ADA 442 - Project Report

Credit Card Fraud Detection

Meltem Akkoca - Berkay Dursun

2023-01-06

Contents

1 Introduction

Explain your motivation for selecting this project.

- Describe the problem you investigate in your project.
    - The project we chose is called Credit Cart Fraud Detection because in these days, credit card fraud is a significant issue in the financial sector. Yearly loss of millions of money results from fraudulent card transactions.
- State the problem
    - In fact, if we need to describe our problem in general around this issue, we can call it as trying to find a way to reduce money losses by increasing fraud detection with machine learning models and strategies. However, due to the extreme imbalance of these data, it seemed quite difficult to design and implement these models. In addition, due to the hidden data that is not open to the public, which we will talk about in our project, it has become much more difficult to determine a strategy and continues to arouse curiosity.
- State the research objectives that you want to accomplish
    - The aim of this project is to implement strategies that can successfully predict fraudulent transactions as given in our data, work in harmony with the sampling and modeling techniques most suitable for our data due to the imbalance of our data set, and give the closest approach to the truth.

2 Methodology

Briefly describe the statistical modeling you will employ to analyze the data.

- Why it is suitable for your design?
- The main equations and properties can be summarized before going further on modeling.
    - While developing the project, we started with the models that we thought were simpler, and continued with the models that looked more complex and that we thought might yield better results. When we were surprised by the result at the end of the project, it helped us understand that the complexity of the model was not directly proportional to the accuracy of the model.
    - We started out with Decision Tree model. Then we tried Logistic Regression, Random Forest and then we also tried XG Boost, which is based on Gradient Boosted Trees.

```
#Importing Libraries

library(dplyr) # for data manipulation
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(stringr) # for data manipulation
library(caret) # for sampling
```

```
## Zorunlu paket yükleniyor: ggplot2
```

```
## Zorunlu paket yükleniyor: lattice
```

```r
library(caTools) # for train/test split
library(ggplot2) # for data visualization
library(corrplot) # for correlations
```

```
## corrplot 0.92 loaded
```

```r
library(ROSE)# for ROSE sampling
```

```
## Loaded ROSE 0.0-4
```

```r
library(rpart)# for decision tree model
library(Rborist)# for random forest model
```

```
## Rborist 0.3-2
```

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

```r
library(xgboost) # for xgboost model
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(data.table) # for data manipulation
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
library(plyr) # for data manipulation
```

```
## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```r
library(pROC) # for ROC analyzes
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(glmnet) # for fit regression models
```

```
## Zorunlu paket yükleniyor: Matrix

## Loaded glmnet 4.1-6
```

3 Data Set

-Describe the data set you used in the analysis.

- We got our dataset from Kaggle and imported it. Due to the privacy and security of the data, we cannot provide detailed information on what the features are, but what we do know is that due to the PCA transformation, the features only contain numeric input variables. Our features are Time, Amount, Class and features from V1 to V28. -Time: Expresses the time in seconds between two transactions. -Amount: refers to the transaction amount. -Class: Response variable and takes the value 1 for Fraud and 0 for Non-Fraud.

```r
# load data from csv file
data = read.csv('creditcard.csv')

head(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         V24
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -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
```

```r
summary(data)
```

```
##       Time              V1                  V2                  V3
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573   Min.   :-48.3256
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855   1st Qu.: -0.8904
##  Median : 84692   Median :  0.01811   Median :  0.06549   Median :  0.1799
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.0000
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372   3rd Qu.:  1.0272
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :  9.3826
##        V4                 V5                   V6                  V7
##  Min.   :-5.68317   Min.   :-113.74331   Min.   :-26.1605   Min.   :-43.5572
##  1st Qu.:-0.84864   1st Qu.:  -0.69160   1st Qu.: -0.7683   1st Qu.: -0.5541
```

```
##  Median :-0.01985   Median :  -0.05434   Median : -0.2742   Median :  0.0401
##  Mean   : 0.00000   Mean   :   0.00000   Mean   :  0.0000   Mean   :  0.0000
##  3rd Qu.: 0.74334   3rd Qu.:   0.61193   3rd Qu.:  0.3986   3rd Qu.:  0.5704
##  Max.   :16.87534   Max.   :  34.80167   Max.   : 73.3016   Max.   :120.5895
##        V8                  V9                 V10                 V11
##  Min.   :-73.21672   Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.20863   1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median :  0.02236   Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.32735   3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 20.00721   Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                V13                V14                V15
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143   Min.   :-4.49894
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256   1st Qu.:-0.58288
##  Median :  0.1400   Median :-0.01357   Median :  0.0506   Median : 0.04807
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000   Mean   : 0.00000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931   3rd Qu.: 0.64882
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268   Max.   : 8.87774
##       V16                 V17                 V18
##  Min.   :-14.12985   Min.   :-25.16280   Min.   :-9.498746
##  1st Qu.: -0.46804   1st Qu.: -0.48375   1st Qu.:-0.498850
##  Median :  0.06641   Median : -0.06568   Median :-0.003636
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.000000
##  3rd Qu.:  0.52330   3rd Qu.:  0.39968   3rd Qu.: 0.500807
##  Max.   : 17.31511   Max.   :  9.25353   Max.   : 5.041069
##       V19                 V20                 V21
##  Min.   :-7.213527   Min.   :-54.49772   Min.   :-34.83038
##  1st Qu.:-0.456299   1st Qu.: -0.21172   1st Qu.: -0.22839
##  Median : 0.003735   Median : -0.06248   Median : -0.02945
##  Mean   : 0.000000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.458949   3rd Qu.:  0.13304   3rd Qu.:  0.18638
##  Max.   : 5.591971   Max.   : 39.42090   Max.   : 27.20284
##       V22                  V23                 V24
##  Min.   :-10.933144   Min.   :-44.80774   Min.   :-2.83663
##  1st Qu.: -0.542350   1st Qu.: -0.16185   1st Qu.:-0.35459
##  Median :  0.006782   Median : -0.01119   Median : 0.04098
##  Mean   :  0.000000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.528554   3rd Qu.:  0.14764   3rd Qu.: 0.43953
##  Max.   : 10.503090   Max.   : 22.52841   Max.   : 4.58455
##       V25                 V26                V27
##  Min.   :-10.29540   Min.   :-2.60455   Min.   :-22.565679
##  1st Qu.: -0.31715   1st Qu.:-0.32698   1st Qu.: -0.070840
##  Median :  0.01659   Median :-0.05214   Median :  0.001342
##  Mean   :  0.00000   Mean   : 0.00000   Mean   :  0.000000
##  3rd Qu.:  0.35072   3rd Qu.: 0.24095   3rd Qu.:  0.091045
##  Max.   :  7.51959   Max.   : 3.51735   Max.   : 31.612198
##       V28                 Amount              Class
##  Min.   :-15.43008   Min.   :    0.00   Min.   :0.000000
##  1st Qu.: -0.05296   1st Qu.:    5.60   1st Qu.:0.000000
##  Median :  0.01124   Median :   22.00   Median :0.000000
##  Mean   :  0.00000   Mean   :   88.35   Mean   :0.001728
##  3rd Qu.:  0.07828   3rd Qu.:   77.17   3rd Qu.:0.000000
##  Max.   : 33.84781   Max.   :25691.16   Max.   :1.000000
```

```r
# We checked that if there is any missing data or not (We can see in results
#none of the variables have missing values)
apply(data, 2, function(x) sum(is.na(x)))
```

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

As a result of the outputs of the above operations, we can see that the mean values of our hidden features are normalized to 0. In addition, when we check if there is missing data, the result is 0, so there is no missing data in the dataset.

Checking for imbalances of the dataset to apply some sampling methods.

```r
# Checking imbalance of class features
table(data$Class)
```

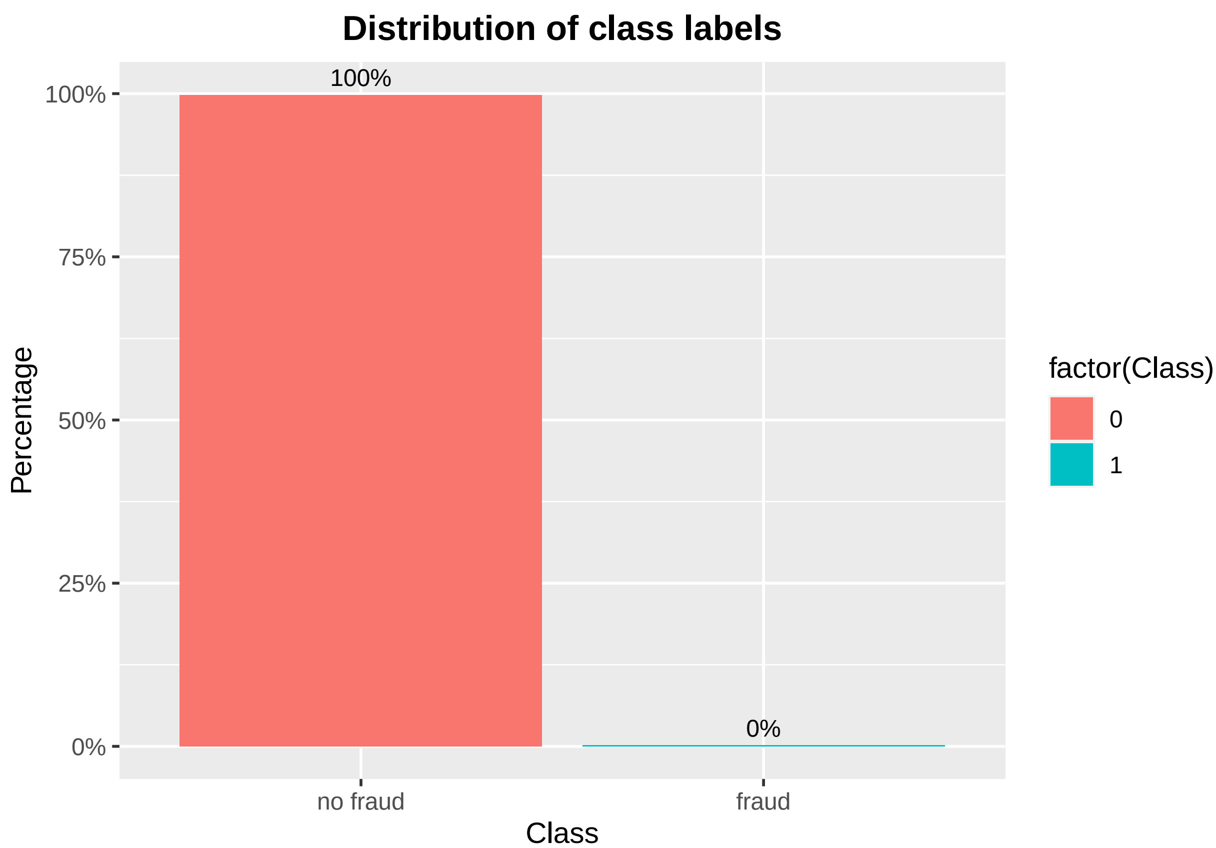
```
## 
##      0      1 
## 284315    492
```

```r
# shows that probability of the imbalance of the class features(non-fraud or fraud )
prop.table(table(data$Class))
```

```
## 
##           0           1 
## 0.998272514 0.001727486
```

```r
common_theme <- theme(plot.title = element_text(hjust = 0.5, face = "bold"))

ggplot(data = data, aes(x = factor(Class),
                        y = prop.table(stat(count)), fill = factor(Class),
                        label = scales::percent(prop.table(stat(count))))) +
    geom_bar(position = "dodge") +
    geom_text(stat = 'count',
              position = position_dodge(.9),
              vjust = -0.5,
              size = 3) +
    scale_x_discrete(labels = c("no fraud", "fraud"))+
    scale_y_continuous(labels = scales::percent)+
    labs(x = 'Class', y = 'Percentage') +
    ggtitle("Distribution of class labels") +
    common_theme
```

```
## Warning: ‘stat(count)‘ was deprecated in ggplot2 3.4.0.
## i Please use ‘after_stat(count)‘ instead.
```



By looking at the table we created above, we can clearly see how big the difference is between the two classes of data (no fraud 0 and fraud 1) and how unevenly the dataset is distributed between these two classes. Even if the data is not complete, we can understand that almost 100% of the data belongs to non-fraud transactions. An accuracy approach that sees non fraud, that is, class=0 as having an accuracy close to 100%, will not be a correct practice as it will create insensitivity to false positives here.

That's why we can transform the data itself with sampling methods.

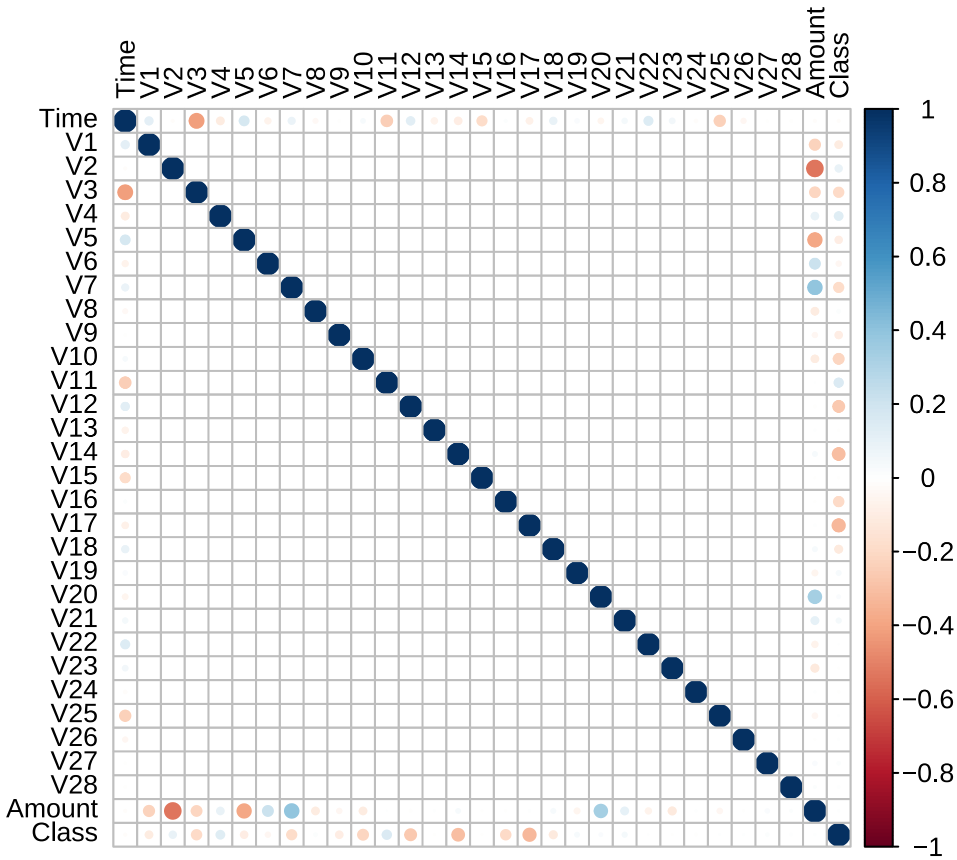
1-)Original data

2-)Up-sampling data

3-)Down-sampling data

4-)ROSE (random over-sampling examples) sampling data

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



As we mentioned before, since our features are confidential, the relationship of these features with each other, that is, the knowledge of how they correlate with each other, has become important for us. So we looked at the correlation of V1-V28s, Time and Amount properties. We concluded that all these features are not very related to each other.

Now, as we explained before, we will apply some sampling methods that we think can make our unbalanced data more balanced, on 4 different models we want to apply, and see which one is more suitable for our data.

4 Data Preparation

```r
#convert all class features to factor

data$Class <- as.factor(data$Class)

#converted names
levels(data$Class) <- c("Not_Fraud", "Fraud")

#Scale numeric variables

data[,-31] <- scale(data[,-31])

head(data)
```

```
##         Time           V1            V2         V3           V4             V5
```

```
## 1 -1.996580 -0.6942411 -0.04407485 1.6727706  0.9733638 -0.245116153
## 2 -1.996580  0.6084953  0.16117564 0.1097969  0.3165224  0.043483276
## 3 -1.996558 -0.6934992 -0.81157640 1.1694664  0.2682308 -0.364571146
## 4 -1.996558 -0.4933240 -0.11216923 1.1825144 -0.6097256 -0.007468867
## 5 -1.996537 -0.5913287  0.53154012 1.0214099  0.2846549 -0.295014918
## 6 -1.996537 -0.2174742  0.58167387 0.7525841 -0.1188331  0.305008424
##            V6         V7          V8         V9         V10        V11
## 1  0.34706734  0.1936786  0.08263713  0.3311272  0.08338540 -0.5404061
## 2 -0.06181986 -0.0637001  0.07125336 -0.2324938 -0.15334936  1.5800001
## 3  1.35145121  0.6397745  0.20737237 -1.3786729  0.19069928  0.6118286
## 4  0.93614819  0.1920703  0.31601704 -1.2625010 -0.05046786 -0.2218912
## 5  0.07199846  0.4793014 -0.22650983  0.7443250  0.69162382 -0.8061452
## 6 -0.02231344  0.3849353  0.21795429 -0.5176177 -0.34110050  1.3140441
##          V12        V13        V14        V15        V16         V17
## 1 -0.6182946 -0.9960972 -0.3246096  1.6040110 -0.5368319  0.24486302
## 2  1.0660867  0.4914173 -0.1499822  0.6943592  0.5294328 -0.13516973
## 3  0.0661365  0.7206986 -0.1731136  2.5629017 -3.2982296  1.30686559
## 4  0.1783707  0.5101678 -0.3003600 -0.6898362 -1.2092939 -0.80544323
## 5  0.5386257  1.3522420 -1.1680315  0.1913231 -0.5152042 -0.27908030
## 6  0.3601815 -0.3597909 -0.1430569  0.5655061  0.4584589 -0.06844494
##           V18         V19         V20         V21          V22         V23
## 1  0.03076988  0.49628116  0.32611744 -0.02492332  0.382853766 -0.17691102
## 2 -0.21876220 -0.17908573 -0.08961071 -0.30737626 -0.880075209  0.16220090
## 3 -0.14478974 -2.77855597  0.68097378  0.33763110  1.063356404  1.45631719
## 4  2.34530040 -1.51420227 -0.26985475 -0.14744304  0.007266895 -0.30477601
## 5 -0.04556892  0.98703556  0.52993786 -0.01283920  1.100009340 -0.22012301
## 6  0.08190778 -0.04077658  0.11021522 -0.28352172 -0.771425648 -0.04227277
##          V24        V25        V26         V27         V28      Amount
## 1  0.1105067  0.2465850 -0.3921697  0.33089104 -0.06378104  0.24496383
## 2 -0.5611296  0.3206933  0.2610690 -0.02225564  0.04460744 -0.34247394
## 3 -1.1380901 -0.6285356 -0.2884462 -0.13713661 -0.18102051  1.16068389
## 4 -1.9410237  1.2419015 -0.4602165  0.15539593  0.18618826  0.14053401
## 5  0.2332497 -0.3952009  1.0416095  0.54361884  0.65181477 -0.07340321
## 6 -0.6132723 -0.4465828  0.2196368  0.62889938  0.24563577 -0.33855582
##       Class
## 1 Not_Fraud
## 2 Not_Fraud
## 3 Not_Fraud
## 4 Not_Fraud
## 5 Not_Fraud
## 6 Not_Fraud
```

```
summary(data)
```

```
##       Time               V1                  V2                 V3
##  Min.   :-1.9966   Min.   :-28.798504   Min.   :-44.03521   Min.   :-31.8717
##  1st Qu.:-0.8552   1st Qu.: -0.469891   1st Qu.: -0.36247   1st Qu.: -0.5872
##  Median :-0.2131   Median :  0.009245   Median :  0.03966   Median :  0.1186
##  Mean   : 0.0000   Mean   :  0.000000   Mean   :  0.00000   Mean   :  0.0000
##  3rd Qu.: 0.9372   3rd Qu.:  0.671693   3rd Qu.:  0.48672   3rd Qu.:  0.6775
##  Max.   : 1.6421   Max.   :  1.253349   Max.   : 13.35773   Max.   :  6.1880
##        V4                V5                  V6                 V7
##  Min.   :-4.01391   Min.   :-82.40795   Min.   :-19.6360   Min.   :-35.20933
##  1st Qu.:-0.59938   1st Qu.: -0.50107   1st Qu.: -0.5767   1st Qu.: -0.44789
```

```
##  Median :-0.01402   Median : -0.03937   Median : -0.2058   Median :  0.03242
##  Mean   : 0.00000   Mean   :  0.00000   Mean   :  0.0000   Mean   :  0.00000
##  3rd Qu.: 0.52501   3rd Qu.:  0.44335   3rd Qu.:  0.2992   3rd Qu.:  0.46111
##  Max.   :11.91872   Max.   : 25.21409   Max.   : 55.0200   Max.   : 97.47807
##        V8                  V9                 V10                 V11
##  Min.   :-61.30242   Min.   :-12.22799   Min.   :-22.58187   Min.   :-4.70012
##  1st Qu.: -0.17468   1st Qu.: -0.58536   1st Qu.: -0.49173   1st Qu.:-0.74702
##  Median :  0.01872   Median : -0.04681   Median : -0.08533   Median :-0.03209
##  Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.27408   3rd Qu.:  0.54353   3rd Qu.:  0.41688   3rd Qu.: 0.72459
##  Max.   : 16.75150   Max.   : 14.19492   Max.   : 21.80754   Max.   :11.77502
##       V12                V13                 V14                 V15
##  Min.   :-18.6986   Min.   :-5.81938   Min.   :-20.04425   Min.   :-4.91518
##  1st Qu.: -0.4059   1st Qu.:-0.65162   1st Qu.: -0.44396   1st Qu.:-0.63681
##  Median :  0.1401   Median :-0.01363   Median :  0.05279   Median : 0.05252
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.6187   3rd Qu.: 0.66565   3rd Qu.:  0.51445   3rd Qu.: 0.70885
##  Max.   :  7.8547   Max.   : 7.16072   Max.   : 10.98145   Max.   : 9.69910
##       V16                 V17                 V18
##  Min.   :-16.12532   Min.   :-29.62640   Min.   :-11.332636
##  1st Qu.: -0.53413   1st Qu.: -0.56956   1st Qu.: -0.595161
##  Median :  0.07579   Median : -0.07733   Median : -0.004338
##  Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.000000
##  3rd Qu.:  0.59720   3rd Qu.:  0.47057   3rd Qu.:  0.597496
##  Max.   : 19.76040   Max.   : 10.89500   Max.   :  6.014331
##       V19                 V20                 V21
##  Min.   :-8.861386   Min.   :-70.69134   Min.   :-47.41898
##  1st Qu.:-0.560536   1st Qu.: -0.27463   1st Qu.: -0.31094
##  Median : 0.004588   Median : -0.08105   Median : -0.04009
##  Mean   : 0.000000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.563792   3rd Qu.:  0.17257   3rd Qu.:  0.25374
##  Max.   : 6.869402   Max.   : 51.13455   Max.   : 37.03465
##       V22                  V23                 V24
##  Min.   :-15.065620   Min.   :-71.75434   Min.   :-4.68363
##  1st Qu.: -0.747346   1st Qu.: -0.25918   1st Qu.:-0.58547
##  Median :  0.009345   Median : -0.01792   Median : 0.06766
##  Mean   :  0.000000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.728335   3rd Qu.:  0.23643   3rd Qu.: 0.72571
##  Max.   : 14.473016   Max.   : 36.07661   Max.   : 7.56967
##       V25                 V26                V27                 V28
##  Min.   :-19.75030   Min.   :-5.4011   Min.   :-55.90650   Min.   :-46.74604
##  1st Qu.: -0.60840   1st Qu.:-0.6781   1st Qu.: -0.17551   1st Qu.: -0.16044
##  Median :  0.03183   Median :-0.1081   Median :  0.00333   Median :  0.03406
##  Mean   :  0.00000   Mean   : 0.0000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.:  0.67280   3rd Qu.: 0.4997   3rd Qu.:  0.22556   3rd Qu.:  0.23715
##  Max.   : 14.42529   Max.   : 7.2940   Max.   : 78.31926   Max.   :102.54324
##      Amount                Class
##  Min.   : -0.35323   Not_Fraud:284315
##  1st Qu.: -0.33084   Fraud    :   492
##  Median : -0.26527
##  Mean   :  0.00000
##  3rd Qu.: -0.04472
##  Max.   :102.36206
```

As we mentioned before, since our features are hidden, we do not know about the unit of each feature, so we used the scale function here to standardize them, that is, to set their mean to 0 and their standard deviation to 1, and to gather us in a single measure about their units.

```
set.seed(123)

split <- sample.split(data$Class, SplitRatio = 0.8)

train <-  subset(data, split == TRUE)

test <- subset(data, split == FALSE)
```

We split the data into test and train data with 20% and 80% ratio.

5 Sampling Techniques

5.1 Down-Sampling

This method helps to reduce the number of observations of the class that has the majority and to balanced the data set.

5.2 Up-Sampling

This method helps to make a trade-off by replicates minority-class observations.
It works with a logic similar to the down-sampling method.

5.3 ROSE (random over-sampling examples)

Instead of replicating and adding the observations from the minority class, it overcome imbalances by generates artificial data. It is also a type of oversampling technique. It uses smoothed bootstrapping to draw artificial samples from the feature space neighbourhood around the minority class.

The original data ratios:

```
# initial class ratio of data
table(train$Class)
```

```
##
## Not_Fraud     Fraud
##    227452       394
```

Data ratio after up-sampling technique is applied:

```
# up_sampling
set.seed(9560)
up_train <- upSample(x = train[, -ncol(train)],
                         y = train$Class)
table(up_train$Class)
```

```
##
## Not_Fraud     Fraud
##    227452    227452
```

Data ratio after down-sampling technique is applied:

```
# down_sampling
set.seed(9560)
down_train <- downSample(x = train[, -ncol(train)],
                         y = train$Class)
table(down_train$Class)
```

```
##
## Not_Fraud     Fraud
##       394       394
```

Data ratio after rose-sampling technique is applied:

```
# rose_sampling
set.seed(9560)
rose_train <- ROSE(Class ~ ., data  = train)$data

table(rose_train$Class)
```

```
##
## Not_Fraud     Fraud
##    114081    113765
```

```
rownames = c("Original ", "Up-sampling ", "Down-sampling ", "Rose-sampling")
colnames = c("Not-Fraud ","Fraud ")
#Define a matrix
matrix <- matrix(cbind(c(227452,394),c(227452,227452),c(394,394),c(114081,113765)), nrow = 4,ncol = 2 ,
print(matrix)
```

```
##                Not-Fraud  Fraud
## Original          227452    394
## Up-sampling       227452 227452
## Down-sampling        394    394
## Rose-sampling     114081 113765
```

6 Models

While evaluating the binary classification algorithm, we used the receiver operating characteristic (ROC) curve, so it would be easier to visually understand the performance of the classifier. As you can see from the graphs below, being closest to the True Positive line actually represents an almost perfect classifier for us. In other words, what we are looking for in this system is to find the ratio with the highest true positive and the lowest false positive ratio.

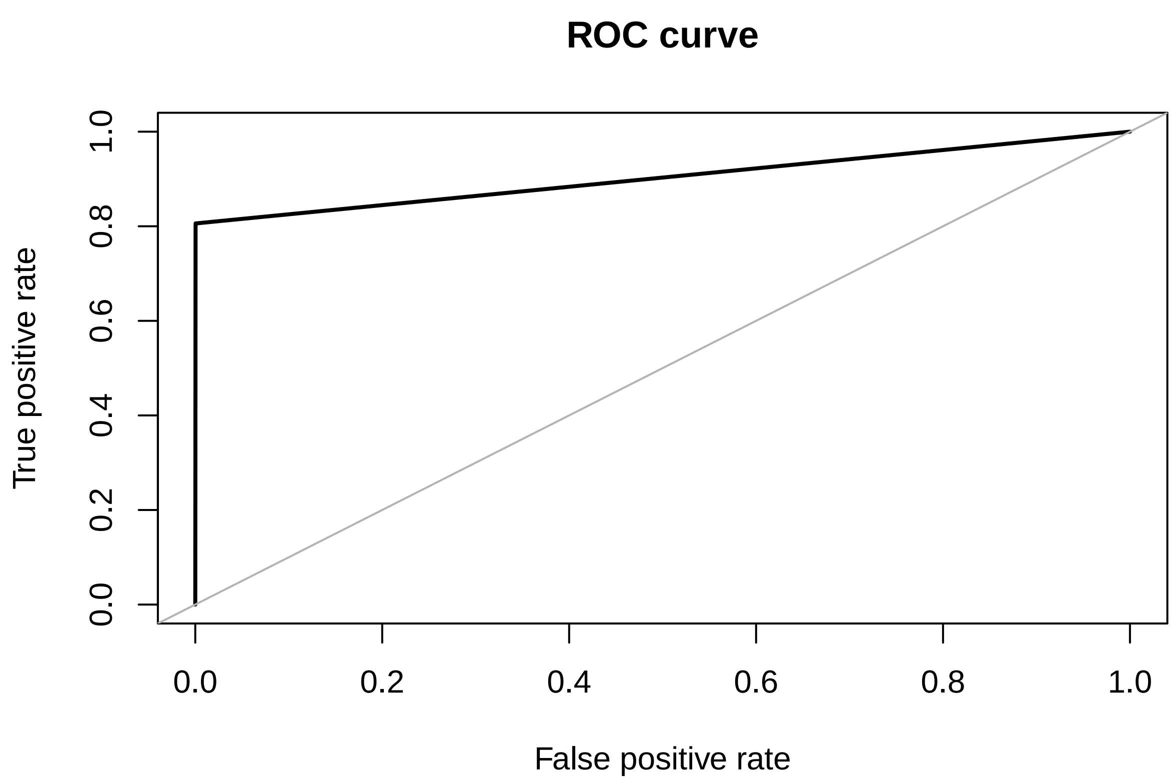
In our result we tried a lot of example with using original data, down-sampling data, up-sampling data and rose sampling data and with this sampling methods we use four different models such as Decision Tree, Logistic Regression, Random Forest and XG Boost. We are looking at this models by using sampling methods in each of them and then, we made a decision about which of the model is better for our data set.

7 Decision Trees

Firstly, we are looking at Decision Tree model. We put original data, down-sampling data, up-sampling data and rose sampling data in the decision tree model.

Decision trees on original (imbalanced) data set

```
#Decision Tree Model Performance on original imbalanced data
set.seed(5627)

orig_fit <- rpart(Class ~ ., data = train)

#Evaluate model performance on test set
pred_orig <- predict(orig_fit, newdata = test, method = "class")

roc.curve(test$Class, pred_orig[,2], plotit = TRUE)
```

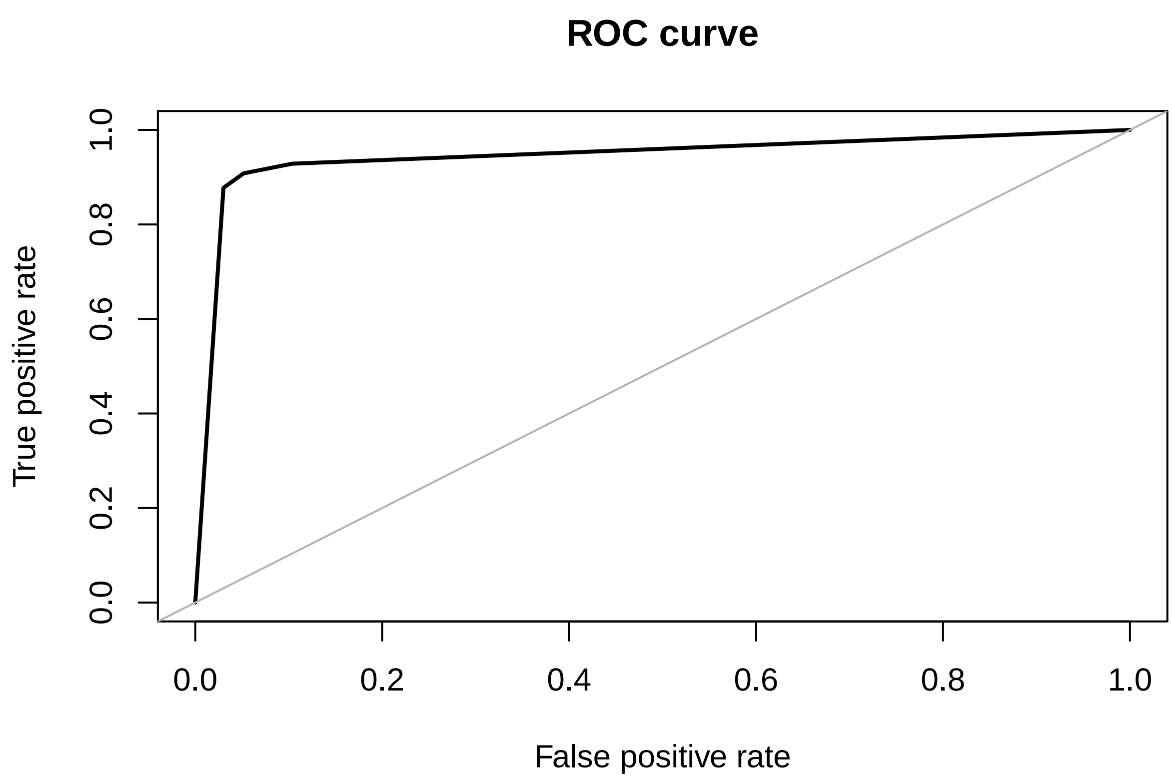



```
## Area under the curve (AUC): 0.903
```

Decision trees on up-sampled dataset

```
set.seed(5627)
# Build up-sampled model with Decision Tree
```

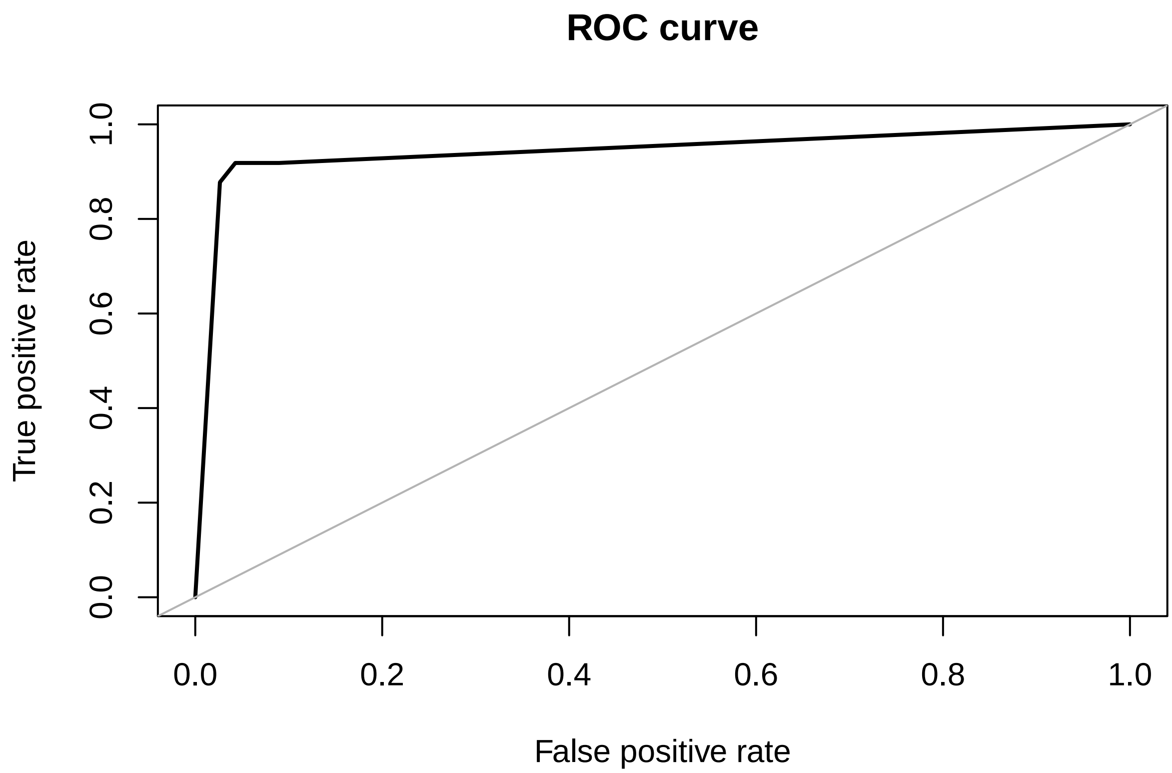
```
up_fit <- rpart(Class ~ ., data = up_train)

# AUC on up-sampled data
pred_up <- predict(up_fit, newdata = test)

roc.curve(test$Class, pred_up[,2], plotit = TRUE)
```




```
## Area under the curve (AUC): 0.944
```

Decision trees on down-sampled dataset

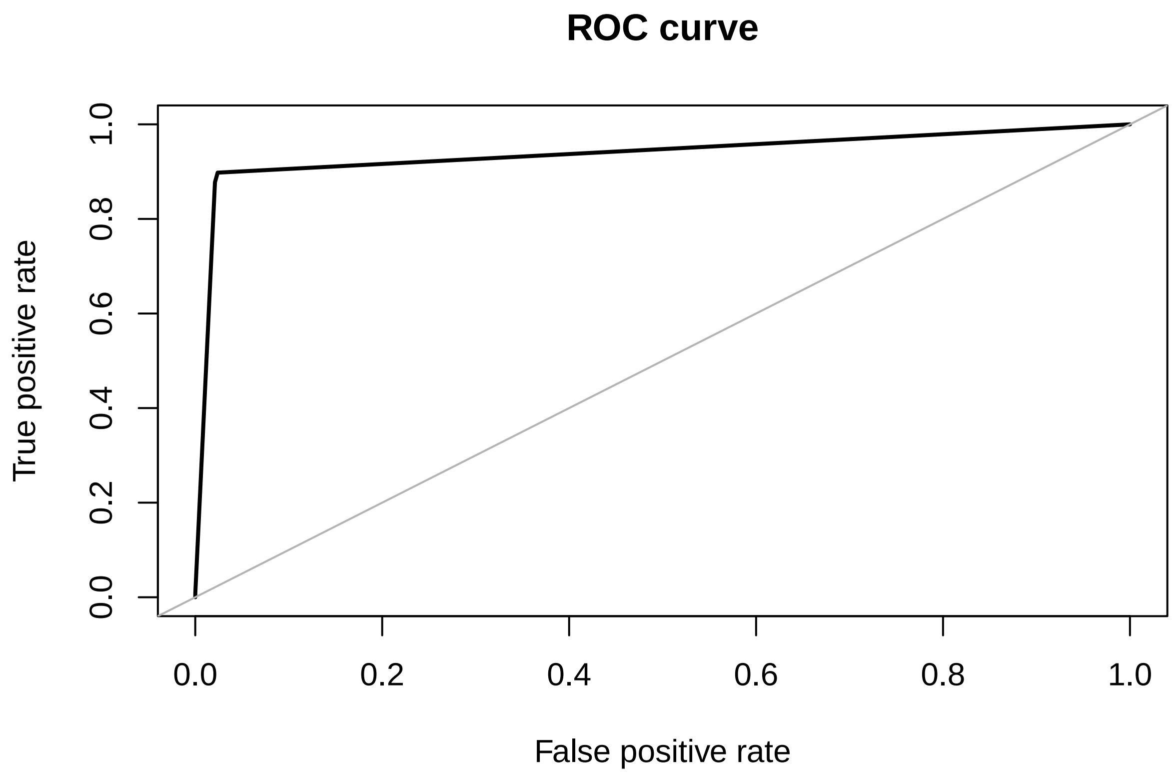
```
set.seed(5627)
# Build down-sampled model with Decision Tree


down_fit <- rpart(Class ~ ., data = down_train)


# AUC on down-sampled data
pred_down <- predict(down_fit, newdata = test)

roc.curve(test$Class, pred_down[,2], plotit = TRUE)
```




```
## Area under the curve (AUC): 0.943
```

Decision trees on rose-sampled dataset

```
set.seed(5627)
# Build rose model with Decision Tree


rose_fit <- rpart(Class ~ ., data = rose_train)

# AUC on rose data
pred_rose <- predict(rose_fit, newdata = test)

roc.curve(test$Class, pred_rose[,2], plotit = TRUE)
```




```
## Area under the curve (AUC): 0.938
```

If we look at the roc curve generally we can see that our roc curves close to the best one which means that close to true positive one. Then if we want to examine detailly or down to a single metric, AUC is useful for us in this issue. AUC stands for area under the (ROC) curve. Generally, the higher the AUC score (closest to 1.0), the better a classifier performs for the given task.

In our result we see that in decision tree model;

- with original data the AUC is 0.903,
- with up-sampling data the AUC is 0.944,
- with down-sampling data the AUC is 0.943,
- with rose-sampling data the AUC is 0.938.

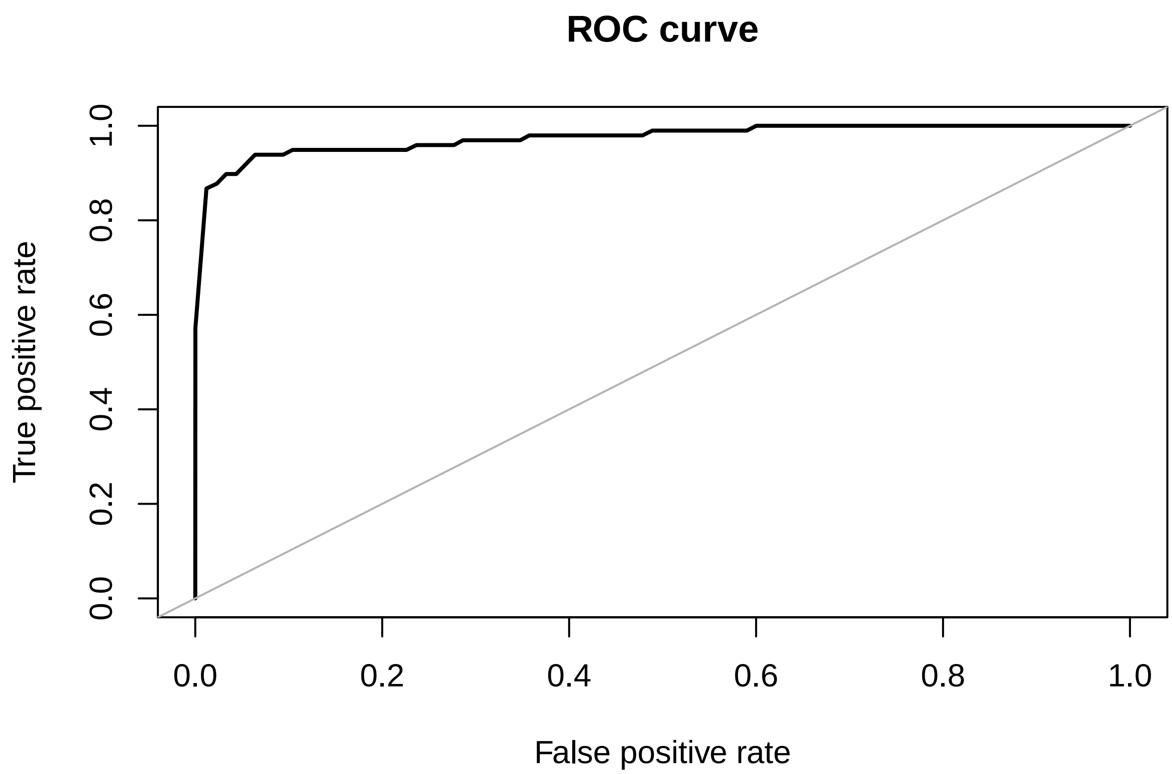
If wee look at the AUC results, the better result or the closest one to the TP line is up-sampling method in decision tree model.

8 Logistic Regression

Secondly, we are looking at Logistic Regression model. We put original data, down-sampling data, up-sampling data and rose sampling data in the logistic regression model.

Logistic Regression on original (imbalanced) dataset

```r
#Logistic regression with original imbalanced data
glm_fit <- glm(Class ~ ., data = train, family = 'binomial')
pred_glm <- predict(glm_fit, newdata = test, type = 'response')
roc.curve(test$Class, pred_glm, plotit = TRUE)
```



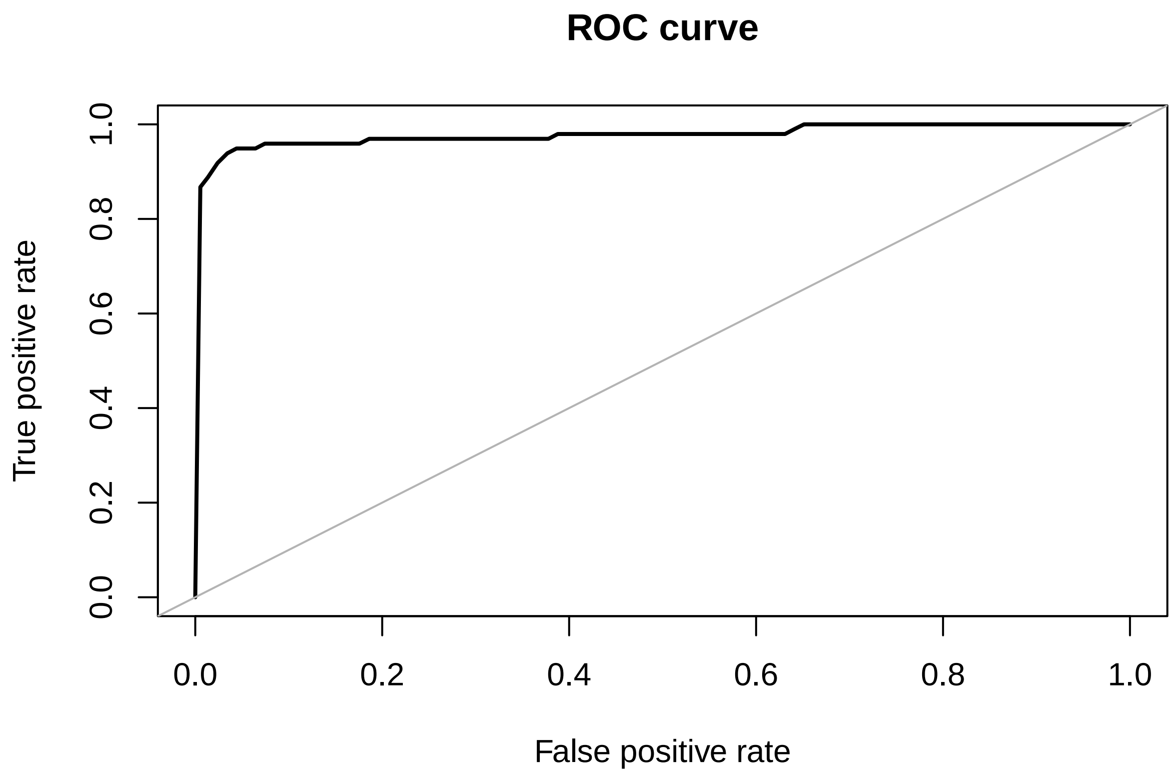

```
## Area under the curve (AUC): 0.974
```

Logistic Regression on up-sampled dataset

```r
#Logistic regression with up_train sampling technique
glm_fit <- glm(Class ~ ., data = up_train, family = 'binomial')
```

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

```r
pred_glm <- predict(glm_fit, newdata = test, type = 'response')
roc.curve(test$Class, pred_glm, plotit = TRUE)
```



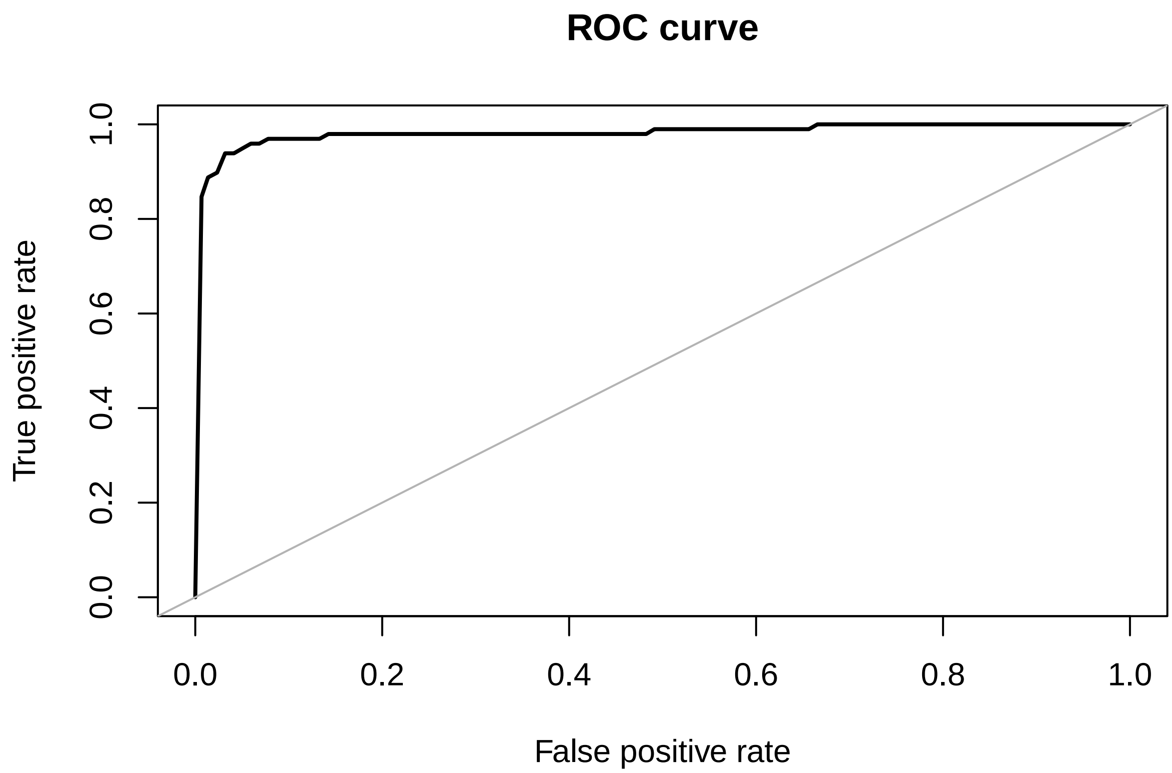

```
## Area under the curve (AUC): 0.976
```

Logistic Regression on down-sampled dataset

```r
#Logistic regression with down_train sampling technique
glm_fit <- glm(Class ~ ., data = down_train, family = 'binomial')
```

```
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
pred_glm <- predict(glm_fit, newdata = test, type = 'response')
roc.curve(test$Class, pred_glm, plotit = TRUE)
```



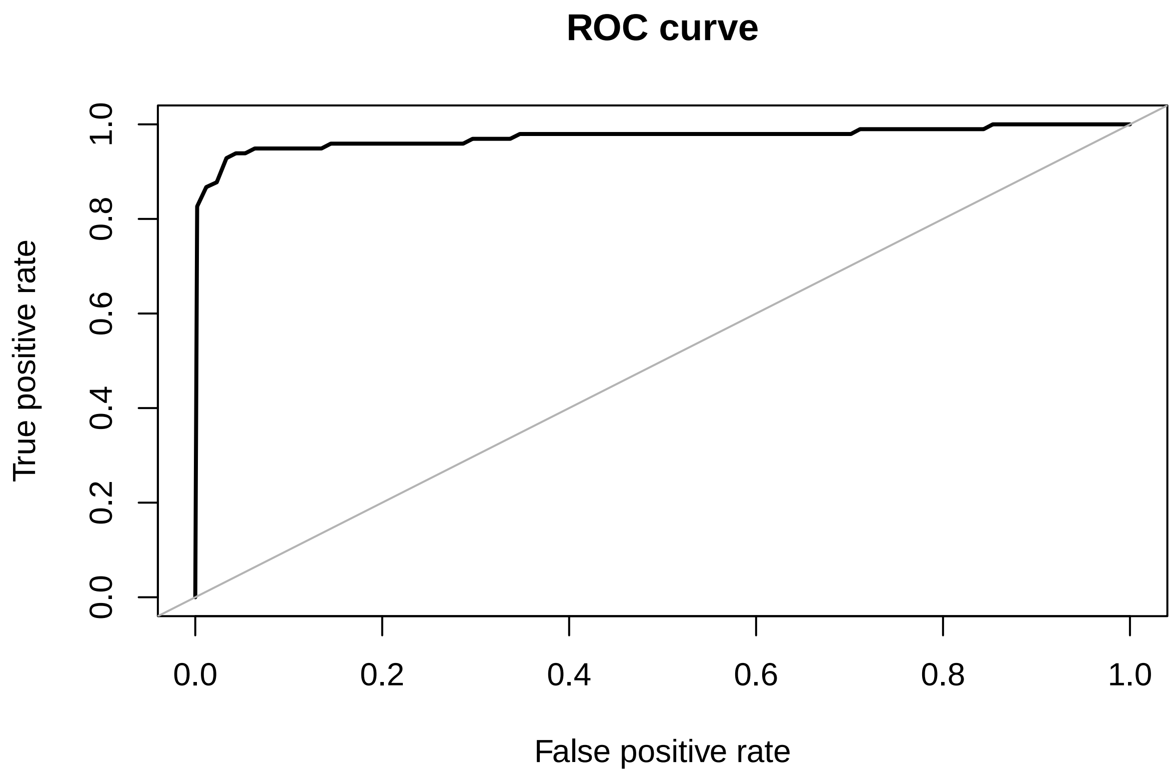

```
## Area under the curve (AUC): 0.981
```

Logistic Regression on rose-sampled dataset

```r
#Logistic regression with rose_train sampling technique

glm_fit <- glm(Class ~ ., data = rose_train, family = 'binomial')
```

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

```r
pred_glm <- predict(glm_fit, newdata = test, type = 'response')

roc.curve(test$Class, pred_glm, plotit = TRUE)
```




```
## Area under the curve (AUC): 0.973
```

In our result we see that in logistic regression model;

- with original data the AUC is 0.974,
- with up-sampling data the AUC is 0.976,
- with down-sampling data the AUC is 0.981,
- with rose-sampling data the AUC is 0.973.

If wee look at the AUC results, the better result or the closest one to the TP line is down-sampling method in decision tree model.

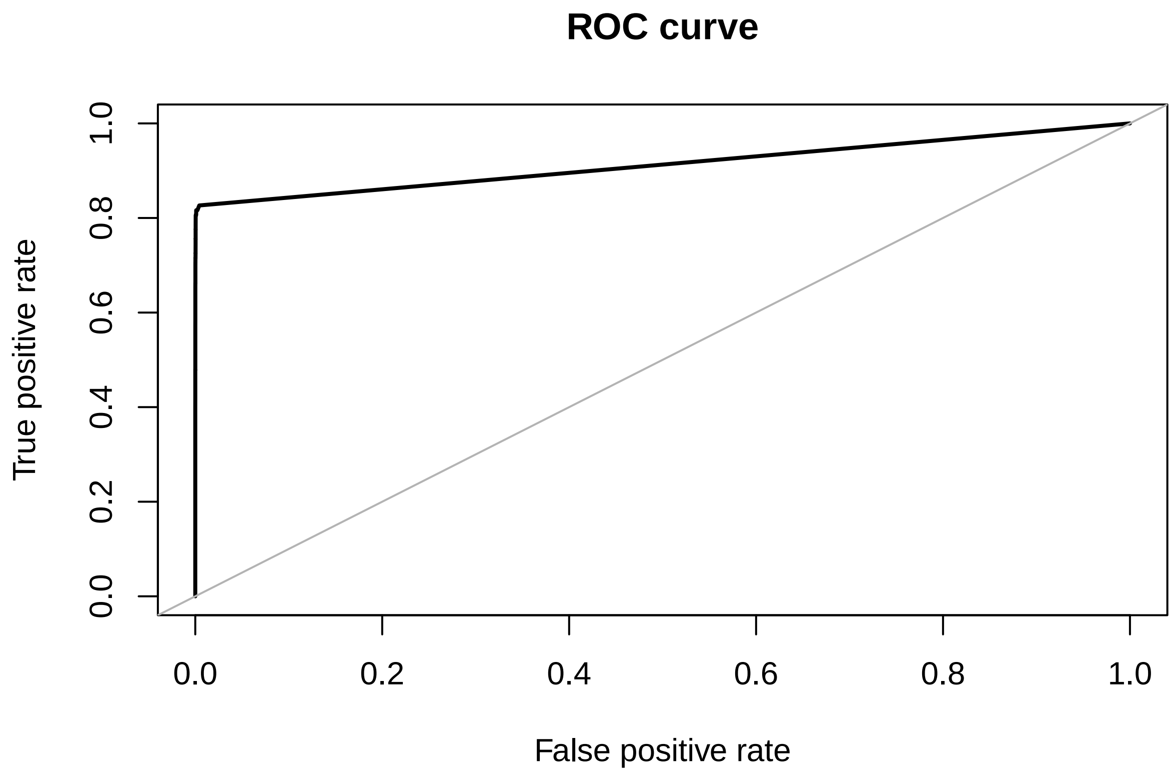
9 Random Forest

Thirdly, we are looking at Random Forest model. We put original data, down-sampling data, up-sampling data and rose sampling data in the random forest model.

Random Forest on original (imbalanced) dataset

```
#Random Forest with original imbalanced data

x = train[, -31]
y = train[,31]
```

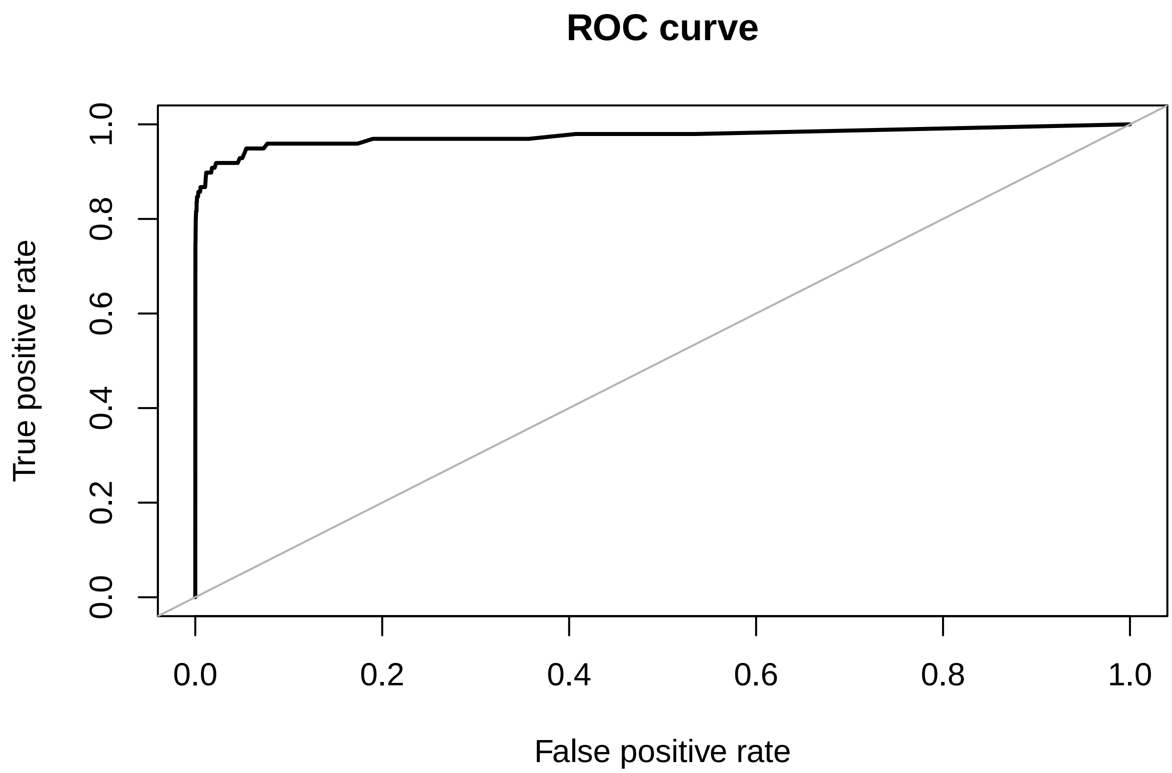
```
rf_fit <- Rborist(x, y, ntree = 1000, minNode = 20, maxLeaf = 13)


rf_pred <- predict(rf_fit, test[,-31], ctgCensus = "prob")
prob <- rf_pred$prob

roc.curve(test$Class, prob[,2], plotit = TRUE, )
```




```
## Area under the curve (AUC): 0.913
```

Random Forest on up-sampled dataset

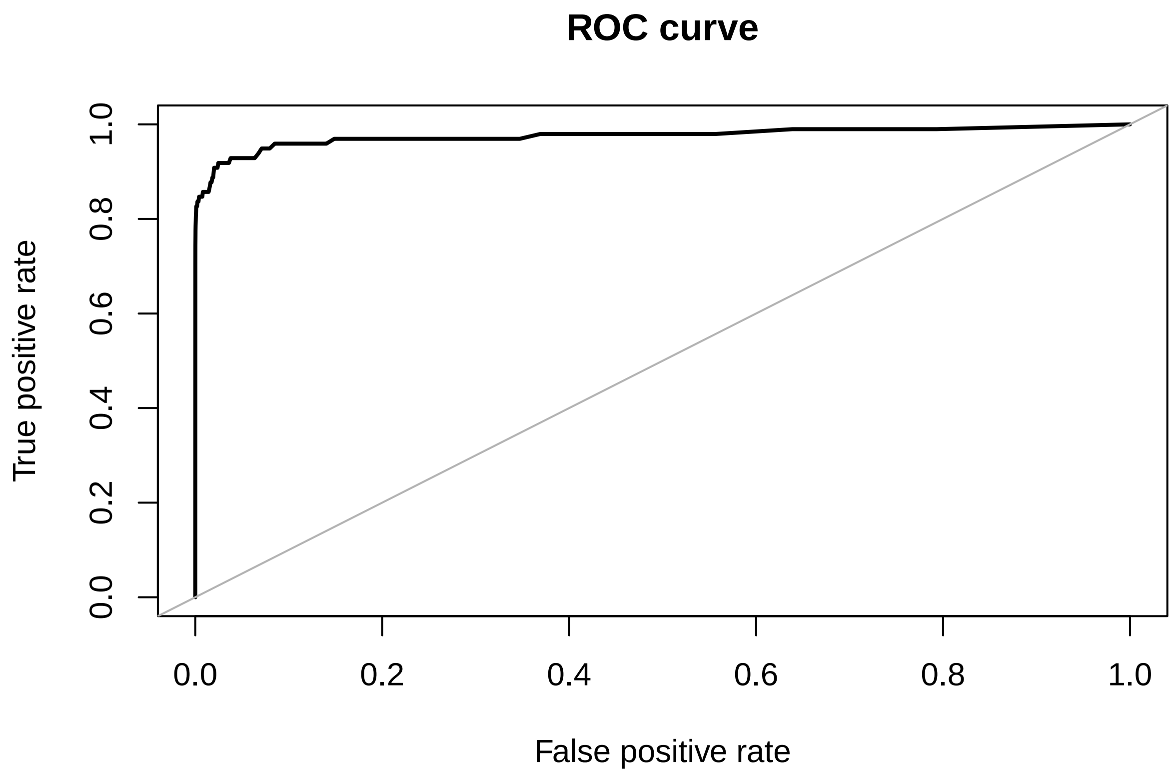
```
#Random Forest with up_train sampling

x = up_train[, -31]
y = up_train[,31]

rf_fit <- Rborist(x, y, ntree = 1000, minNode = 20, maxLeaf = 13)


rf_pred <- predict(rf_fit, test[,-31], ctgCensus = "prob")
prob <- rf_pred$prob

roc.curve(test$Class, prob[,2], plotit = TRUE, )
```




```
## Area under the curve (AUC): 0.975
```

Random Forest on down-sampled dataset

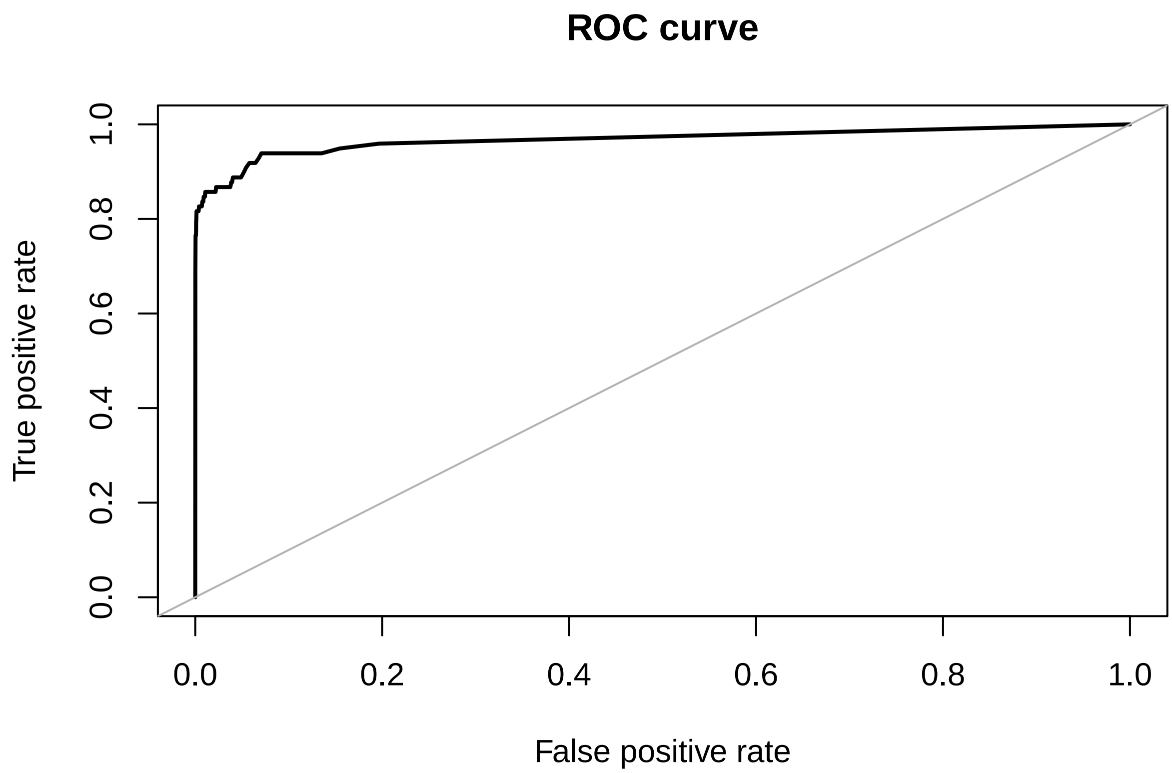
```r
#Random Forest with down_train sampling

x = down_train[, -31]
y = down_train[,31]

rf_fit <- Rborist(x, y, ntree = 1000, minNode = 20, maxLeaf = 13)


rf_pred <- predict(rf_fit, test[,-31], ctgCensus = "prob")
prob <- rf_pred$prob

roc.curve(test$Class, prob[,2], plotit = TRUE, )
```




```
## Area under the curve (AUC): 0.976
```

Random Forest on rose-sampled dataset

```
#Random Forest with rose_train sampling

x = rose_train[, -31]
y = rose_train[,31]

rf_fit <- Rborist(x, y, ntree = 1000, minNode = 20, maxLeaf = 13)


rf_pred <- predict(rf_fit, test[,-31], ctgCensus = "prob")
prob <- rf_pred$prob

roc.curve(test$Class, prob[,2], plotit = TRUE, )
```




```
## Area under the curve (AUC): 0.968
```

In our result we see that in random forest model;

- with original data the AUC is 0.913,
- with up-sampling data the AUC is 0.975,
- with down-sampling data the AUC is 0.976,
- with rose-sampling data the AUC is 0.968.

If wee look at the AUC results, the better result or the closest one to the TP line is down-sampling method in random forest model.

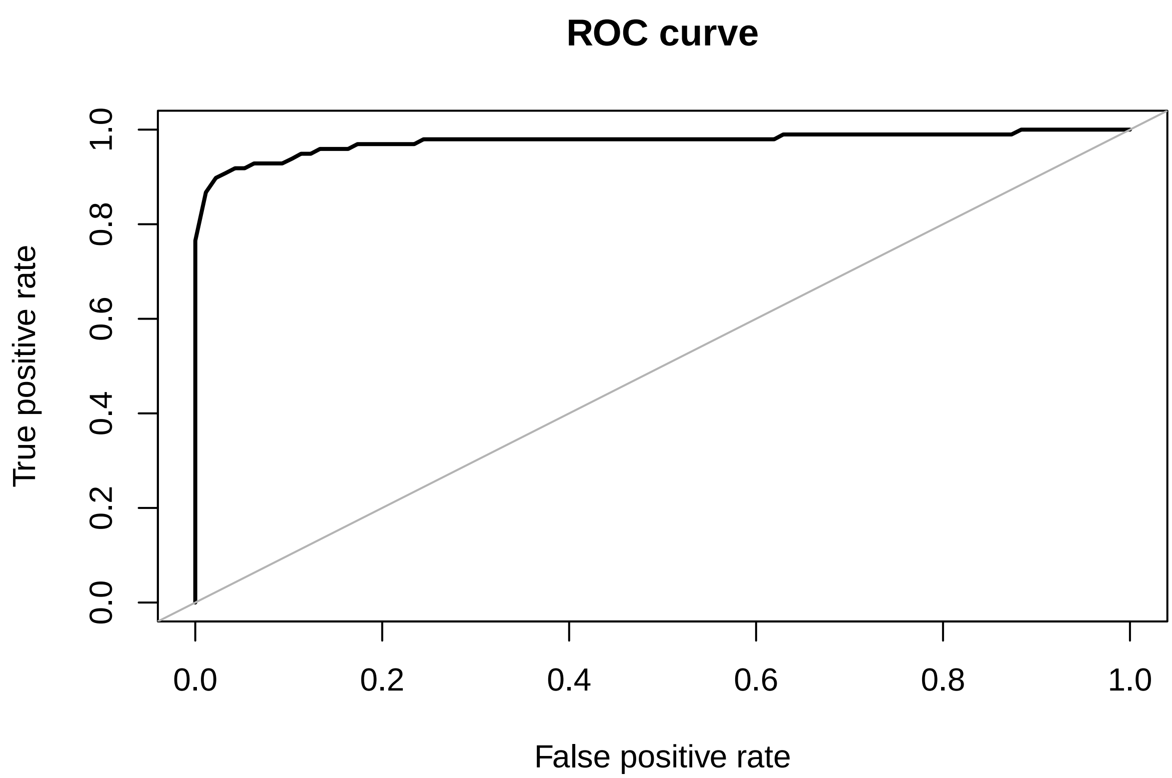
10 XG Boost

Fourthly, we are looking at XG Boost model. We put original data, down-sampling data, up-sampling data and rose sampling data in the xg boost model.

XG Boost on original (imbalanced) dataset

```
#XG BOOST original imbalanced data

# Convert class labels from factor to numeric

labels_original <- train$Class
```

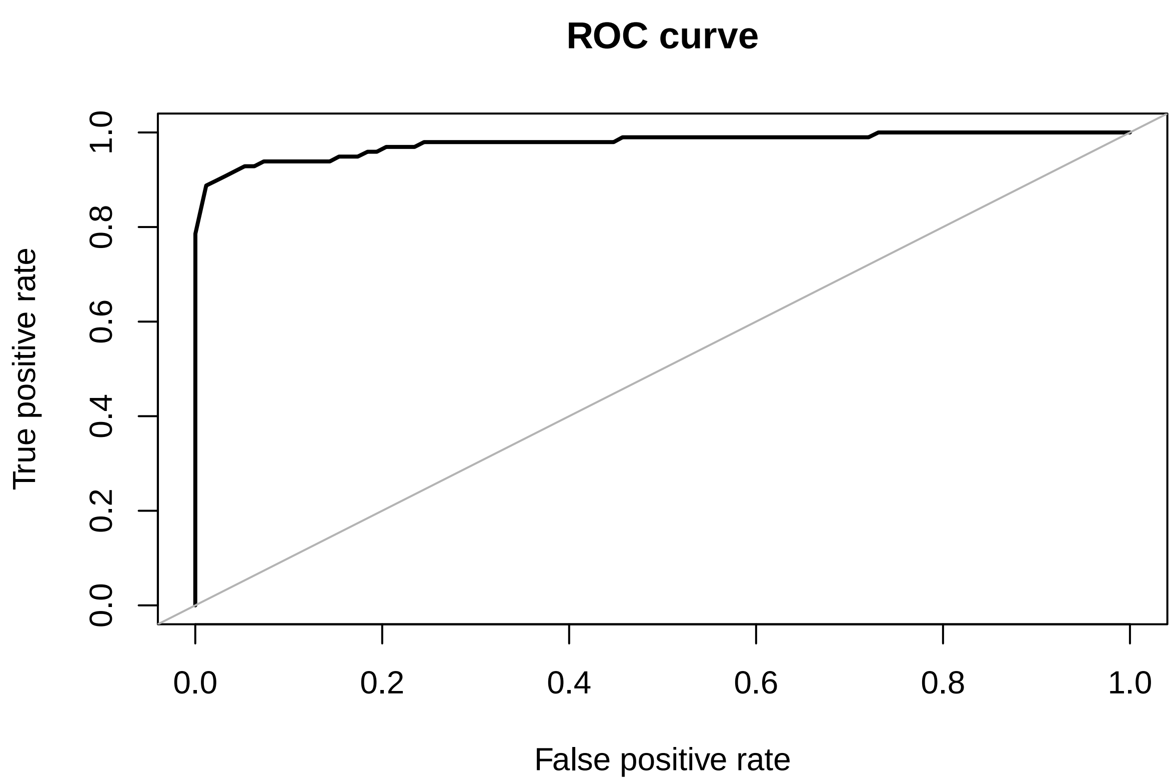
```
y <- recode(labels_original, 'Not_Fraud' = 0, "Fraud" = 1)

set.seed(42)
xgb <- xgboost(data = data.matrix(train[,-31]),
 label = y,
 eta = 0.1,
 gamma = 0.1,
 max_depth = 10,
 nrounds = 300,
 objective = "binary:logistic",
 colsample_bytree = 0.6,
 verbose = 0,
 nthread = 7,
)
xgb_pred <- predict(xgb, data.matrix(test[,-31]))

roc.curve(test$Class, xgb_pred, plotit = TRUE)
```




```
## Area under the curve (AUC): 0.975
```

XG Boost on up-sampled dataset

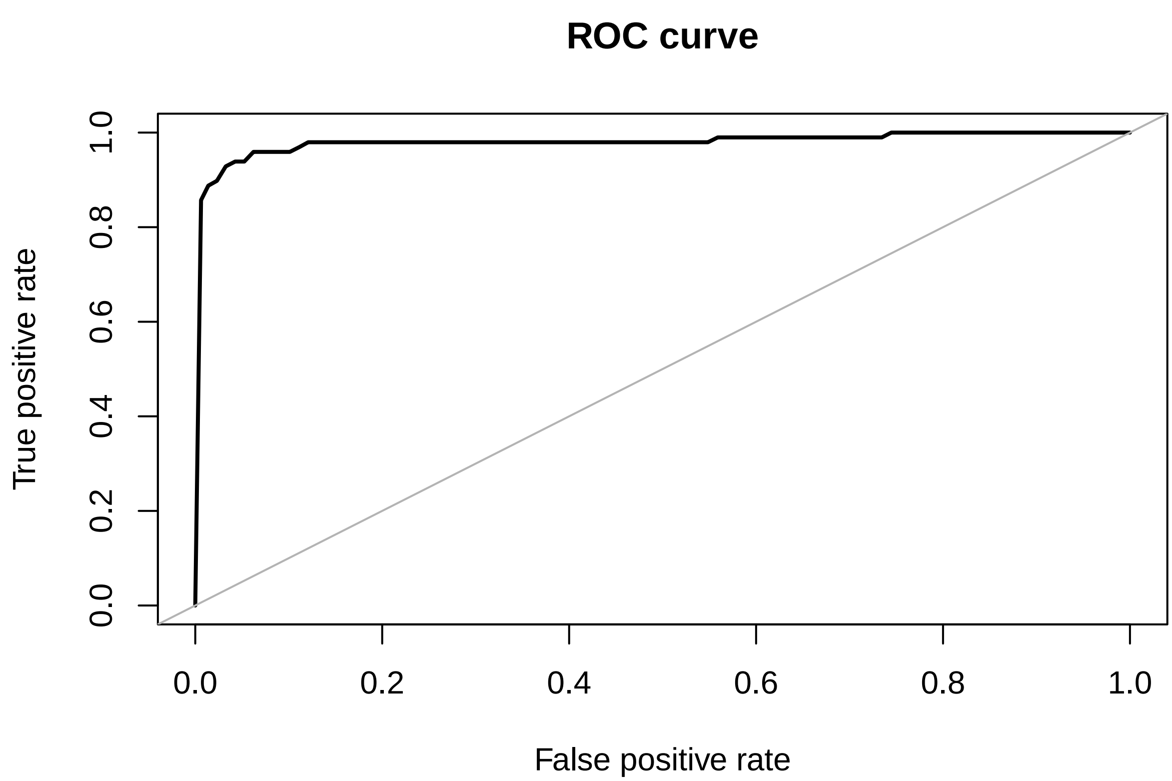
```r
#XG BOOST up_train sampling technique

# Convert class labels from factor to numeric

labels_up <- up_train$Class

y <- recode(labels_up, 'Not_Fraud' = 0, "Fraud" = 1)

set.seed(42)
xgb <- xgboost(data = data.matrix(up_train[,-31]),
 label = y,
 eta = 0.1,
 gamma = 0.1,
 max_depth = 10,
 nrounds = 300,
 objective = "binary:logistic",
 colsample_bytree = 0.6,
 verbose = 0,
 nthread = 7,
)
xgb_pred <- predict(xgb, data.matrix(test[,-31]))

roc.curve(test$Class, xgb_pred, plotit = TRUE)
```




```
## Area under the curve (AUC): 0.977
```

XG Boost on down-sampled dataset

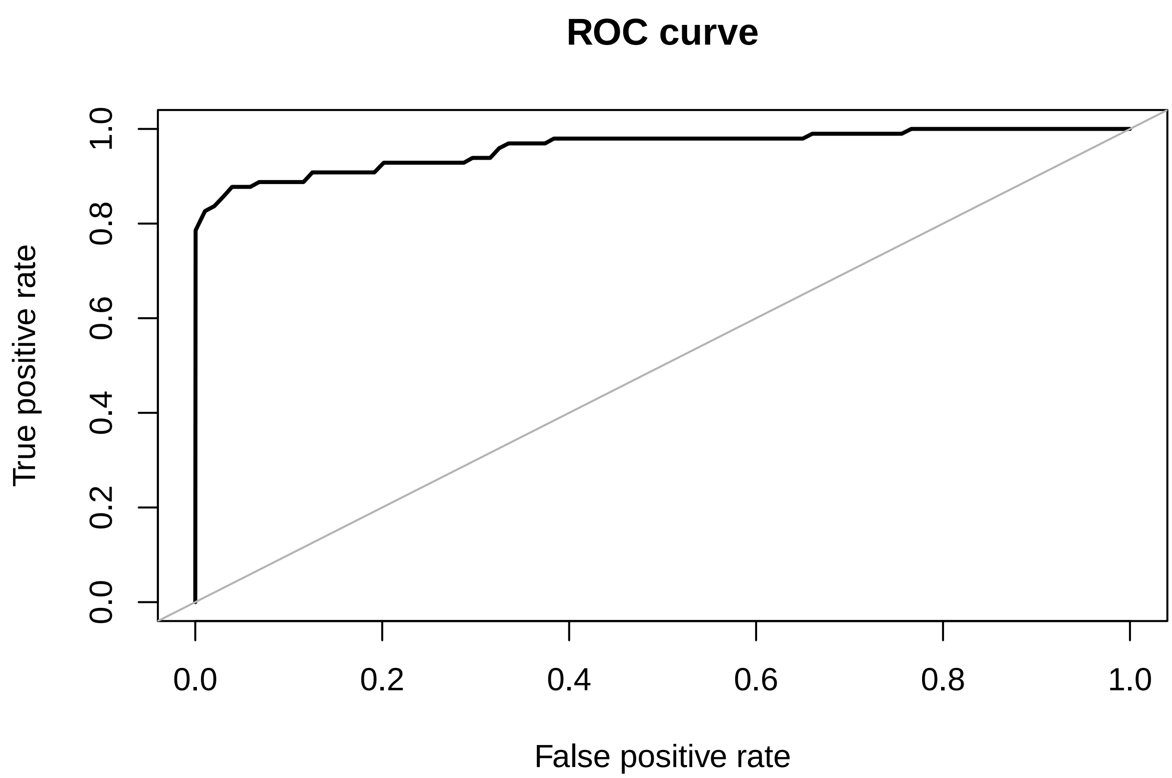
```
#XG BOOST down_train sampling technique

# Convert class labels from factor to numeric

labels_down <- down_train$Class

y <- recode(labels_down, 'Not_Fraud' = 0, "Fraud" = 1)

set.seed(42)
xgb <- xgboost(data = data.matrix(down_train[,-31]),
 label = y,
 eta = 0.1,
 gamma = 0.1,
 max_depth = 10,
 nrounds = 300,
 objective = "binary:logistic",
 colsample_bytree = 0.6,
 verbose = 0,
 nthread = 7,
)
xgb_pred <- predict(xgb, data.matrix(test[,-31]))

roc.curve(test$Class, xgb_pred, plotit = TRUE)
```



```
## Area under the curve (AUC): 0.979
```

XG Boost on rose-sampled dataset

```r
#XG BOOST rose_train sampling technique

# Convert class labels from factor to numeric

labels_rose <- rose_train$Class

y <- recode(labels_rose, 'Not_Fraud' = 0, "Fraud" = 1)

set.seed(42)
xgb <- xgboost(data = data.matrix(rose_train[,-31]),
 label = y,
 eta = 0.1,
 gamma = 0.1,
 max_depth = 10,
 nrounds = 300,
 objective = "binary:logistic",
 colsample_bytree = 0.6,
 verbose = 0,
 nthread = 7,
)
xgb_pred <- predict(xgb, data.matrix(test[,-31]))

roc.curve(test$Class, xgb_pred, plotit = TRUE)
```




```
## Area under the curve (AUC): 0.960
```

In our result we see that in XG Boost model;

- with original data the AUC is 0.975,
- with up-sampling data the AUC is 0.977,
- with down-sampling data the AUC is 0.979,
- with rose-sampling data the AUC is 0.960.

If wee look at the AUC results, the better result or the closest one to the TP line is down-sampling method in XG Boost model.

```r
# Define the column and row names.
colnames = c("Original ", "Up-sampling ", "Down-sampling ", "Rose-sampling")
rownames = c("Decision Tree ", "Logistic Regression ", "Random Forest ", "XG Boost ")
#Define a matrix
matrix <- matrix(cbind(c(0.903 ,0.944 ,0.943 ,0.938 ),c(0.974,0.976,0.981,0.973),c(0.913,0.975,0.976,0.9
print(matrix)
```

```
##                      Original  Up-sampling  Down-sampling  Rose-sampling
## Decision Tree           0.903        0.944          0.943          0.938
## Logistic Regression     0.974        0.976          0.981          0.973
## Random Forest           0.913        0.975          0.976          0.968
## XG Boost                0.975        0.977          0.979          0.960
```

To show our result in a one matrix table.

11 Conclusion

In conclusion, We found that;

1- Up-sampling method gives us better result in decision tree model. So, up-sampling is the most suitable sampling technique for the decision tree model in our dataset.

2- Down-sampling method gives us better result in logistic regression model. As a result of, down-sampling is the most suitable sampling technique for the logistic regression model in our dataset.

3- Down-sampling method gives us better result in random forest model. As a result of, down-sampling is the most suitable sampling technique for the random forest model in our dataset.

4- Down-sampling method gives us better result in XG Boost model. As a result of, down-sampling is the most suitable sampling technique for the XG Boost model in our dataset.

The best sampling technique's AUC scores for the models are;

1- Decision Tree: 0.944

2- Logistic Regression: 0.981

3- Random Forest: 0.976

4- XG Boost: 0.979

To sum up, we searched for the answer to the question of how we can make the our data more balanced in our Credit Card Fraud Detection project, that is, in a project where fraud cases are quite low and the dataset is very imbalanced. In response to this question, we saw that some resampling methods could be tried, besides, we tried these methods not only on a single model, but also on 4 different models, and determined which sampling method would perform better on which model for our data set. Among the sampling methods, we saw that the down-sampling method showed the best performance on 3 different models. We have come to the conclusion that the model that shows the most appropriate approach for our data, standing out among the complex algorithms such as Decision Tree, Random Forest and XG Boost, is Logistic Regression with a value of 0.981 AUC.

12 References

– Ingle, A. (2020, December 21). Credit Card Fraud Detection with R + (sampling). Kaggle. https://www.kaggle.com/code/atharvaingle/credit-card-fraud-detection-with-r-sampling/notebook

– Zientala, P. (n.d.). Credit card fraud detection using Machine Learning. http://rstudio-pubs-static.s3.amazonaws.com/334864_28050f7860dd4927a596872f0cd52401.html

– Team, T. A. I. (2020, May 29). How, When, and Why Should You Normalize / Standardize / Rescale. . . Towards AI. https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff

– Team, T. A. I. (2022, March 17). Standardize Data Frame Columns in R (2 Examples) | scale Function. Statistics Globe. https://statisticsglobe.com/standardize-data-frame-columns-in-r-scale-function

– Steen, D. (2021, December 15). Understanding the ROC Curve and AUC - Towards Data Science. Medium. https://towardsdatascience.com/understanding-the-roc-curve-and-auc-dd4f9a192ecb