Credit Card Fraud Detection Project Report

Arezou Tufekci

5/15/2021

Contents

1 Background

The USA is the global leader as the most credit card fraud prone country with 38.6 percent of reported fraud losses in 2018 [1]. Credit card fraud is the most common and popular kind of identity theft and makes up 35.4% of all identity theft reports [1]. Fraud detection is a challenging problem since fraudulent transactions are rare and they represent a small fraction of transactions but can quickly turn into large sums of money. The good news is that with advances in Machine Learning, system can learn, adapt and uncover emerging patterns for preventing fraud.

1.1 Objectives

The objective of this project is to train a Machine Learning algorithm on a dataset made up of credit card transactions in order to successfully predict fraudulent transactions.

1.2 Structure

This document is structured as follows:

1. **Introduction/overview:** Introduces the problem and describe the goal of the project.
2. **Summary:** Describes the dataset and variables, explores the data further and prepares data for analysis.
3. **Method/Analysis:** Explains the process and techniques used. Defines the models as we improve upon the model and explains different Machine Learning algorithms and the results for each method.
4. **Results:** Summary of findings.presents the modeling results and discusses the model performance.
5. **Conclusion:** Gives a brief summary of the report, its potential impact, its limitations, and future work.
6. **Reference Page:** Cited sources.
7. **Definition Of Terms:** Describe terms further for clarity

2 Summary

2.1 Understanding the Data

As a pre-requisite, install all the required packages and import libraries and dependencies for the dataframe. We start by downloading the csv file called creditcard.csv from Kaggle website (Source - https://www.kaggle.com/mlg-ulb/creditcardfraud). This dataset was collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. The dataset is made up of credit card transactions that occured in two days in September 2013 by European cardholders.

We read the csv file into a dataframe called 'df' and quickly analyze the structure of the dataframe.

```
# We first look at top 6 rows of the dataframe.
head(df)
```

```
##   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
```

```
#  We then use str to know more about the dataframe and its constituents.
str(df)
```

```
## 'data.frame':    284807 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

The dataframe has 284,807 transactions with 31 columns (variables). The dataset contains only numerical variables. As stated in the Kaggle website this is a result of a PCA (Principal component analysis) dimensionality reduction to protect sensitive information [3]. Features V1, V2, … V28 are obtained by PCA except for 'Time', 'Amount', and 'Class'. The variable 'Class' indicates whether a transaction is fraudulent (1 = Fraud) or not (0 = Legal).

```
summary(df)
```

```
##       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.:139320   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
```

We can see that the distribution of many PCA components is centered around zero, suggesting that the variables were standardized as part of the PCA transform. Also none of the columns have inconsistent datatypes so no conversions are required.

```
# Perform basic data cleansing by checking for nulls in the dataset.
colSums(is.na(df))
```

```
##    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
```

```
# Observed no null values and hence no null treatment is required.
```

There are no missing values in our dataset.

```
# The following code will convert our dependent variable (Class) to a factor.
df$Class = factor(df$Class)
table(df$Class)
```

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

```
prop.table(table(df$Class))
```

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

This dataframe contains 492 fraud and 284,315 legal transactions. This shows a highly imbalanced set to work with as the frauds account for 0.172% of all transactions. We will address the problem of training a model to perform against highly imbalanced data and outline some techniques and expectations.

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally [2]. We have a binary classification problem with 284,807. A total of 284,315 Legal labeled with Class '0' and the remaining 492 are labeled with Class '1'. This is an imbalanced dataset and the ration Class '1' to Class '0' is ~ 1:578 ! If we to use Logistic Regression, the result will ignore fraud Class and most of the time the result are overfit to the legal transaction Class. More interestingly, there are problems where a class imbalance is not just common, it is expected! For example, in datasets like this one that characterize fraudulent transaction are imbalanced. The vast majority of the transactions will be in the legal Class and a very small minority will be in the Fraud Class. It is very common to start with classification accuracy, because it is often the first measure we use when evaluating models on our classification problems [2]. Accuracy is not appropriate here as even a classifier which labels all transactions as non-fraudulent will have over 99% accuracy. Why is that? Because our models look at the data and cleverly decide that the best thing to do is to always predict 'Legal' Class and achieve high accuracy. This is exactly why Machine Learning does not work well with imbalanced data. So now that we understand what class imbalance is and why it provides misleading classification accuracy, we will need to change our performance metric. Next, we explore our options.

2.2 Preparing the Data

2.2.1 Re-sampling the Dataset

Machine Learning algorithms assume that the dataset has balanced class distributions. As mentioned above, Machine Learning algorithms struggle with accuracy because of the unequal distribution in dependent variable. This causes the performance of existing classifiers to get biased towards majority class. We can use Sampling to build a more balanced data. Sampling technique should only be applied to the training set and not to the testing set. The three main methods are:

- **Over-sampling**: Add copies of instances from the under-represented class.
- **Under-sampling**: Delete instances from the over-represented class.
- **Synthetic Data Generation**: Randomly sample the attribute from instances in the minority Class.
  - **SMOTE** Draws artificial samples by choosing points that lie on the line connecting the rare observation to one of its nearest neighbors.
  - **ROSE** Uses smoothed bootstrapping to draw artificial samples from the neighborhood around the minority class.

2.3 Splitting the Data into Train and Test Sets

We will split the dataset into train and test sets in 70:30 ratio respectively. We make the decision to remove 'Time' feature from our set prior to splitting. Time feature does not indicate the actual time of the transaction and is more of listing the data in chronological order. We assume that 'Time' feature has little or no significance in correctly classifying a fraud transaction and hence eliminate this column from further analysis.

```
#Remove 'Time' variable
df <- df[,-1]

#Change Class variable to factor
df$Class <- as.factor(df$Class)
levels(df$Class)  <- c("Legal", "Fraud")
#Scale numeric variables
df[,-30] <- scale(df[,-30])
head(df)
```

```
##           V1          V2        V3         V4           V5          V6
## 1 -0.6942411 -0.04407485 1.6727706  0.9733638 -0.245116153  0.34706734
## 2  0.6084953  0.16117564 0.1097969  0.3165224  0.043483276 -0.06181986
## 3 -0.6934992 -0.81157640 1.1694664  0.2682308 -0.364571146  1.35145121
## 4 -0.4933240 -0.11216923 1.1825144 -0.6097256 -0.007468867  0.93614819
## 5 -0.5913287  0.53154012 1.0214099  0.2846549 -0.295014918  0.07199846
## 6 -0.2174742  0.58167387 0.7525841 -0.1188331  0.305008424 -0.02231344
##           V7          V8         V9         V10        V11        V12
## 1  0.1936786  0.08263713  0.3311272  0.08338540 -0.5404061 -0.6182946
## 2 -0.0637001  0.07125336 -0.2324938 -0.15334936  1.5800001  1.0660867
## 3  0.6397745  0.20737237 -1.3786729  0.19069928  0.6118286  0.0661365
## 4  0.1920703  0.31601704 -1.2625010 -0.05046786 -0.2218912  0.1783707
## 5  0.4793014 -0.22650983  0.7443250  0.69162382 -0.8061452  0.5386257
## 6  0.3849353  0.21795429 -0.5176177 -0.34110050  1.3140441  0.3601815
##          V13        V14        V15        V16         V17         V18
## 1 -0.9960972 -0.3246096  1.6040110 -0.5368319  0.24486302  0.03076988
## 2  0.4914173 -0.1499822  0.6943592  0.5294328 -0.13516973 -0.21876220
## 3  0.7206986 -0.1731136  2.5629017 -3.2982296  1.30686559 -0.14478974
## 4  0.5101678 -0.3003600 -0.6898362 -1.2092939 -0.80544323  2.34530040
## 5  1.3522420 -1.1680315  0.1913231 -0.5152042 -0.27908030 -0.04556892
## 6 -0.3597909 -0.1430569  0.5655061  0.4584589 -0.06844494  0.08190778
##           V19         V20         V21          V22         V23        V24
## 1  0.49628116  0.32611744 -0.02492332  0.382853766 -0.17691102  0.1105067
## 2 -0.17908573 -0.08961071 -0.30737626 -0.880075209  0.16220090 -0.5611296
## 3 -2.77855597  0.68097378  0.33763110  1.063356404  1.45631719 -1.1380901
## 4 -1.51420227 -0.26985475 -0.14744304  0.007266895 -0.30477601 -1.9410237
## 5  0.98703556  0.52993786 -0.01283920  1.100009340 -0.22012301  0.2332497
## 6 -0.04077658  0.11021522 -0.28352172 -0.771425648 -0.04227277 -0.6132723
##          V25        V26         V27         V28      Amount Class
## 1  0.2465850 -0.3921697  0.33089104 -0.06378104  0.24496383 Legal
## 2  0.3206933  0.2610690 -0.02225564  0.04460744 -0.34247394 Legal
## 3 -0.6285356 -0.2884462 -0.13713661 -0.18102051  1.16068389 Legal
## 4  1.2419015 -0.4602165  0.15539593  0.18618826  0.14053401 Legal
## 5 -0.3952009  1.0416095  0.54361884  0.65181477 -0.07340321 Legal
## 6 -0.4465828  0.2196368  0.62889938  0.24563577 -0.33855582 Legal
```

```
# Split dataset into train and test sets in 70:30 ratio respectively
set.seed(42)
split <- sample.split(df$Class, SplitRatio = 0.7)
train <-  subset(df, split == TRUE)
test <- subset(df, split == FALSE)
```

2.4 Creating Sampled Sets of Data

We will now use the sampling techniques defined above in order to produce different versions of the train set.

```
# Create Original Train Set
table(train$Class)
```

```
##
##  Legal  Fraud
## 199020    344
```

```
# Create Under-sampling Train Set
set.seed(42)
downsamp_train <- downSample(x = train[, -ncol(train)], y = train$Class)
table(downsamp_train$Class)
```

```
##
## Legal Fraud
##   344   344
```

```
# Create Over-sampling Train Set
```

```
set.seed(42)
upsamp_train <- upSample(x = train[, -ncol(train)], y = train$Class)
table(upsamp_train$Class)
```

```
##
##  Legal  Fraud
## 199020 199020
```

```
# Create SMOTE Train Set
set.seed(42)
smote_train <- SMOTE(Class ~ ., data  = train)
table(smote_train$Class)
```

```
##
## Legal Fraud
##  1376  1032
```

```
# Create ROSE Train Set
set.seed(42)
rose_train <- ROSE(Class ~ ., data  = train)$data
table(rose_train$Class)
```

```
##
## Legal Fraud
## 99791 99573
```

The Following table summarizes the different count of Class variable (Legal/Fraud) in our train set.

Sampling Methods (Train set)	# of Legal	# of Fraud
No Sampling: Original	199020	344
Under Sampling	344	344
Over Sampling	199020	199020
SMOTE	1376	1032
ROSE	99791	99573

3 Modeling Approach

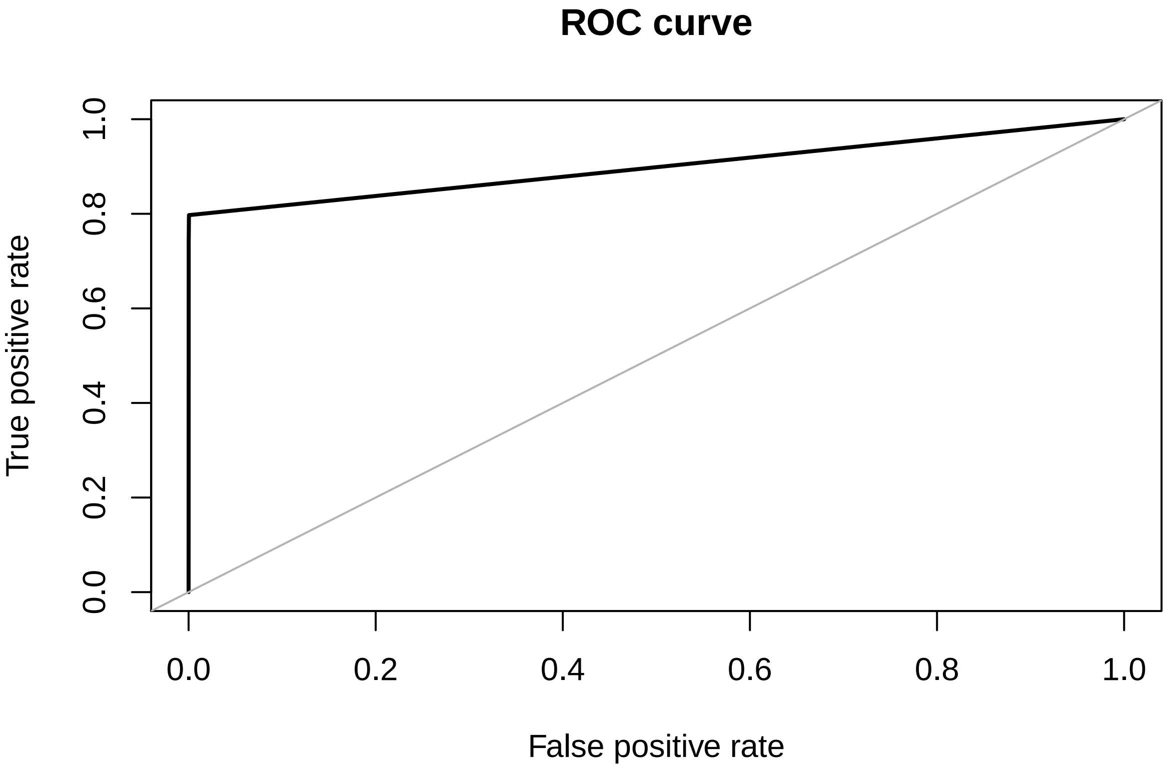
As said before, Accuracy is not the metric to use when working with an imbalanced dataset. There are metrics that have been designed to tell a better story when working with imbalanced classes. ROC Curves and Precision-Recall Curves are some useful tools. ROC curves are appropriate when the observation are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets. Since we have created a balanced data using the sampling technique above, we will continue with ROC curves for predicting the probability of a binary outcome. ROC curve stands for Receiver Operating Characteristic curve. The plot depicts the false positive rate on x-axis vs the true positive rate (y-axis) for a number of different candidate threshold (between 0 and 1). In other words, it plots the false alarm rate versus the hit rate [4]. We will explore ROC Curves AUC (Area Under Curve). The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. The AUC value lies between 0.5 and 1 where 0.5 denotes a bad classifier and 1 denotes an excellent classifier. Our modeling approach will involve training a single classifier on the train set with class imbalance suitably altered using each of the train set versions above. Depending on which version of train set yields the best roc-auc score on a holdout test set. We will then build subsequent models using that chosen technique. In our analysis, we will train several algorithms such as CART, GLM, Random Forest, and XGBoost.

3.1 CART Method

We choose CART (Classification And Regression Tree) as first model. Before we start using sampling let us first look at how CART performs with imbalanced data. we use the function roc.curve available in the ROSE package to gauge model performance on the test set.

3.1.1 CART Method: Calculate AUC using Original Train Dataset

```
# CART Model Performance on original imbalanced dataset
set.seed(42)
original_fit <- rpart(Class ~ ., data = train)
#Evaluate Model Performance on test set
pred_original  <- predict(original_fit, newdata = test, method = "class")
ROSE::roc.curve(test$Class, pred_original[,2], plotit = TRUE)
```



```
## Area under the curve (AUC): 0.898
```

We evaluate the model performance on test data by calculating the roc auc score. AUC score on the original dataset is **0.898**. This is our first attempt at our AUC and we believe we can improve on this score.

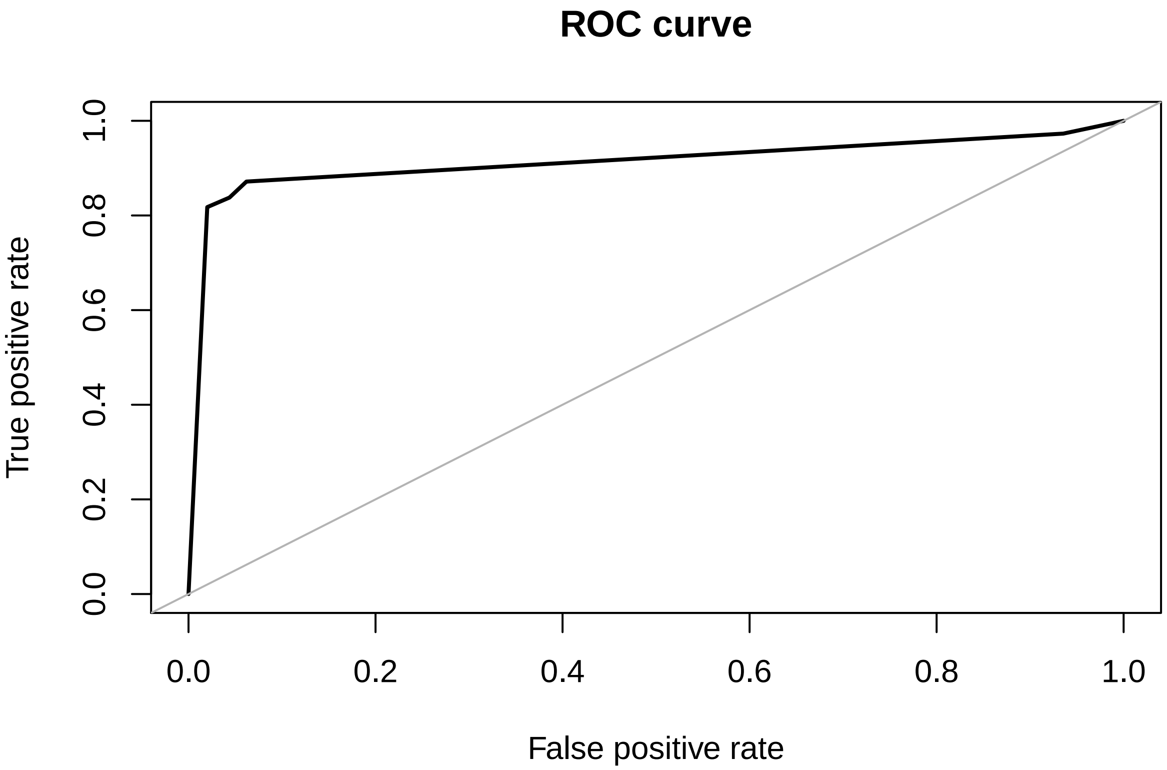
Method	AUC
CART Method: Original	**0.898**

3.1.2 CART Method: Calculate AUC using Under-sampled Train Dataset

```
set.seed(42)
# Build down-sampled model
downsample_fit <- rpart(Class ~ ., data = downsamp_train)
predict_down <- predict(downsample_fit, newdata = test)
print('Fitting downsampled model to test data')
```

```
## [1] "Fitting downsampled model to test data"
```

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



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

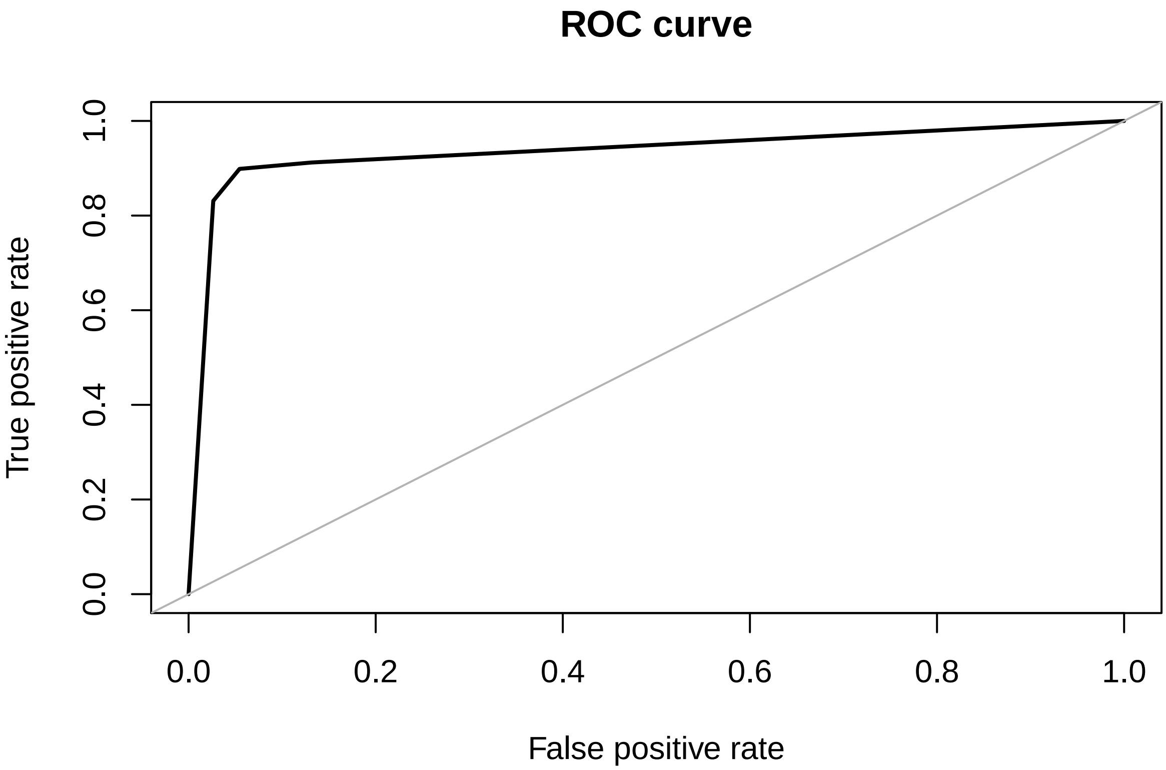
Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	**0.913**

3.1.3 CART Method: Calculate AUC using Over-sampled Train Dataset

```
set.seed(42)
# Build up-sampled model
upsamp_fit <- rpart(Class ~ ., data = upsamp_train)
predict_up <- predict(upsamp_fit, newdata = test)
print('Fitting upsampled model to test data')
```

```
## [1] "Fitting upsampled model to test data"
```

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



```
## Area under the curve (AUC): 0.935
```

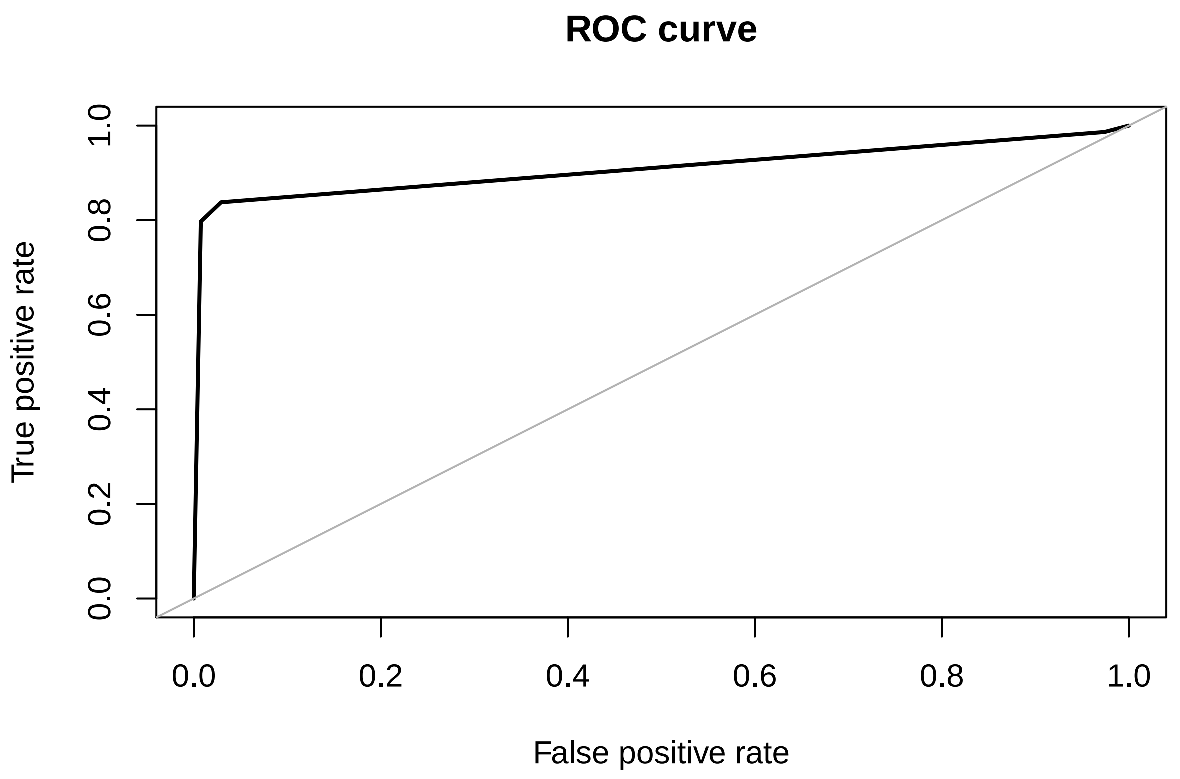
Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	**0.935**

3.1.4 CART Method: Calculate AUC using SMOTE sampled Train Dataset

```
set.seed(42)
# train the models
smote_fit <- rpart(Class ~ ., data = smote_train)
pred_smote <- predict(smote_fit, newdata = test)
print('Fitting smote model to test data')
```

```
## [1] "Fitting smote model to test data"
```

```
#try and predict an outcome from the test set
ROSE::roc.curve(test$Class, pred_smote[,2], plotit = TRUE)
```



```
## Area under the curve (AUC): 0.908
```

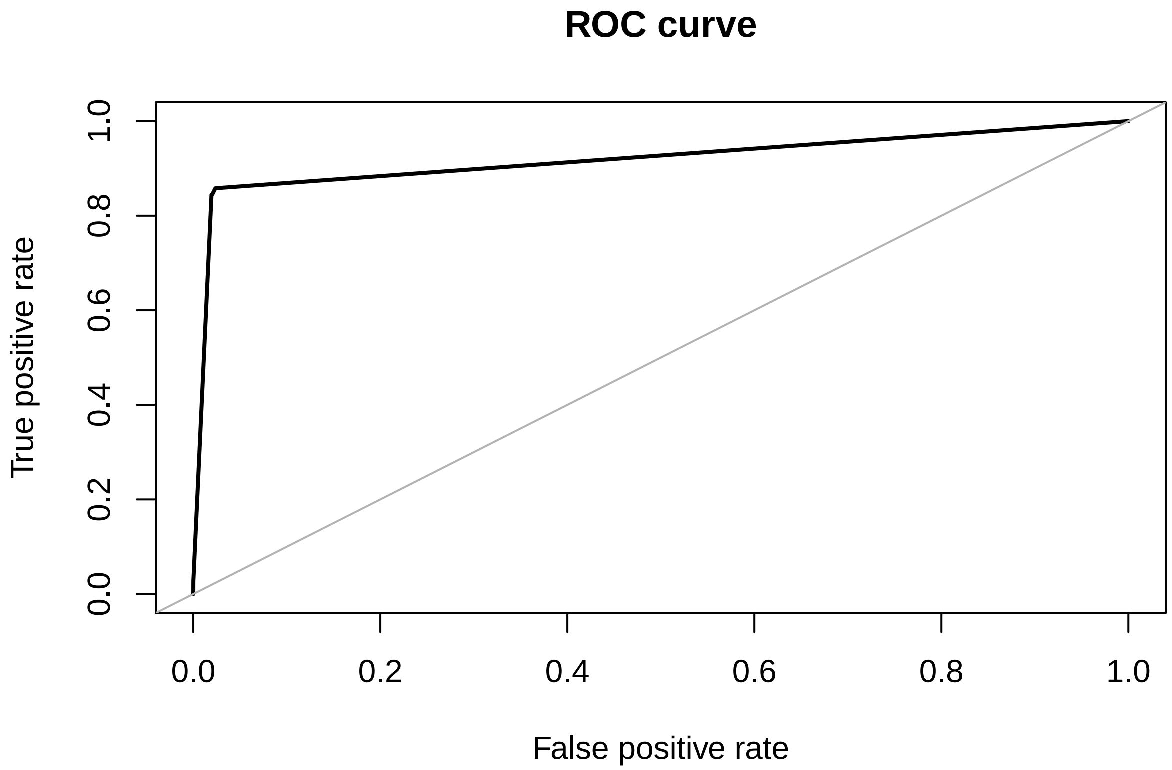
Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	0.935
CART Method: SMOTE	**0.908**

3.1.5 CART Method: Calculate AUC using ROSE sampled Train Dataset

```
set.seed(42)
# # train the models. Build rose model
rose_fit <- rpart(Class ~ ., data = rose_train)
pred_rose <- predict(rose_fit, newdata = test)
print('Fitting rose model to test data')
```

```
## [1] "Fitting rose model to test data"
```

```
# try and predict an outcome from the test set
ROSE::roc.curve(test$Class, pred_rose[,2], plotit = TRUE)
```



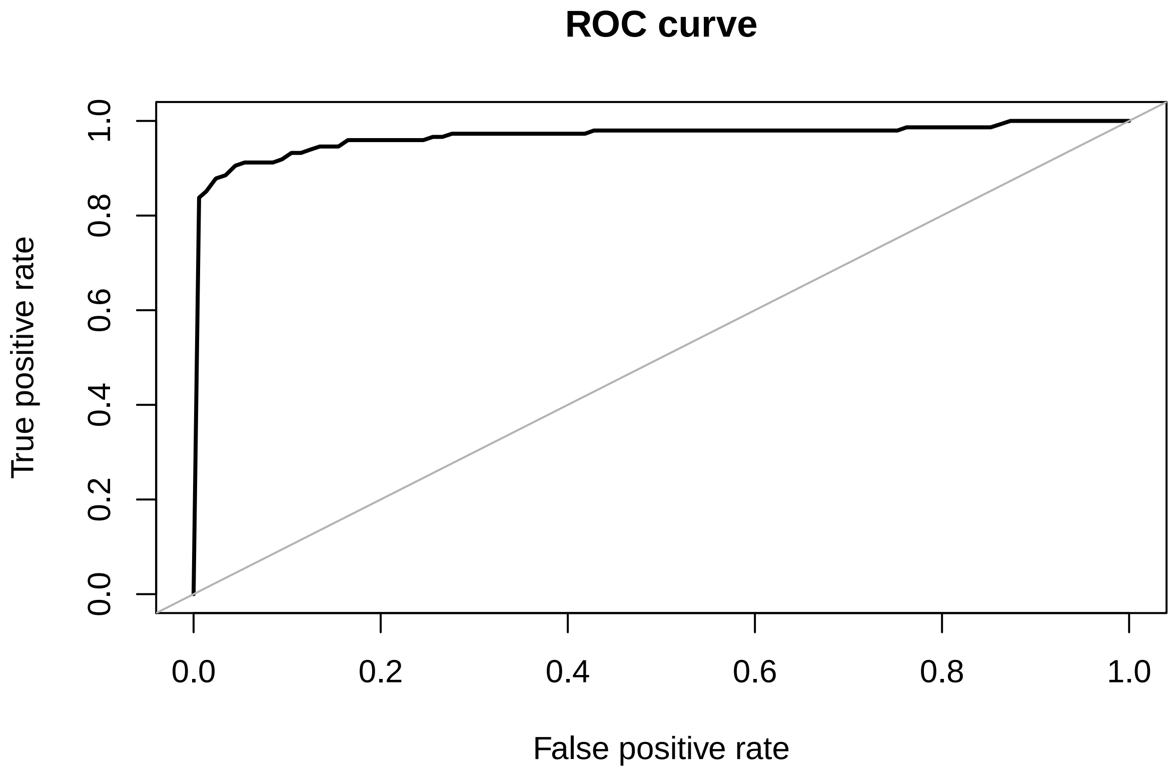
```
## Area under the curve (AUC): 0.919
```

Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	0.935
CART Method: SMOTE	0.908
CART Method: ROSE	**0.919**

We see that all the sampling techniques have yielded better AUC scores than the simple imbalanced dataset. We will test different models now using the up sampling technique as that has given the highest AUC score.

3.2 Logistic Regression (GLM) Model

```
# train the models
fit_glm <- glm(Class ~ ., data = upsamp_train, family = 'binomial')
predict_glm <- predict(fit_glm, newdata = test, type = 'response')
# try and predict an outcome from the test set
ROSE::roc.curve(test$Class, predict_glm, plotit = TRUE)
```

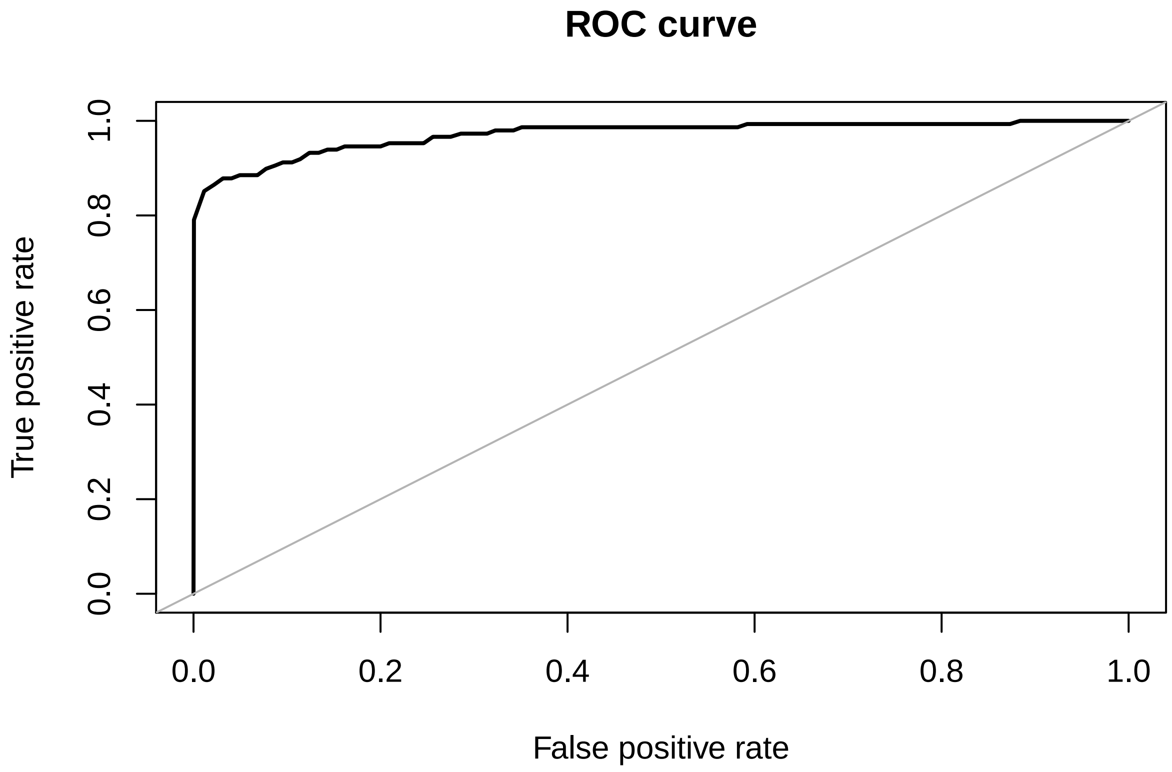


```
## Area under the curve (AUC): 0.967
```

Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	0.935
CART Method: SMOTE	0.908
CART Method: ROSE	0.919
GLM Fit	**0.967**

3.3 Random Forest (RF Fit) Model

```
# train the models
x = upsamp_train[,-30]
y = upsamp_train[,30]
fit_rf <- Rborist(x, y, ntree = 1000, minNode = 21, maxLeaf = 12)
predict_rf <- predict(fit_rf, test[,-30], ctgCensus = "prob")
prob <- predict_rf$prob
# try and predict an outcome from the test set
ROSE::roc.curve(test$Class, prob[,2], plotit = TRUE)
```



```
## Area under the curve (AUC): 0.971
```

Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	0.935
CART Method: SMOTE	0.908
CART Method: ROSE	0.919
GLM Fit	0.967
RF Fit	**0.971**

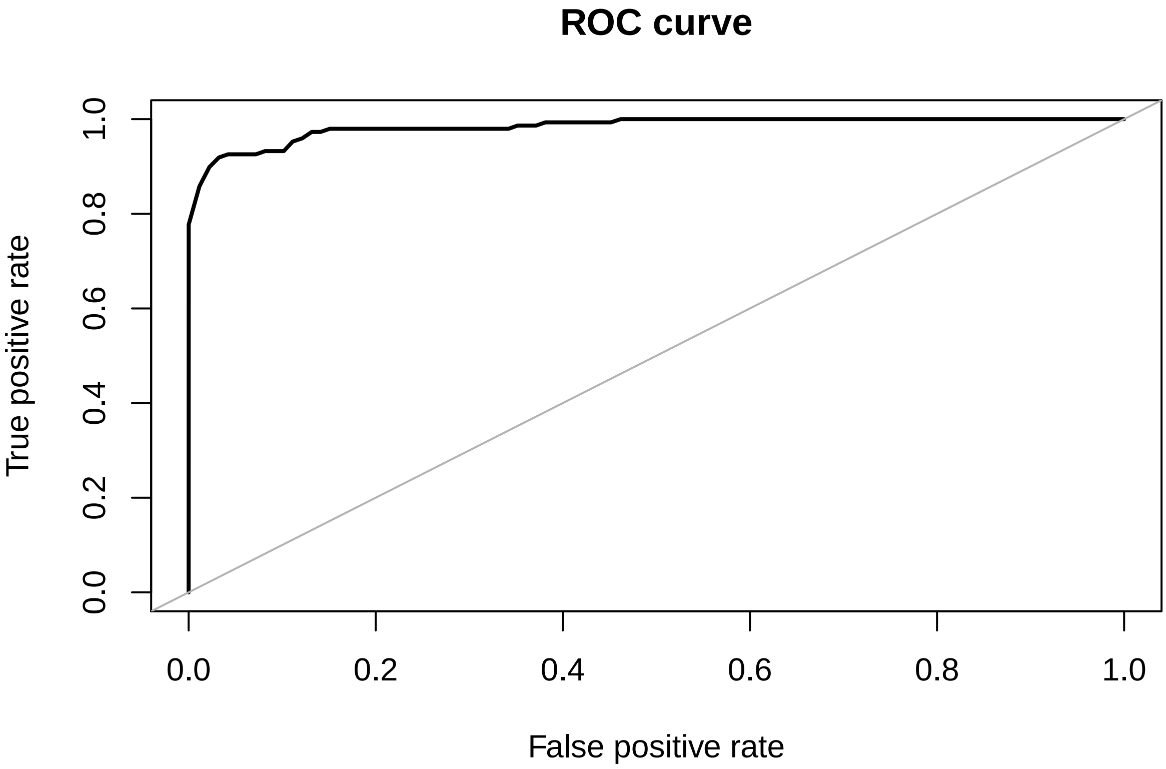
3.4 xgboost (XGB Fit) Model

Lastly, we can also try XGBoost, which is based on Gradient Boosted Trees and is a more powerful model compared to both Logistic Regression and Random Forest. Now that we have seen how to evaluate models on this dataset, let's look at how we can use a final model to make predictions.

```
# train the models
labels <- upsamp_train$Class
y <- recode(labels, "Legal" = 0, "Fraud" = 1)
xgb <- xgboost(data = data.matrix(upsamp_train[,-30]),
 label = y,
 eta = 0.1,
 gamma = 0.1,
 max_depth = 10,
 nrounds = 400,
 objective = "binary:logistic",
 colsample_bytree = 0.5,
 verbose = 0,
 nthread = 8,
 seed = 42
)
```

```
## [21:01:40] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
```

```
predict_xgb <- predict(xgb, data.matrix(test[,-30]))
#try and predict an outcome from the test set
ROSE::roc.curve(test$Class, predict_xgb, plotit = TRUE)
```

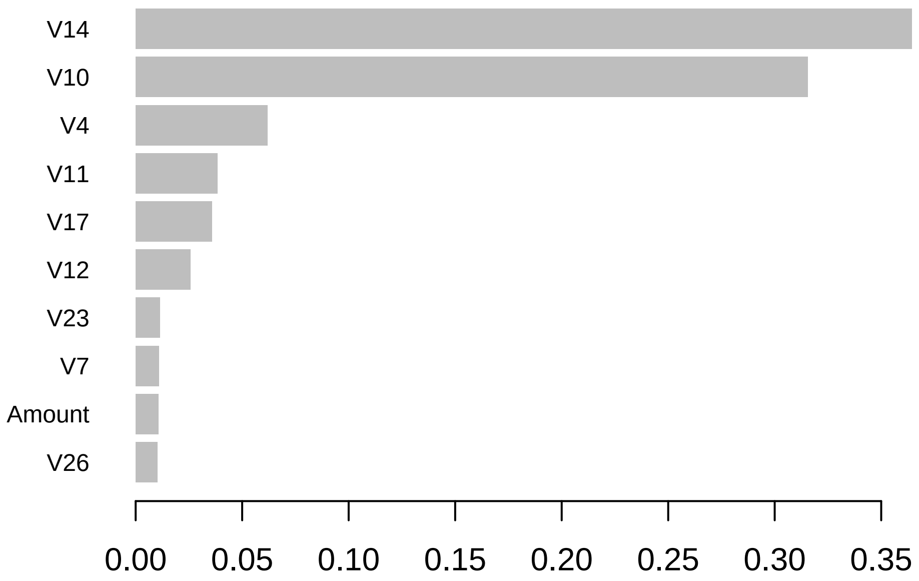


```
## Area under the curve (AUC): 0.984
```

Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	0.935
CART Method: SMOTE	0.908
CART Method: ROSE	0.919
GLM Fit	0.967
RF Fit	0.971
XGB Fit	**0.984**

XGB can also automatically provide estimates of feature importance from a trained predictive model. Feature importance scores can provide insight into the dataset. This means we could save disk space and computation time by only training the model on the most correlated/important variables.

```
names <- dimnames(data.matrix(upsamp_train[,-30]))[[2]]
importance_matrix <- xgb.importance(names, model = xgb)
xgb.plot.importance(importance_matrix[1:10,])
```



4 Results

We see that all the sampling techniques have yielded better AUC scores than the simple imbalanced dataset. We then tested different models using the up sampling technique as that has given the highest AUC score. With an auc score of 0.984 the XGBOOST model has performed the best though both the random forest and logistic regression models have shown reasonable performance. This also indicated that ROC Curve AUC has been a good performance metric as it allow us to compare variety of Machine Learning algorithms to achieve AUC closer to 1. It has been shown that even a very simple logistic regression model can achieve good result, while a much more complex Random Forest model improves upon logistic regression in terms of AUC. However, XGBoost model improves upon both models.

Method	AUC
CART Method: Original	0.898
CART Method: Undersampling	0.913
CART Method: Oversampling	0.935
CART Method: SMOTE	0.908
CART Method: ROSE	0.919
GLM Fit	0.967
RF Fit	0.971
XGB Fit	**0.984**

5 Conclusion

This project has explored the task of identifying fraudulent transactions based on a dataset of anonymized features. We have studied sampling as a way to deal with unbalanced datasets and discussed why accuracy is not an appropriate measure of performance and we used the metric AUC ROC to evaluate how sampling can lead to better training. We concluded that the oversampling technique works best on the dataset and has achieved significant improvement in model performance over the imbalanced data. The best score of 0.984 was obtained by XGBOOST model. RF and Logistic models also performed fairly well. using an XGBOOST model though both random forest and logistic regression models performed well too. This model has many valid real-world use cases. For example, a bank could take a similar approach and reduce the amount of money spent trying to detect fraud by automating it with a machine. We can protect consumers with this technology by having a similar model integrated into the transaction process to notify the consumer and bank of fraud within minutes, rather than days. This project has explored the task of identifying fraudulent transactions based on a dataset of anonymized features.

A future task to explore is to focus on two metrics: ROC Curves and Precision-Recall Curves to use on an imbalanced dataset and compare the AUC score. Learn more about, does it make sense to create balance set or use tools like Precision Recall on an imbalanced data.

6 Sources

- [1]. Shift Credit Card Processing incorporate. “Credit Card Fraud Statistics”, Updated January 2021, https://shiftprocessing.com/credit-card-fraud-statistics/
- [2]. Brownlee, Jason. “8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.”, Machine Learning Mastery, 19 August 2015, https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
- [3]. Machine Learning Group - ULB. “Credit Card Fraud Detection Anonymized credit card transactions labeled as fraudulent or genuine”, Kaggle.com, 22 March 2018, Version 3, https://www.kaggle.com/mlg-ulb/creditcardfraud
- [4]. Brownlee, Jason. “How to Use ROC Curves and Precision-Recall Curves for Classification in Python”, Machine Learning Mastery, 13 January 2021, https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

7 Definition of Terms

- **False Positive.** Predict an event when there was no event.
- **False Negative.** Predict no event when in fact there was an event.
- **ROC Curves.** summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
- **Precision-Recall curves.** summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.
- **PCA.** Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
- **CART.** The classical name Decision Tree and the more Modern name CART for the algorithm. The representation used for CART is a binary tree. Predictions are made with CART by traversing the binary tree given a new input record. The tree is learned using a greedy algorithm on the training data to pick splits in the tree.
- **GLM.**Generalized linear models generalize the possible distributions of the residuals to a family of distributions called the exponential family. This family includes the normal as well as the binomial, Poisson, beta, and gamma distributions, among others.
- **RF.** Random forest is a supervised learning algorithm. The “forest” it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.
- **XGBOOST.** XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.