2020

Credit Card Fraud Detection

Table of Contents

1. Introduction

Credit card fraud has become as major problem in the electronic payment sector. The goal of this study is to build a classifier that tells if a transaction is a fraud or not. One of the challenges we are facing working with this dataset is highly unbalanced data with 284315 (99.8%) observations were identified as non-fraud transactions and 492 (0.2%) observations were identified as fraud transactions.

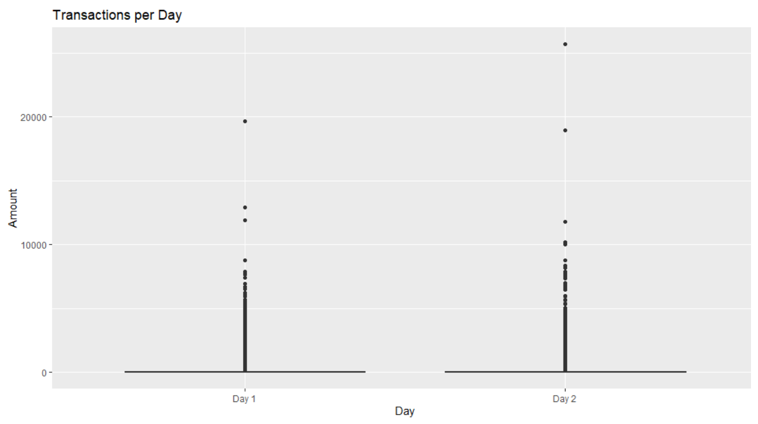
The dataset is the Kaggle Credit Card Fraud Detection dataset here. It contains two-day transactions made on 09/2013 by European cardholders. There are 284807 observations with 31 columns. After some investigations, we found that it only contains numerical variables. Features V2, V3, … V29 are the principal components obtained with PCA transformation. The only features which have not been transformed are 'Time' and 'Amount'. 'Time' is the seconds elapsed between each transaction and the first. 'Amount' is the transaction amount. 'Class' is the response variable with 1 as fraud and 0 otherwise.

2. Data Transformation &Visualization

1.1 Exploratory Data Analysis

The dataset contains 31 variables naming V1:V31. At first, we rename columns V1, V30, V31 to Time, Amount & Class respectively, while features V2:V29 remain as principle components obtained with PCA transformation. Since Feature 'Time' is recorded in the number of seconds since the first transaction, it appears that this data set includes all transactions recorded over the course of two days.

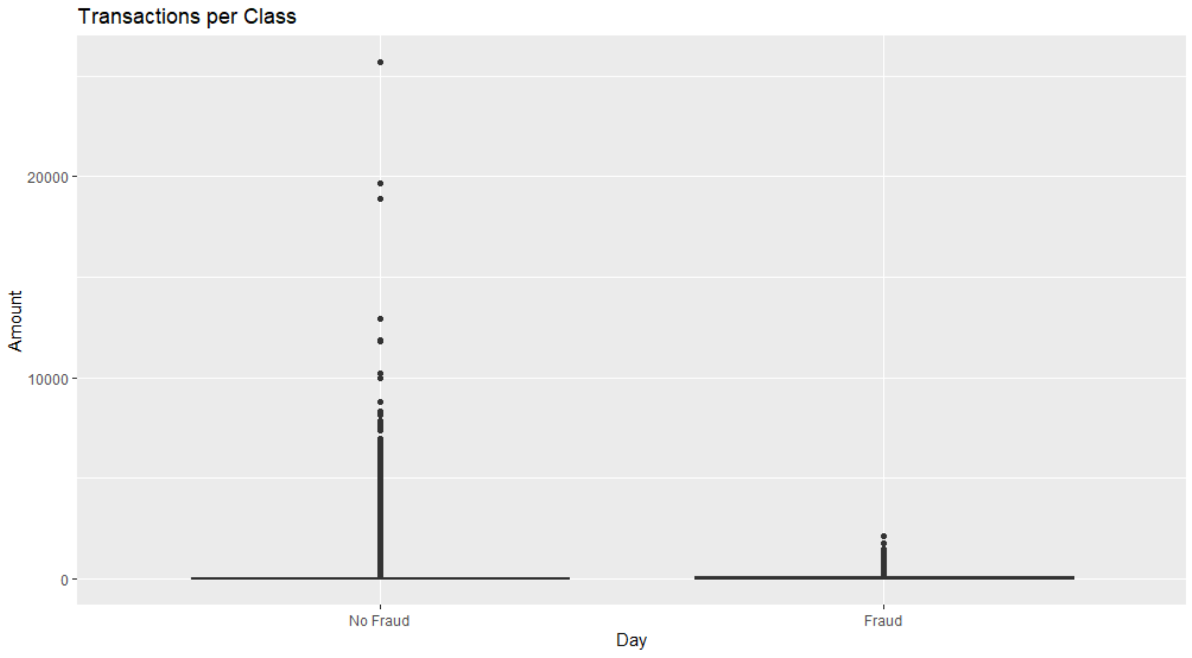
The mean value of all transactions is $88.35 while the highest transaction in this data set is $25,691.16. Based on the mean & maximum transactions, the distribution of all transaction values is highly skewed. New feature "Day" was created to see if there were any irregularities found in during this time. Furthermore, there were no missing values in the data set. It looks like there were more variability in transaction values for Day 2.



Day	Average	Median	Min	Max
1	90.4	23.4	0	19657
2	86.2	20.3	0	25691

The transaction values in Non-fraudulent class was much higher than Fraud class and there was more variability in non-fraudulent transactions. Luckily the fraudulent transaction amount was not high compared to non-fraudulent transactions.

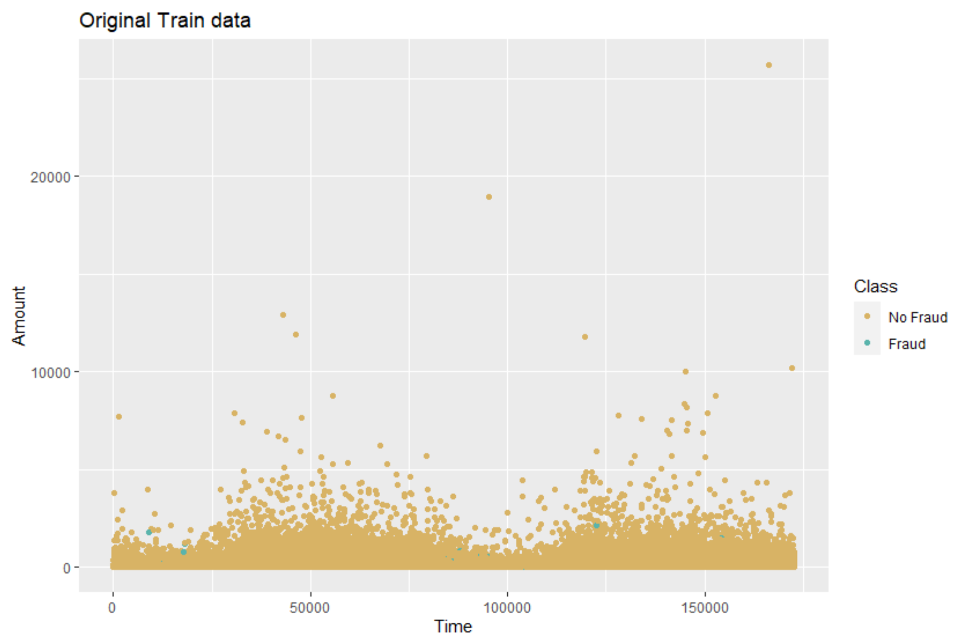
Class	Mean	Median	Min	Max
No Fraud	88.3	22	0	25691
Fraud	122	9.25	0	2126



3. Section 3: Machine Learning Algorithm

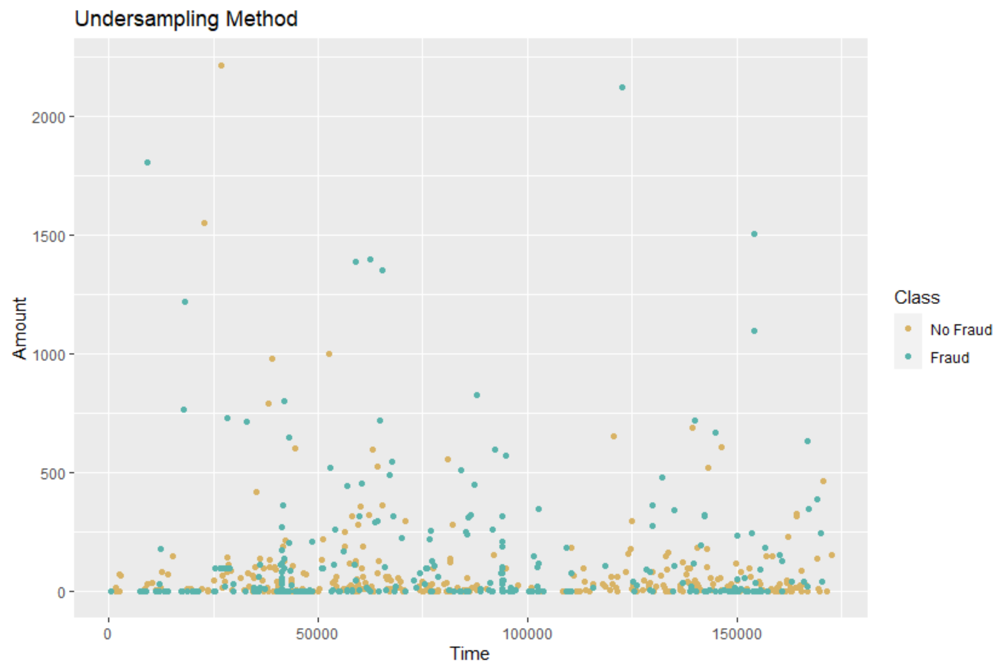
1.2 Balancing imbalanced data.

Imbalance data is a case where the classification dataset class has a skewed proportion. The “Original train data” chart shows that most of the observations in this dataset are non-fraud transactions. Since imbalance class creates a bias where the machine learning model tends to predict the majority class which is non-fraudulent transactions in this case. The original train dataset has the lowest AUC score in this study which is 0.889.

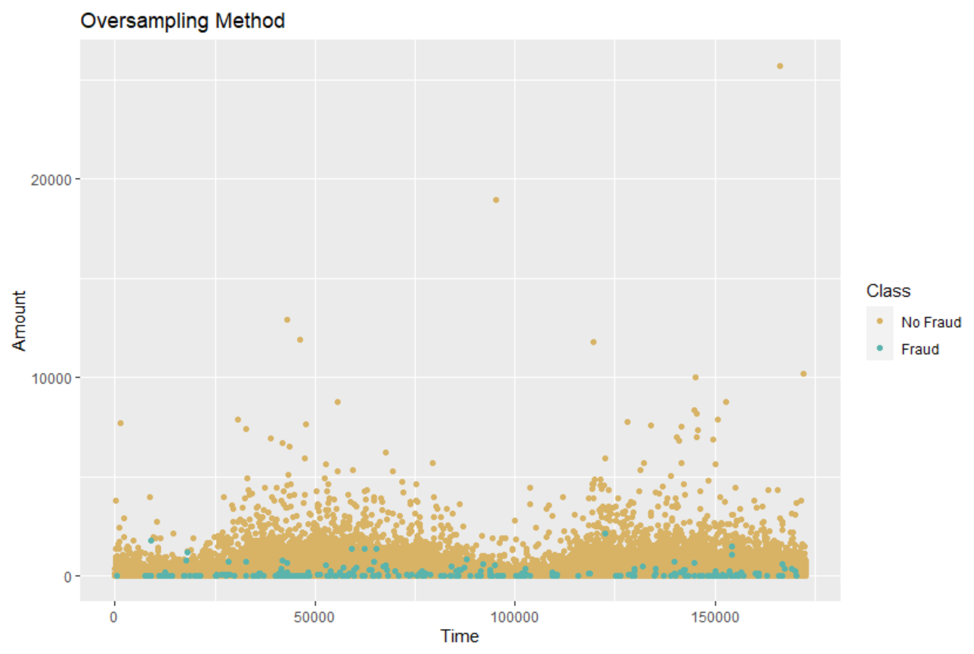


In this study, we used 4 techniques to handle imbalanced data.

1. UnderSampling: remove observations from the training dataset that belong to the majority class in order to better balance the class distribution. This method is also useful when we have a large dataset and it helps to reduce the number of training samples in order to improve run time and storage troubles. However, using this method may cause training dataset to lose important information pertaining to majority class.

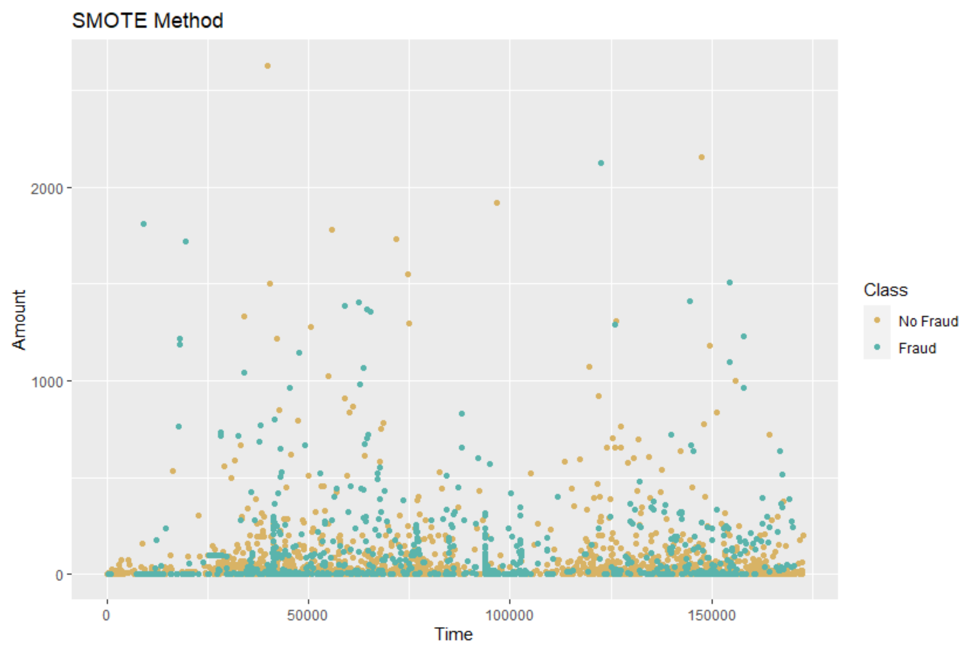


2. OverSampling: this method is an opposite of UnderSampling where it works with smaller group. This method is used to randomly oversampling the minority class. However, this might lead to overfitting problem and reduce run time as well as storage troubles.



3. SMOTE - Synthetic Minority Oversampling Technique

Smote is an oversampling technique that generates synthetic samples from the minority class, and it is one of the most commonly used oversampling methods to solve the imbalance problem.

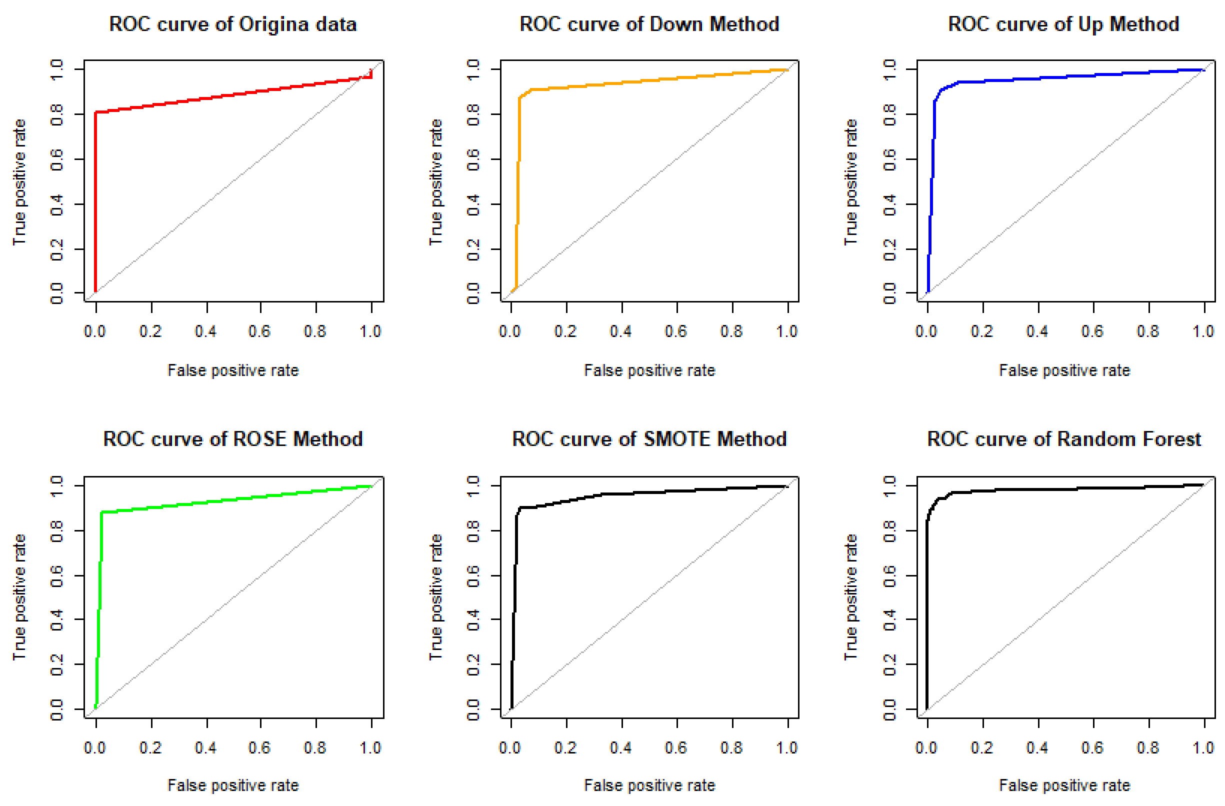


It shows that the data generated original train data without sampling giving the lowest AUC score of 0.889, while UpSampling and SMOTE method both have the highest accuracy score of AUC which is 0.951. Therefore, SMOTE method dataset will be chosen to predictive models. In this study, SMOTE method has the highest AUC score among other techniques (AUC = 0.951).

1.3 Decision Tree

Before proceeding to the fitting process, we split the train set data into two chunks: 70% of observations will be included in training set & 30% of observations will be in testing set. The training set will be used to fit the predictive model which will be used testing over the testing set.

All data in training set was used to fit the decision tree model. It appears the final accuracy score using ROC AUC = 0.889 is not good enough. Therefore, this model is not worth to use, and the data need to be balanced before applying a machine learning algorithm.



It shows that the data generated from DownSampling, UpSampling, SMOTE method provide better accuracy scores compared to Imbalanced data. Therefore, SMOTE method dataset was chosen to build Random Forest Model to see whether this model will give us a better accuracy result.

2.3 Random Forest

By default, the number of decision trees in the forest is 500 and the number of features used as potential candidates for each split is 5. We will use all the Predictors in the dataset.

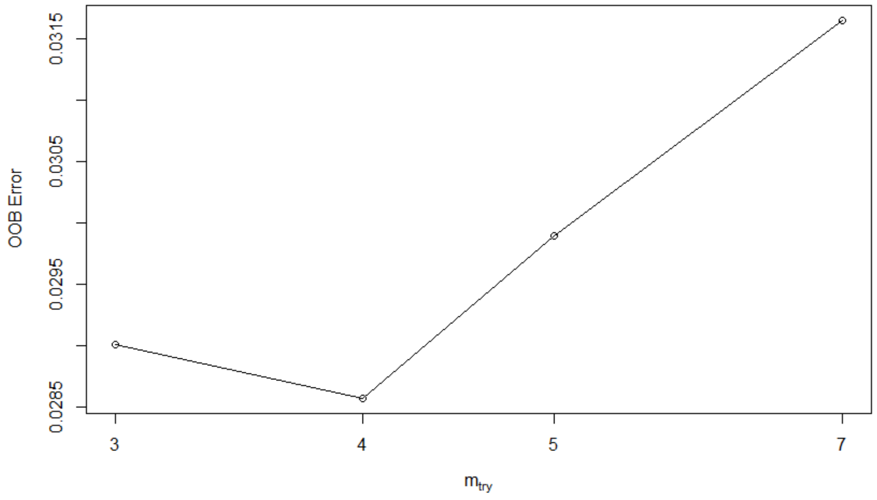
```
Call:
 randomForest(formula = Class ~ ., data = smote_train, ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 2.81%
Confusion matrix:
         No Fraud Fraud class.error
No Fraud     1287    13  0.01000000
Fraud          51   924  0.05230769
```

We then use the model to predict whether the transaction is fraudulent or non-fraudulent.

```
rf.pred <- predict(rf.fit,newdata = test,type="prob")
```

The number of variables randomly selected at each split is 5. After searching for the optimal value (with respect to Out-of-Bag error estimate) of mtry for randomForest, the best mtry is 4.

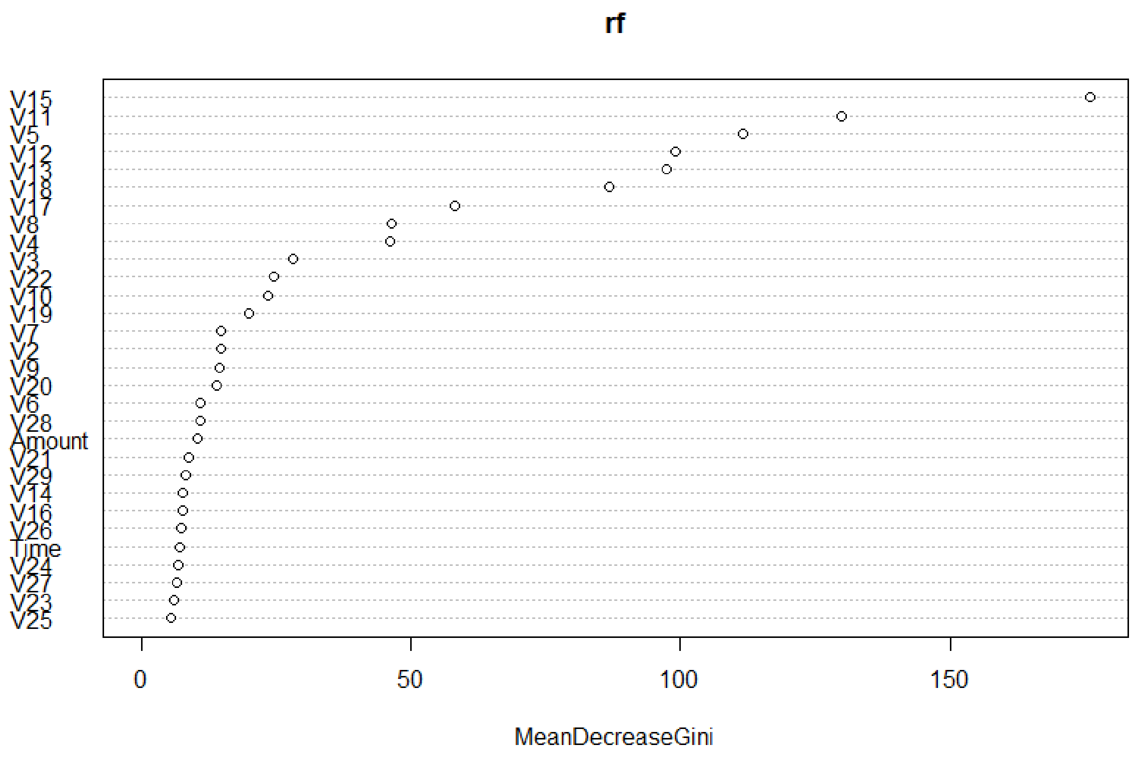


The error tends to be minimized at around mtry = 4. We then built the model again using best mtry value.

```
Call:
 randomForest(formula = Class ~ ., data = smote_train, ntree = 500,      mtry = best.m)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 2.64%
Confusion matrix:
         No Fraud Fraud class.error
No Fraud     1292     8 0.006153846
Fraud          52   923 0.053333333
```

In the plot shown below, V15 is the most importance variable following by V11, V5, V12, etc


rf
V15
V11
V5
V12
V13
V18
V17
V8
V4
V3
V22
V10
V19
V7
V2
V9
V20
V6
V28
Amount
V21
V29
V14
V16
V26
Time
V24
V27
V23
V25
0
50
100
150
MeanDecreaseGini