##
##                Accuracy : 0.9661
##                  95% CI : (0.9646, 0.9676)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9322
##
##  Mcnemar's Test P-Value : 6.744e-14
```

```
##
##             Sensitivity : 0.9603
##             Specificity : 0.9719
##          Pos Pred Value : 0.9716
##          Neg Pred Value : 0.9607
##              Prevalence : 0.5000
##          Detection Rate : 0.4801
##    Detection Prevalence : 0.4942
##       Balanced Accuracy : 0.9661
##
##        'Positive' Class : 0
##
```

```
print(paste('Overall accuracy: ',overall_accuracy))
```

```
## [1] "Overall accuracy:  0.966094541361846"
```

```
print(paste('Sensitivity: ',sensitivity))
```

```
## [1] "Sensitivity:  0.960291221159257"
```

```
print(paste('Specificity: ',specificity))
```

```
## [1] "Specificity:  0.971897861564434"
```

As we can see the overall accuracy, sensitivity and specificity are 96.61%, 0.9603 and 0.9719 respectively.

Random Forest

Another approach for this classification problem is the random forest. This method is an ensemble learning method. In this approach, several decision trees are created. These trees are used to build up the forest. It can achieve higher accuracy than a single decision tree. To do that, it uses feature randomness and bootstrap aggregating (also know as bagging). Here, *randomForest* function from *randomForest* library is used to create a random forest. We can set the number of randomly sampled variables by *mtry* and the number of trees by *ntree*. First we run the code for 6 different *mtry*.

```
mtry_i <- c(1:6,rep(3,4))
ntree_i <- c(rep(50,6),seq(10,70,by = 20))
sensitivity_i <- numeric(10)
specificity_i <- numeric(10)
overall_accuracy <- numeric(10)
for (i in 1:6){
gc()
rf_classifier <- randomForest(formula = Class ~ ., data =
                                 training_data,ntree=ntree_i[i],
                               mtry= mtry_i[i], importance = TRUE)
pred <- predict(rf_classifier, test_data)
conf <- confusionMatrix(pred,test_data[['Class']])
overall_accuracy[i] <- conf$overall[['Accuracy']]
sensitivity_i[i] <- conf$byClass[['Sensitivity']]
specificity_i[i] <- conf$byClass[['Specificity']]
```

```r
number_of_trees <- 1:ntree_i[i]
}

tab_1 <- data.frame(mtry = mtry_i[1:6],ntree= rep(50,6),
                    overall_accuracy = overall_accuracy[1:6],
                    Sensitivity = sensitivity_i[1:6]
                    , Specificity = specificity_i[1:6])
print(tab_1)
```

```
##   mtry ntree overall_accuracy Sensitivity Specificity
## 1    1    50        0.9983821   0.9980304   0.9987338
## 2    2    50        0.9988042   0.9987338   0.9988745
## 3    3    50        0.9988218   0.9987690   0.9988745
## 4    4    50        0.9989273   0.9990504   0.9988042
## 5    5    50        0.9987690   0.9988745   0.9986635
## 6    6    50        0.9988569   0.9988745   0.9988393
```

From the above table, we can see the best result is obtained when *mtry* is equal to 3. Now we set *mtry* to 3 and change the number of trees (*ntree*). We choose *ntree* to be 10,30,50 and 70.

```r
gc()
```

```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   2941051 157.1    7447146  397.8  10808368  577.3
## Vcells 30018189 229.1  331238251 2527.2 379778216 2897.5
```

```r
for (i in 7:10){
gc()
rf_classifier <- randomForest(formula = Class ~ ., data =
                                  training_data,ntree=ntree_i[i],
                              mtry= mtry_i[i], importance = TRUE)
pred <- predict(rf_classifier, test_data)
conf <- confusionMatrix(pred,test_data[['Class']])
overall_accuracy[i] <- conf$overall[['Accuracy']]
sensitivity_i[i] <- conf$byClass[['Sensitivity']]
specificity_i[i] <- conf$byClass[['Specificity']]
number_of_trees <- 1:ntree_i[i]
}
```

```r
tab_2 <- data.frame(mtry = mtry_i,ntree= ntree_i,
                    overall_accuracy = overall_accuracy,
                    Sensitivity = sensitivity_i, Specificity = specificity_i)

print(tab_2)
```

```
##    mtry ntree overall_accuracy Sensitivity Specificity
## 1     1    50        0.9983821   0.9980304   0.9987338
## 2     2    50        0.9988042   0.9987338   0.9988745
## 3     3    50        0.9988218   0.9987690   0.9988745
## 4     4    50        0.9989273   0.9990504   0.9988042
## 5     5    50        0.9987690   0.9988745   0.9986635
## 6     6    50        0.9988569   0.9988745   0.9988393
## 7     3    10        0.9981711   0.9980304   0.9983118
## 8     3    30        0.9986986   0.9987338   0.9986635
## 9     3    50        0.9988218   0.9986986   0.9989449
## 10    3    70        0.9987866   0.9987690   0.9988042
```

```r
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2940940 157.1    7447146  397.8  10808368  577.3
## Vcells 30835596 235.3  460622672 3514.3 466975468 3562.8
```

All the results are presented in the above table. The row number 9 gives the best results. In this row we have 50 *trees* in the forest and *mtry* is 3.

Conclusion

Recursive partitioning and random forest models are used in this project to detect frauds in the credit cards transactions. Overall accuracy, sensitivity and specificity are chosen to assess the performance of the models.

Although both models give high accuracy results, the random forest is a better alternative when the parameters are set appropriately. The best model is a random forest with 50 trees and *mtry* equal tree. In this case, overall accuracy, sensitivity and specificity are 99.88%, 0.9989 and 0.9987 respectively.