Analyzing the Effectiveness of Data Set Balancing Techniques in Classifications for Detecting Truly Fraudulent Credit Card Transactions

In partial fulfillment for the Data Science Professional Certification from Harvard University

Arturo P. Caronongan III

July 16, 2020

Abstract

This project is the second of two deliverables for Harvard University's Data Science Professional Certification under the course PH125.9x Data Science: Capstone. In this project, the ULB dataset containing 284,807 transaction records will be used to classify fraudulent and non-fraudulent credit card transactions. The aforementioned dataset is heavily unbalanced, with a very large bias towards the amount of non-fraudulent transactions, which make up 99.83% of the data set. Different data balancing techniques will be performed in different circumstances and the impact to the resulting classifier model of these techniques will be analyzed.

Contents

1.0 Introduction and Overview of the Problem

Credit Card fraud is a problem that exists in today's technologically savy world where digital shopping, online transactions, and anything related to handling transactions involving money is sgrowing each day. The issue is that the norm for completing online transactions normally require the use of credit cards or online payment services like paypal in completing these transactions.

As these transactions are done over the web along with the existence of security flaws due to network vulnerables being discovered, it is inevitable that sometimes sensitive data of certain individuals can get compromised. This has often led to data theft, and more particularly credit card fraud.

Credit card fraud is defined as a type of identity theft which involves the process of completing a purchase with another individual's credit card information without proper authorization from the said individual. It is a very costly in a way that it costs Australia and the 1.6 Billion USD per year and UK £844.8 in the year 2018 alone.[1,2] From 2016 to 2025, the US-based research agency projected the fraud losses will nearly double, climbing from 22.8 billion to nearly 50 billion USD. [3]

Due to the huge implications, both in terms of the economic and mental well-being of individuals affected by fraudulent transactions, research has been done with regards to detecting fraudulent cases. Several studies have been done with regards to classifying Fraudulent Cases using Decision trees. However, the dataset used in certain studies have led to certain issues, particularly with regards to an imbalance that exists within the instances of fraudulent and non-fraudulent transactions due to the sheer volume of transactions that occur over time and the prevalence of non-fraudulent transactions compared to the prevalence of fraudulent transactions due to an increase in data security.

1.1 ULB Credit Card Dataset

The ULB dataset used for this study contains transactions made by credit cards in by european cardholders that occured in September 2013 over a span of two days. The data set recorded a total of 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, where the positive class (frauds) account for 0.172% of all transactions.

The dataset contains only numerical input variables which are the result of a PCA transformation due to consumer confidentiality. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

1.2 Approach and Objective

As mentioned in the previous section, the dataset is very unbalanced which can prove difficult for classification. However, there exists techniques for data balancing, techniques such as Random Oversampling (ROS), Random Undersampling (RUS), both Random OverSampling & Random Undersampling (ROS + RUS), and Synthetic Minority Oversampline Technique (SMOTE). Several of the studies mentioned in section 1.0 made use of Classification Trees as algorithms to distinguish the difference between Fraudulent Transactions and Non-Fraudulent transactions. Many of their recommendations indicated the need for a balanced dataset, when identifying the Fraudulent cases (or in the case of the dataset, True-Negatives).

This study will make use of sampling techniques and observe the impact of performing sampling techniques with regards to determining the role of data balancing techniques in determining the True-Negatives with regards to detecting fraudulent transactions while minimizing the impact in True Positive values. There will be two approaches that will be experimented on, where the first will experiment on taking a very small subset of the dataset and applying the data balancing technique on that small subset only. The second approach will examine the impact of applying data balancing before obtaining the training and the testing dataset to the classification algorithm.

2.0 Preliminary Data Analysis

The dataset has been mentioned to be unbalanced, with the distributon count heavily biased towards the number of non-fraudulent cases. We will confirm this statement using preliminary analysis of the data.

```
str(credit_card)
```

```
## '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 ...
```

```
head(credit_card)
```

```
##   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
```

```
summary(credit_card)
```

```
##       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
```

We can clearly see that there is approximately 284807 rows and 31 columns in our dataset. With the sheer volume of data, it is important that we check if there are any missing values so we can see if we will need to perform additional data pre-processing.

```
sum(is.na(credit_card))
```

```
## [1] 0
```

Fortunately, there are no missing values present.

For pre-processing the data, we are going to convert the Class attribute into a factor. Currently, the `credit_card$Class` data is of `int` type. As a regression type analysis will not be used for this experiment, it will be converted into a factor so that the data set may be used for classification. This is done by executing the following script.

```
credit_card$Class <- factor(credit_card$Class, levels = c(0,1))
```

We can now observe how unbalanced the dataset is. The dataset indicates that transactions labelled with a Class value of 0 are considered as non-fraudulent transactions while transactions labelled with a class value of 1 are deemed as fraudulent transactions.

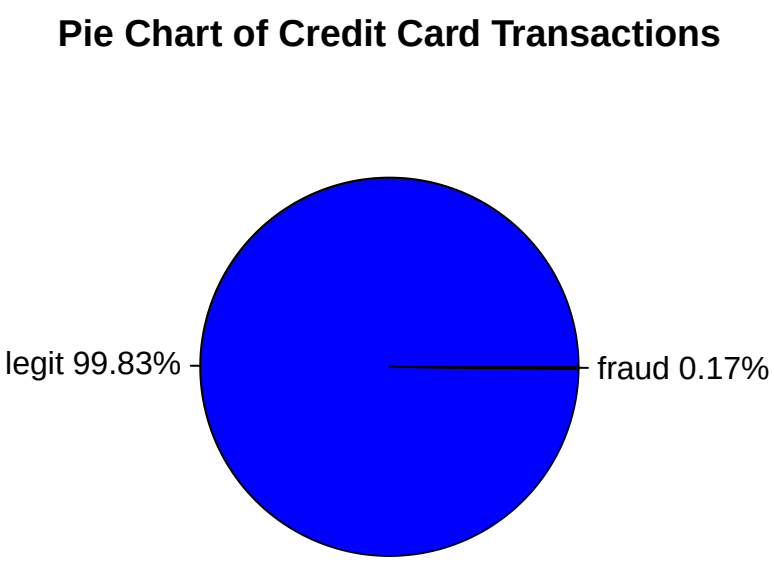
```
#get the distribution of fraud and legit transactions in the dataset
table(credit_card$Class)
```

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

```
#get the percentage of fraud and legit transactions in the dataset
prop.table(table(credit_card$Class))
```

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

```
#Pie chart of credit card transactions
labels <- c("legit", "fraud")
labels <- paste(labels, round(100*prop.table(table(credit_card$Class)),2))
labels <- paste0(labels, "%")

pie (table(credit_card$Class), labels, col = c("blue","red"),
     main = "Pie Chart of Credit Card Transactions")
```



It is now painfully obvious that the dataset is heavily unbalanced. The data shows that there are 284315 cases of non-fraudulent transactions compared to only 492 cases recorded in the dataset. This means that the data set is comprised of 99.83% of non-fraudulent transactions and only 0.17% of fraudulent transactions.

To visualize the effect of an extremely unbalanced dataset towards a classification model, we will attempt to create a baseline model that will predict that all transactions are non-fraudulent cases using a ZeroR classifier.

```
predictions <- rep.int(0,nrow(credit_card))
predictions <- factor(predictions, levels = c(0,1))
```

We will then generate the confusion matrix of the generated model.

```
confusionMatrix(data = predictions, reference = credit_card$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 284315    492
##          1      0      0
##
##                Accuracy : 0.9983
##                  95% CI : (0.9981, 0.9984)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 0.512
```

```
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9983
##          Neg Pred Value :    NaN
##              Prevalence : 0.9983
##          Detection Rate : 0.9983
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

Again, the 99.83% accuracy pertains to the data being comprised of 99.83% of non-fraudulent cases. This shows that relying on classification accuracy is not reliable under these circumstances. In this scenario, we classified even the fraudulent cases as non-fraudulent transactions which still led to the very high accuracy. Despite that, we can say that the 492 cases of fraudulent transactions could have a big economic impact.

Since classification is a big issue due to the imbalanced data set, this experiment will be conducted to see how effective data balancing techniques are in improving classifications for detecting truly fraudulent cases.

As a result, the aim of this study is to examine the effectivity of these data set balancing methods, such as the Random Oversampling technique (ROS), the Random Undersampling Technique (RUS), ROS and RUS combined (ROS+RUS), and the Synthetic Minority Oversampling Technique (SMOTE) in maximizing the True Negative (Truly Fraudulent cases) value obtained while minimizing the decrease in the True Positive Rate (Truly Non-Fraudulent cases) value obtained by Classification Trees.

3.0 Applying Data Balancing Techniques to a Subset of the Dataset

For faster processing, this section of the experiment will use a subset of the dataset. To further emphasize data balancing, 10% of the data is going to be extracted.

```
set.seed(1)
credit_sample = sample.split(credit_card$Class,SplitRatio=0.90) #Take 10% of the dataset

final_credit_sample = subset(credit_card,credit_sample == TRUE)
sub_credit_card = subset(credit_card,credit_sample == FALSE) #sub_credit_card will be 10%
#of the dataset
```

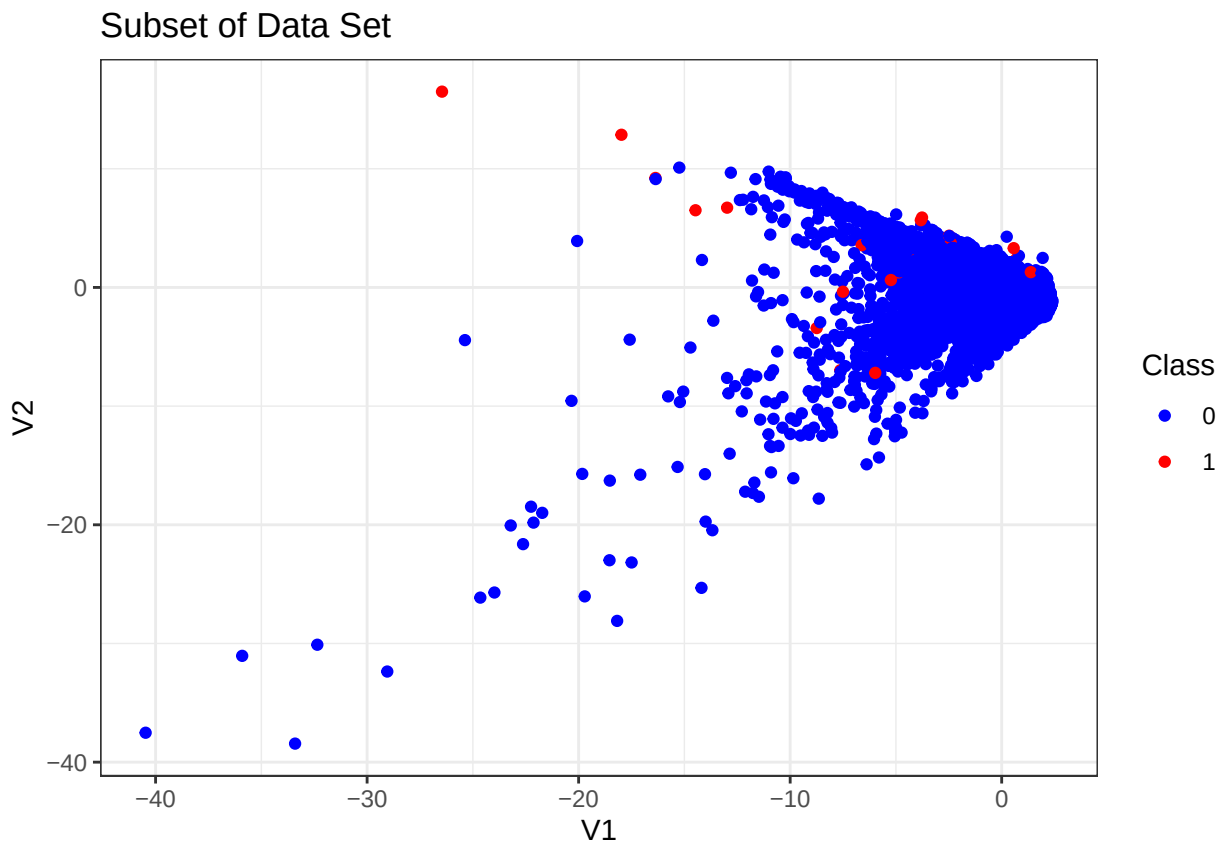
We will take analyze the resulting data set that was obtained. Again, we can see that the data is imbalanced.

```
table(sub_credit_card$Class)
```

```
##
##     0     1
## 28431    49
```

We can see the distribution of Fraudulent Cases to Non-Fraudulent cases taking the V1 and V2 variables in relation with each other in our new data set.

```
ggplot(data = sub_credit_card, aes(x = V1, y = V2, col = Class)) +
  geom_point() +
  ggtitle("Subset of Data Set") +
  theme_bw() +
  scale_color_manual(values = c('blue', 'red'))
```



We will further generate a test set and a training set from this subset.

```
set.seed(123)

data_sample = sample.split(sub_credit_card$Class,SplitRatio=0.80)

train_data = subset(sub_credit_card,data_sample == TRUE)
test_data = subset(sub_credit_card,data_sample == FALSE)
```

Our final hypothetical data sets will now consist of the following.

First, the training data set's distribution, value summaries and visualization:

```
summary(train_data)
```

```
##       Time              V1                   V2                   V3
##  Min.   :     1   Min.   :-40.47014   Min.   :-37.52043   Min.   :-30.177317
##  1st Qu.: 53654   1st Qu.: -0.91799   1st Qu.: -0.60147   1st Qu.: -0.887750
##  Median : 85010   Median :  0.01263   Median :  0.06623   Median :  0.175749
```

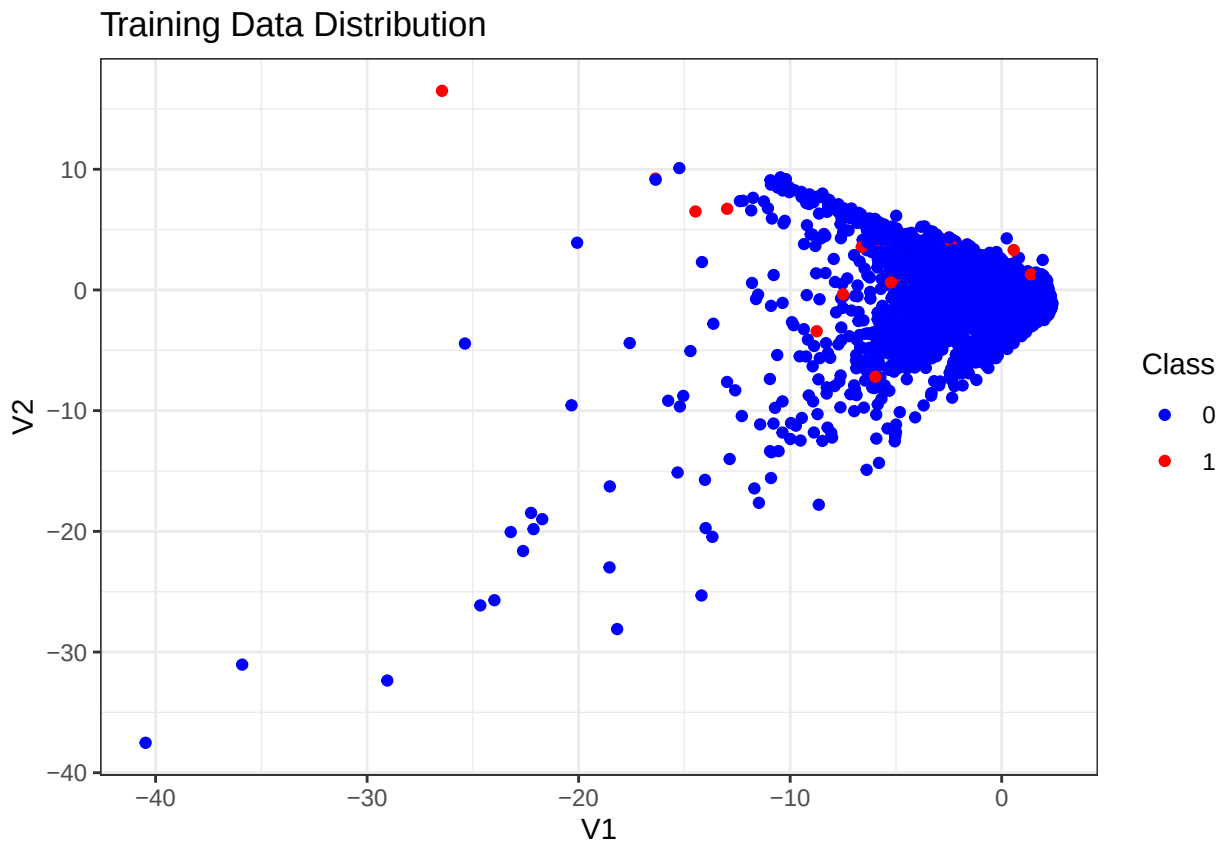
```
##  Mean   : 94566   Mean   :  0.00612   Mean   : -0.00585   Mean   :  0.003258
##  3rd Qu.:139094   3rd Qu.:  1.31786   3rd Qu.:  0.79988   3rd Qu.:  1.020340
##  Max.   :172769   Max.   :  2.40466   Max.   : 16.49747   Max.   :  3.934998
##        V4                  V5                  V6
##  Min.   :-5.071241   Min.   :-31.356750   Min.   :-8.998368
##  1st Qu.:-0.856658   1st Qu.: -0.692601   1st Qu.:-0.768587
##  Median :-0.024544   Median : -0.041111   Median :-0.277036
##  Mean   :-0.005847   Mean   :  0.008527   Mean   : 0.000992
##  3rd Qu.: 0.734841   3rd Qu.:  0.614709   3rd Qu.: 0.399919
##  Max.   :11.844777   Max.   : 15.848810   Max.   :20.379524
##        V7                  V8                  V9
##  Min.   :-31.197329   Min.   :-27.156906   Min.   :-9.462573
##  1st Qu.: -0.559609   1st Qu.: -0.204475   1st Qu.:-0.641320
##  Median :  0.038160   Median :  0.026772   Median :-0.042925
##  Mean   : -0.003984   Mean   :  0.001242   Mean   : 0.003455
##  3rd Qu.:  0.565115   3rd Qu.:  0.330614   3rd Qu.: 0.599951
##  Max.   : 30.897666   Max.   : 10.724728   Max.   : 9.272376
##       V10                 V11                 V12
##  Min.   :-22.187089   Min.   :-4.22479   Min.   :-15.022700
##  1st Qu.: -0.545047   1st Qu.:-0.75075   1st Qu.: -0.426439
##  Median : -0.093242   Median :-0.02596   Median :  0.127769
##  Mean   : -0.005575   Mean   : 0.00879   Mean   : -0.008596
##  3rd Qu.:  0.446802   3rd Qu.: 0.74417   3rd Qu.:  0.622619
##  Max.   : 15.236028   Max.   :10.54526   Max.   :  4.406338
##       V13                 V14                 V15
##  Min.   :-3.617695   Min.   :-15.06637   Min.   :-4.005558
##  1st Qu.:-0.648709   1st Qu.: -0.41436   1st Qu.:-0.583298
##  Median :-0.017636   Median :  0.06262   Median : 0.046254
##  Mean   : 0.000246   Mean   :  0.01542   Mean   : 0.002228
##  3rd Qu.: 0.663282   3rd Qu.:  0.50169   3rd Qu.: 0.653900
##  Max.   : 3.856013   Max.   :  6.99175   Max.   : 5.685899
##       V16                 V17                 V18
##  Min.   :-11.350244   Min.   :-21.710188   Min.   :-8.859452
##  1st Qu.: -0.474017   1st Qu.: -0.480328   1st Qu.:-0.492681
##  Median :  0.067451   Median : -0.062534   Median : 0.001828
##  Mean   :  0.007437   Mean   :  0.005502   Mean   : 0.007325
##  3rd Qu.:  0.533251   3rd Qu.:  0.396489   3rd Qu.: 0.503227
##  Max.   :  7.059132   Max.   :  7.766636   Max.   : 4.071329
##       V19                 V20                 V21
##  Min.   :-4.038451   Min.   :-25.222345   Min.   :-14.031579
##  1st Qu.:-0.448497   1st Qu.: -0.211432   1st Qu.: -0.228404
##  Median : 0.000927   Median : -0.063751   Median : -0.027356
##  Mean   : 0.003183   Mean   : -0.001533   Mean   :  0.002813
##  3rd Qu.: 0.449522   3rd Qu.:  0.131358   3rd Qu.:  0.190352
##  Max.   : 4.475127   Max.   : 15.815051   Max.   : 22.614889
##       V22                 V23                 V24
##  Min.   :-8.593642   Min.   :-20.757358   Min.   :-2.51321
##  1st Qu.:-0.533697   1st Qu.: -0.163126   1st Qu.:-0.35401
##  Median : 0.015145   Median : -0.010590   Median : 0.04150
##  Mean   : 0.008182   Mean   :  0.003183   Mean   : 0.00115
##  3rd Qu.: 0.533589   3rd Qu.:  0.150118   3rd Qu.: 0.43753
##  Max.   : 4.359627   Max.   : 16.722816   Max.   : 3.94924
##       V25                 V26                 V27                 V28
##  Min.   :-4.129885   Min.   :-1.72693   Min.   :-7.263482   Min.   :-8.338123
```

```
##  1st Qu.:-0.321134   1st Qu.:-0.32646   1st Qu.:-0.069971   1st Qu.:-0.053854
##  Median : 0.014445   Median :-0.04956   Median : 0.000887   Median : 0.010375
##  Mean   :-0.001641   Mean   :-0.00008   Mean   : 0.001736   Mean   :-0.002831
##  3rd Qu.: 0.350226   3rd Qu.: 0.23484   3rd Qu.: 0.091028   3rd Qu.: 0.078162
##  Max.   : 4.808122   Max.   : 3.11929   Max.   : 7.126343   Max.   :14.046301
##      Amount         Class
##  Min.   :   0.00   0:22745
##  1st Qu.:   5.38   1:   39
##  Median :  21.90
##  Mean   :  86.74
##  3rd Qu.:  75.86
##  Max.   :7879.42
```

```
table(train_data$Class)
```

```
##
##     0     1
## 22745    39
```

```
ggplot(data = train_data, aes(x = V1, y = V2, col = Class)) +
  geom_point() +
  ggtitle("Training Data Distribution") +
  theme_bw() +
  scale_color_manual(values = c('blue', 'red'))
```



Followed by the test data set's distribution, value summarries, and visualization:

```
summary(test_data)
```

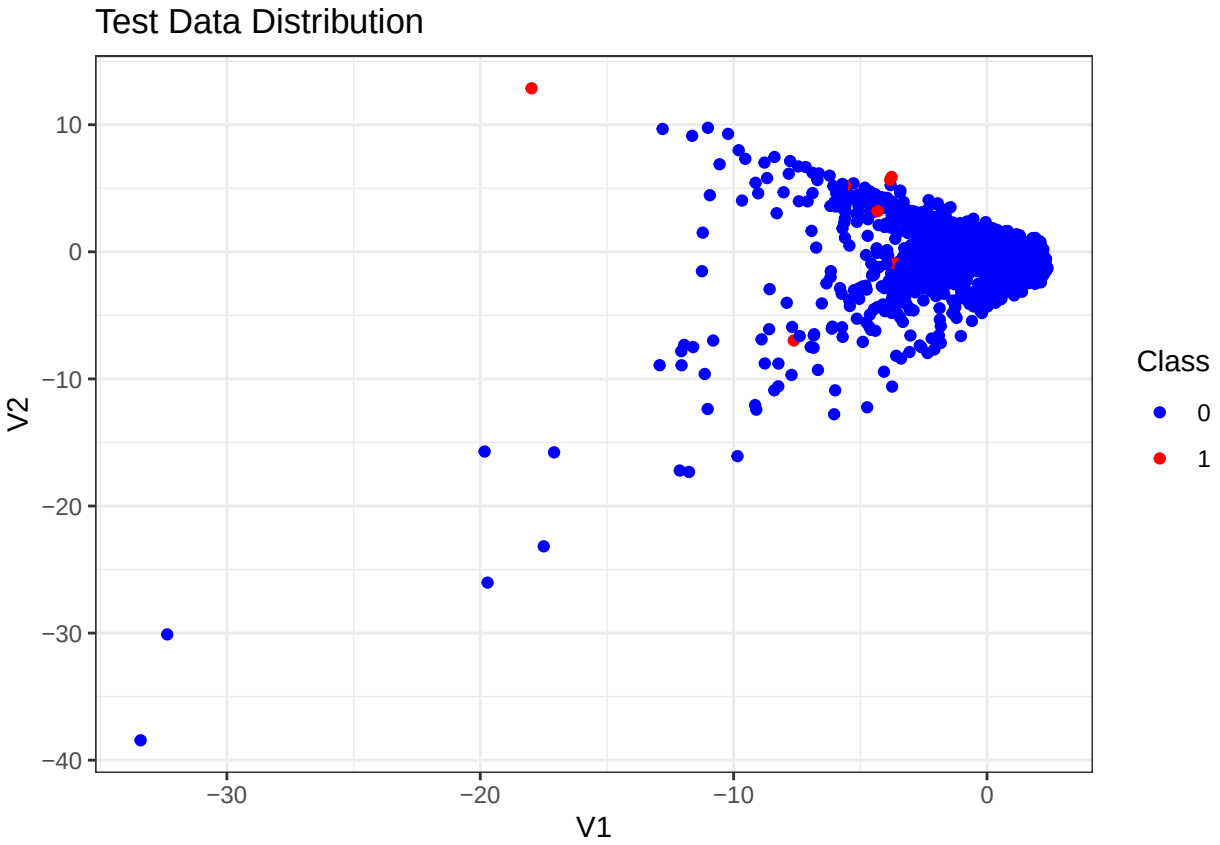
```
##       Time              V1                  V2                  V3
##  Min.   :    16   Min.   :-33.40408   Min.   :-38.43682   Min.   :-19.575066
##  1st Qu.: 54220   1st Qu.: -0.91876   1st Qu.: -0.58967   1st Qu.: -0.895718
##  Median : 85914   Median :  0.01350   Median :  0.07347   Median :  0.168954
##  Mean   : 95395   Mean   : -0.01466   Mean   : -0.01640   Mean   :  0.001624
##  3rd Qu.:140523   3rd Qu.:  1.30491   3rd Qu.:  0.82833   3rd Qu.:  1.030303
##  Max.   :172759   Max.   :  2.38838   Max.   : 12.86499   Max.   :  3.624300
##        V4                  V5                  V6
##  Min.   :-4.354308   Min.   :-32.09213   Min.   :-12.37728
##  1st Qu.:-0.869273   1st Qu.: -0.73529   1st Qu.: -0.77584
##  Median : 0.000680   Median : -0.08599   Median : -0.28471
##  Mean   : 0.002865   Mean   : -0.03623   Mean   : -0.01509
##  3rd Qu.: 0.729853   3rd Qu.:  0.57705   3rd Qu.:  0.40965
##  Max.   :12.699542   Max.   : 17.70941   Max.   : 21.39307
##        V7                  V8                  V9
##  Min.   :-23.78347   Min.   :-50.68842   Min.   :-8.504285
##  1st Qu.: -0.58347   1st Qu.: -0.20335   1st Qu.:-0.651372
##  Median :  0.01638   Median :  0.02073   Median :-0.032806
##  Mean   : -0.03306   Mean   : -0.02625   Mean   : 0.002797
##  3rd Qu.:  0.55983   3rd Qu.:  0.31867   3rd Qu.: 0.616891
##  Max.   : 34.30318   Max.   :  9.67394   Max.   : 6.366108
##       V10                 V11                 V12
##  Min.   :-16.60120   Min.   :-4.009307   Min.   :-18.683715
##  1st Qu.: -0.53713   1st Qu.:-0.773726   1st Qu.: -0.409511
##  Median : -0.09378   Median :-0.011436   Median :  0.145403
##  Mean   : -0.02151   Mean   :-0.001127   Mean   : -0.006059
##  3rd Qu.:  0.42024   3rd Qu.: 0.742937   3rd Qu.:  0.624921
##  Max.   : 10.60879   Max.   : 8.879476   Max.   :  4.299511
##       V13                 V14                 V15
##  Min.   :-3.766274   Min.   :-15.29766   Min.   :-3.614591
##  1st Qu.:-0.647246   1st Qu.: -0.42025   1st Qu.:-0.588271
##  Median :-0.013491   Median :  0.05801   Median : 0.034153
##  Mean   :-0.004825   Mean   :  0.01307   Mean   : 0.001668
##  3rd Qu.: 0.656360   3rd Qu.:  0.49705   3rd Qu.: 0.650586
##  Max.   : 3.754715   Max.   :  7.51840   Max.   : 3.722359
##       V16                 V17                 V18                 V19
##  Min.   :-12.186362   Min.   :-21.33819   Min.   :-8.04544   Min.   :-4.03151
##  1st Qu.: -0.456716   1st Qu.: -0.47136   1st Qu.:-0.51057   1st Qu.:-0.43702
##  Median :  0.075296   Median : -0.05606   Median :-0.01016   Median : 0.01202
##  Mean   :  0.000604   Mean   :  0.01274   Mean   :-0.01704   Mean   : 0.01707
##  3rd Qu.:  0.519624   3rd Qu.:  0.42639   3rd Qu.: 0.51332   3rd Qu.: 0.47532
##  Max.   :  4.087802   Max.   :  4.78430   Max.   : 3.10234   Max.   : 3.54200
##       V20                 V21                 V22
##  Min.   :-20.097918   Min.   :-22.757540   Min.   :-4.896349
##  1st Qu.: -0.213020   1st Qu.: -0.230183   1st Qu.:-0.538172
##  Median : -0.060212   Median : -0.033059   Median : 0.001340
##  Mean   : -0.001677   Mean   : -0.009661   Mean   : 0.006827
##  3rd Qu.:  0.130969   3rd Qu.:  0.190598   3rd Qu.: 0.542416
##  Max.   : 10.609220   Max.   : 15.248762   Max.   : 7.357255
##       V23                 V24                 V25
##  Min.   :-22.57500   Min.   :-2.440202   Min.   :-3.640055
```

```
##  1st Qu.: -0.16181   1st Qu.:-0.357104   1st Qu.:-0.319902
##  Median : -0.01380   Median : 0.041960   Median : 0.020575
##  Mean   : -0.01028   Mean   :-0.004085   Mean   : 0.003933
##  3rd Qu.:  0.13947   3rd Qu.: 0.430464   3rd Qu.: 0.345413
##  Max.   : 11.67740   Max.   : 3.448092   Max.   : 2.431118
##       V26                 V27                 V28                Amount
##  Min.   :-1.3511282   Min.   :-5.999996   Min.   :-5.944923   Min.   :   0.00
##  1st Qu.:-0.3209698   1st Qu.:-0.074021   1st Qu.:-0.054068   1st Qu.:   5.00
##  Median :-0.0503856   Median :-0.000343   Median : 0.011933   Median :  21.25
##  Mean   :-0.0009276   Mean   :-0.002867   Mean   : 0.003645   Mean   :  90.31
##  3rd Qu.: 0.2383079   3rd Qu.: 0.088436   3rd Qu.: 0.079137   3rd Qu.:  75.00
##  Max.   : 3.0679074   Max.   : 9.200883   Max.   :15.522649   Max.   :7712.43
##  Class
##  0:5686
##  1:  10
##
##
##
##
```

```
table(test_data$Class)
```

```
##
##    0    1
## 5686   10
```

```
ggplot(data = test_data, aes(x = V1, y = V2, col = Class)) +
  geom_point() +
  ggtitle("Test Data Distribution") +
  theme_bw() +
  scale_color_manual(values = c('blue', 'red'))
```



As of now, it is important to note that the training data contains 22745 cases of non-fraudulent transactions and 39 cases of fraudulent transactions. These values will be important when performing our different data balancing techniques.

The training data will be used to apply the different data balancing techniques, and new data sets will be generated as a result of applying these data balancing methods.

3.1 The Random Oversampling Method (ROS)

Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset., where Examples from the training dataset are selected randomly with replacement. Examples from the minority class can be chosen and added to the new "more balanced" training dataset multiple times; they are selected from the original training dataset, added to the new training dataset, and then returned or "replaced" in the original dataset, allowing them to be selected again. This technique can be effective for those machine learning algorithms that are affected by a skewed distribution and where multiple duplicate examples for a given class can influence the fit of the model. [12]

Do note that the ROS method is only going to be performed on the training data-set, that way we will be able to avoid over-fitting. We will create our dataset using the ROS method using the following script.

```
n_legit <- 22745 #No. of Non-Fraudulent Transactions in our training data-set
new_frac_legit <- 0.50
new_n_total <- n_legit/new_frac_legit

oversampling_result <- ovun.sample(Class ~., data = train_data, method = "over",
                                   N = new_n_total, seed = 2019)
```

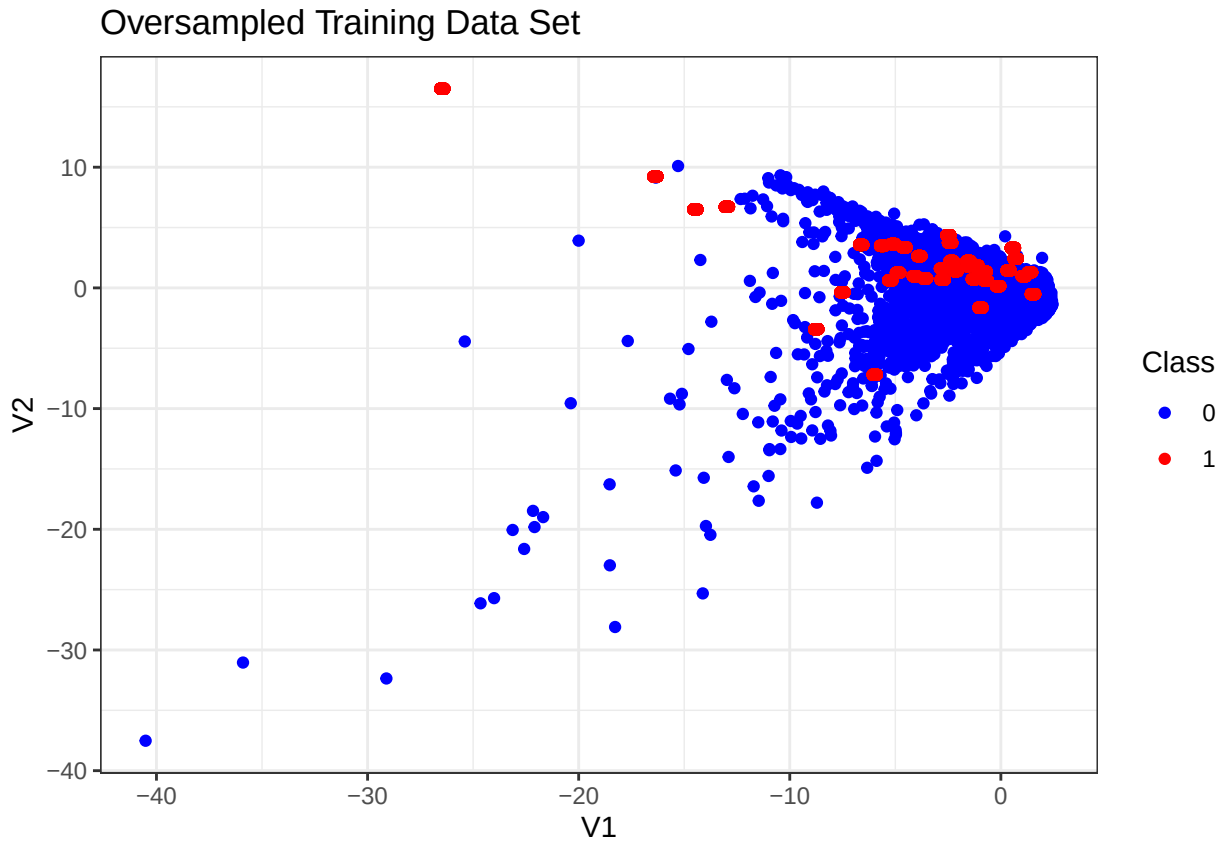
```
oversample_credit <- oversampling_result$data
```

Our resulting dataset called `oversample_credit` is now as follows. We can see that the distribution between the fraudulent cases and non-fraudulent cases are now both at 22745.

```
table(oversample_credit$Class)
```

```
##
##     0     1
## 22745 22745
```

```
ggplot(data = oversample_credit, aes(x = V1, y = V2, col = Class)) +
  geom_point(position = position_jitter(width = 0.1)) +
  ggtitle("Oversampled Training Data Set") +
  theme_bw() +
  scale_color_manual(values = c('blue', 'red'))
```



We can clearly see from the graph that we have augmented the number of fraudulent cases in our dataset. We can further analyze the values that were returned by summarizing our `oversample_credit`

```
summary(oversample_credit)
```

```
##       Time              V1                  V2                  V3
```

```
##  Min.   :     1   Min.   :-40.4701   Min.   :-37.5204   Min.   :-30.1773
##  1st Qu.: 48884   1st Qu.: -2.7560   1st Qu.: -0.2284   1st Qu.: -3.8439
##  Median : 80010   Median : -0.9646   Median :  0.7640   Median : -1.1459
##  Mean   : 90050   Mean   : -2.0034   Mean   :  1.0331   Mean   : -2.5709
##  3rd Qu.:134288   3rd Qu.:  0.9289   3rd Qu.:  1.8087   3rd Qu.:  0.2649
##  Max.   :172769   Max.   :  2.4047   Max.   : 16.4975   Max.   :  3.9350
##        V4                 V5                 V6                 V7
##  Min.   :-5.0712   Min.   :-31.3568   Min.   :-8.9984   Min.   :-31.1973
##  1st Qu.:-0.2239   1st Qu.: -1.3538   1st Qu.:-1.3265   1st Qu.: -2.7925
##  Median : 0.9391   Median : -0.4483   Median :-0.6003   Median : -0.7332
##  Mean   : 1.6720   Mean   : -0.9799   Mean   :-0.5594   Mean   : -2.1379
##  3rd Qu.: 2.8307   3rd Qu.:  0.3940   3rd Qu.: 0.1055   3rd Qu.:  0.2743
##  Max.   :11.8448   Max.   : 15.8488   Max.   :20.3795   Max.   : 30.8977
##        V8                 V9                V10                 V11
##  Min.   :-27.1569   Min.   :-9.4626   Min.   :-22.18709   Min.   :-4.2248
##  1st Qu.: -0.2095   1st Qu.:-1.6784   1st Qu.: -3.53865   1st Qu.:-0.1728
##  Median :  0.1347   Median :-0.5308   Median : -1.02315   Median : 1.1682
##  Mean   : -0.2273   Mean   :-0.9302   Mean   : -2.36810   Mean   : 1.6379
##  3rd Qu.:  0.6067   3rd Qu.: 0.2597   3rd Qu.: -0.05524   3rd Qu.: 3.1021
##  Max.   : 10.7247   Max.   : 9.2724   Max.   : 15.23603   Max.   :10.5453
##       V12                V13                V14                V15
##  Min.   :-15.0227   Min.   :-3.61769   Min.   :-15.06637   Min.   :-4.00556
##  1st Qu.: -4.5403   1st Qu.:-0.75551   1st Qu.: -5.21014   1st Qu.:-0.52426
##  Median : -0.7684   Median : 0.01686   Median : -1.42562   Median : 0.07738
##  Mean   : -2.5379   Mean   : 0.01415   Mean   : -2.90207   Mean   : 0.07246
##  3rd Qu.:  0.2333   3rd Qu.: 0.76587   3rd Qu.:  0.06384   3rd Qu.: 0.63833
##  Max.   :  4.4063   Max.   : 3.85601   Max.   :  6.99175   Max.   : 5.68590
##       V16                V17                V18                V19
##  Min.   :-11.3502   Min.   :-21.7102   Min.   :-8.8595   Min.   :-4.0385
##  1st Qu.: -3.2580   1st Qu.: -5.2119   1st Qu.:-1.8382   1st Qu.:-0.3086
##  Median : -0.7673   Median : -0.6085   Median :-0.4116   Median : 0.2863
##  Mean   : -1.8166   Mean   : -2.7992   Mean   :-0.9354   Mean   : 0.4579
##  3rd Qu.:  0.2355   3rd Qu.:  0.2239   3rd Qu.: 0.3459   3rd Qu.: 1.0239
##  Max.   :  7.0591   Max.   :  7.7666   Max.   : 4.0713   Max.   : 4.4751
##       V20                V21                V22                V23
##  Min.   :-25.22235   Min.   :-14.0316   Min.   :-8.59364   Min.   :-20.75736
##  1st Qu.: -0.22052   1st Qu.: -0.1479   1st Qu.:-0.36505   1st Qu.: -0.23510
##  Median : -0.02463   Median :  0.1352   Median : 0.05766   Median : -0.03298
##  Mean   :  0.06324   Mean   :  0.2660   Mean   : 0.10348   Mean   : -0.01496
##  3rd Qu.:  0.36117   3rd Qu.:  0.6130   3rd Qu.: 0.59857   3rd Qu.:  0.20898
##  Max.   : 15.81505   Max.   : 22.6149   Max.   : 4.35963   Max.   : 16.72282
##       V24                V25                V26                V27
##  Min.   :-2.51321   Min.   :-4.12988   Min.   :-1.726928   Min.   :-7.26348
##  1st Qu.:-0.31545   1st Qu.:-0.36590   1st Qu.:-0.333277   1st Qu.:-0.09410
##  Median : 0.01150   Median : 0.03703   Median :-0.006168   Median : 0.02704
##  Mean   :-0.01521   Mean   :-0.06173   Mean   : 0.043624   Mean   : 0.06057
##  3rd Qu.: 0.34577   3rd Qu.: 0.32422   3rd Qu.: 0.353896   3rd Qu.: 0.39234
##  Max.   : 3.94924   Max.   : 4.80812   Max.   : 3.119295   Max.   : 7.12634
##       V28              Amount         Class
##  Min.   :-8.33812   Min.   :   0.00   0:22745
##  1st Qu.:-0.10667   1st Qu.:   2.12   1:22745
##  Median : 0.02175   Median :  24.64
##  Mean   :-0.02417   Mean   : 105.67
##  3rd Qu.: 0.14475   3rd Qu.: 129.00
```

```
##  Max.   :14.04630   Max.   :7879.42
```

3.2 The Random Undersampling Method (RUS)

Random undersampling involves randomly selecting examples from the majority class to delete from the training dataset. The effect of this is reducing the number of examples in the majority class in the transformed version of the training dataset. This process can be repeated until the desired class distribution is achieved, such as an equal number of examples for each class. A limitation of undersampling is that examples from the majority class are deleted that may be useful, important, or perhaps critical to fitting a robust decision boundary. Given that examples are deleted randomly, there is no way to detect or preserve "good" or more information-rich examples from the majority class. [12]

The following script will be used to perform RUS to our training dataset.

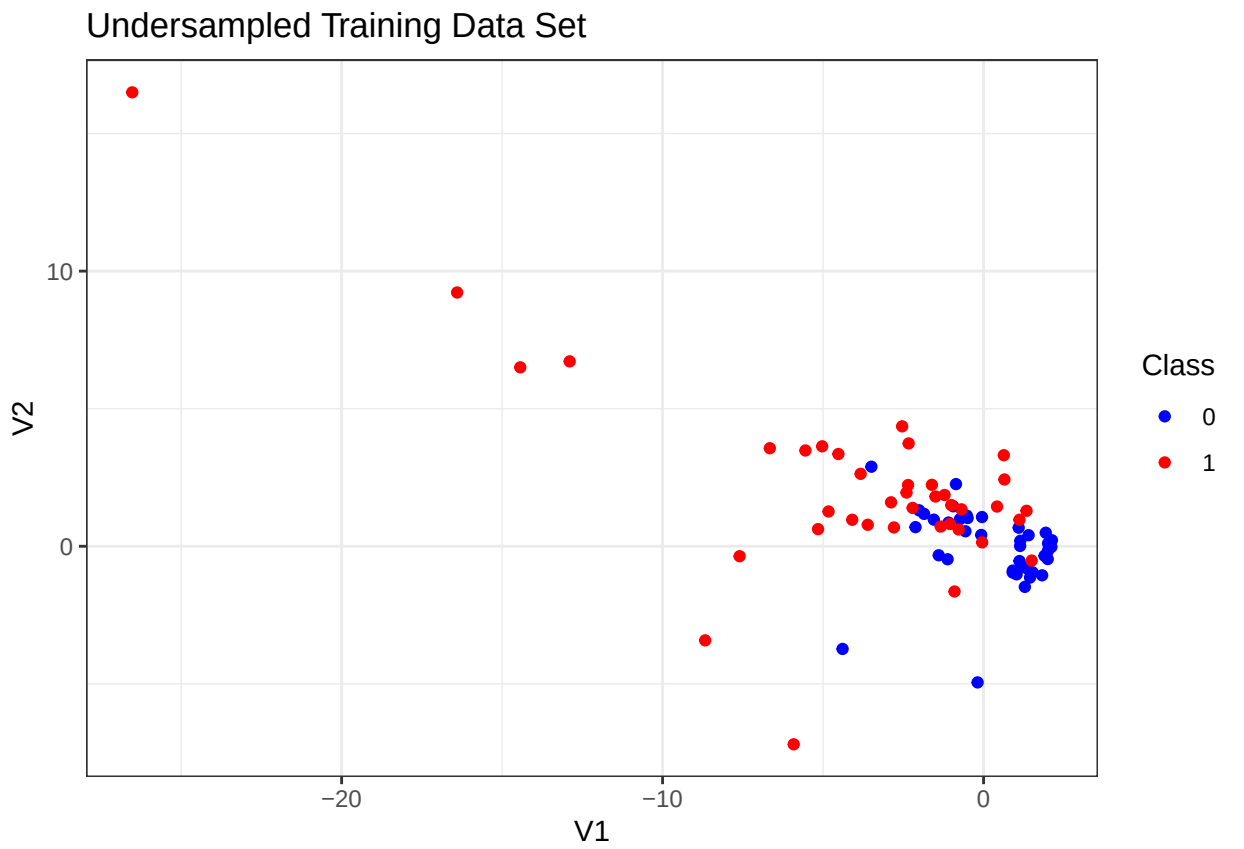
```
n_fraud <- 39 # No. of fraudulent cases in the training dataset
new_frac_fraud <- 0.50
new_n_total <- n_fraud/new_frac_fraud

undersampling_result <- ovun.sample(Class ~ .,
                                    data = train_data,
                                    method = "under",
                                    N = new_n_total,
                                    seed = 2019)

undersampled_credit <- undersampling_result$data
```

Our resulting dataset called `undersampled_credit` is now as follows. We can see that the distribution between the fraudulent cases and non-fraudulent cases are now both at `39`.

```
table(undersampled_credit$Class)
```

```
##
##  0  1
## 39 39
```

```
ggplot(data = undersampled_credit, aes(x = V1, y = V2, col = Class)) +
  geom_point(position = position_jitter(width = 0.1)) +
  ggtitle("Undersampled Training Data Set") +
  theme_bw() +
  scale_color_manual(values = c('blue', 'red'))
```



It is probably dangerous to see how low all our instances are, and how that could affect the classifier, but it probably could lead to less bias when the dataset is used to classify instances. We can now see the distribution of the dataset with the succeeding script.

```
summary(oversample_credit)
```

```
##       Time              V1                 V2                V3
##  Min.   :     1   Min.   :-40.4701   Min.   :-37.5204   Min.   :-30.1773
##  1st Qu.: 48884   1st Qu.: -2.7560   1st Qu.: -0.2284   1st Qu.: -3.8439
##  Median : 80010   Median : -0.9646   Median :  0.7640   Median : -1.1459
##  Mean   : 90050   Mean   : -2.0034   Mean   :  1.0331   Mean   : -2.5709
##  3rd Qu.:134288   3rd Qu.:  0.9289   3rd Qu.:  1.8087   3rd Qu.:  0.2649
##  Max.   :172769   Max.   :  2.4047   Max.   : 16.4975   Max.   :  3.9350
##        V4                V5                 V6                V7
##  Min.   :-5.0712   Min.   :-31.3568   Min.   :-8.9984   Min.   :-31.1973
##  1st Qu.:-0.2239   1st Qu.: -1.3538   1st Qu.:-1.3265   1st Qu.: -2.7925
##  Median : 0.9391   Median : -0.4483   Median :-0.6003   Median : -0.7332
##  Mean   : 1.6720   Mean   : -0.9799   Mean   :-0.5594   Mean   : -2.1379
##  3rd Qu.: 2.8307   3rd Qu.:  0.3940   3rd Qu.: 0.1055   3rd Qu.:  0.2743
##  Max.   :11.8448   Max.   : 15.8488   Max.   :20.3795   Max.   : 30.8977
##        V8                V9                V10               V11
##  Min.   :-27.1569   Min.   :-9.4626   Min.   :-22.18709   Min.   :-4.2248
##  1st Qu.: -0.2095   1st Qu.:-1.6784   1st Qu.: -3.53865   1st Qu.:-0.1728
##  Median :  0.1347   Median :-0.5308   Median : -1.02315   Median : 1.1682
##  Mean   : -0.2273   Mean   :-0.9302   Mean   : -2.36810   Mean   : 1.6379
##  3rd Qu.:  0.6067   3rd Qu.: 0.2597   3rd Qu.: -0.05524   3rd Qu.: 3.1021
```

```
##  Max.   : 10.7247   Max.   : 9.2724   Max.   : 15.23603   Max.   :10.5453
##       V12                V13                V14                V15
##  Min.   :-15.0227   Min.   :-3.61769   Min.   :-15.06637   Min.   :-4.00556
##  1st Qu.: -4.5403   1st Qu.:-0.75551   1st Qu.: -5.21014   1st Qu.:-0.52426
##  Median : -0.7684   Median : 0.01686   Median : -1.42562   Median : 0.07738
##  Mean   : -2.5379   Mean   : 0.01415   Mean   : -2.90207   Mean   : 0.07246
##  3rd Qu.:  0.2333   3rd Qu.: 0.76587   3rd Qu.:  0.06384   3rd Qu.: 0.63833
##  Max.   :  4.4063   Max.   : 3.85601   Max.   :  6.99175   Max.   : 5.68590
##       V16                V17                V18                V19
##  Min.   :-11.3502   Min.   :-21.7102   Min.   :-8.8595   Min.   :-4.0385
##  1st Qu.: -3.2580   1st Qu.: -5.2119   1st Qu.:-1.8382   1st Qu.:-0.3086
##  Median : -0.7673   Median : -0.6085   Median :-0.4116   Median : 0.2863
##  Mean   : -1.8166   Mean   : -2.7992   Mean   :-0.9354   Mean   : 0.4579
##  3rd Qu.:  0.2355   3rd Qu.:  0.2239   3rd Qu.: 0.3459   3rd Qu.: 1.0239
##  Max.   :  7.0591   Max.   :  7.7666   Max.   : 4.0713   Max.   : 4.4751
##       V20                V21                V22                V23
##  Min.   :-25.22235   Min.   :-14.0316   Min.   :-8.59364   Min.   :-20.75736
##  1st Qu.: -0.22052   1st Qu.: -0.1479   1st Qu.:-0.36505   1st Qu.: -0.23510
##  Median : -0.02463   Median :  0.1352   Median : 0.05766   Median : -0.03298
##  Mean   :  0.06324   Mean   :  0.2660   Mean   : 0.10348   Mean   : -0.01496
##  3rd Qu.:  0.36117   3rd Qu.:  0.6130   3rd Qu.: 0.59857   3rd Qu.:  0.20898
##  Max.   : 15.81505   Max.   : 22.6149   Max.   : 4.35963   Max.   : 16.72282
##       V24                V25                V26                V27
##  Min.   :-2.51321   Min.   :-4.12988   Min.   :-1.726928   Min.   :-7.26348
##  1st Qu.:-0.31545   1st Qu.:-0.36590   1st Qu.:-0.333277   1st Qu.:-0.09410
##  Median : 0.01150   Median : 0.03703   Median :-0.006168   Median : 0.02704
##  Mean   :-0.01521   Mean   :-0.06173   Mean   : 0.043624   Mean   : 0.06057
##  3rd Qu.: 0.34577   3rd Qu.: 0.32422   3rd Qu.: 0.353896   3rd Qu.: 0.39234
##  Max.   : 3.94924   Max.   : 4.80812   Max.   : 3.119295   Max.   : 7.12634
##       V28              Amount        Class
##  Min.   :-8.33812   Min.   :   0.00   0:22745
##  1st Qu.:-0.10667   1st Qu.:   2.12   1:22745
##  Median : 0.02175   Median :  24.64
##  Mean   :-0.02417   Mean   : 105.67
##  3rd Qu.: 0.14475   3rd Qu.: 129.00
##  Max.   :14.04630   Max.   :7879.42
```

3.3 ROS and the RUS Method Combined

Another approach involves combining both the ROS and RUS method. Here, we reduce the number of instances of our non-fraudulent transactions through RUS, while increasing the number of fraudulent transactions through ROS. Again, there is danger of important data being lost in this approach, but the bias that the classifier will commit from this dataset may possibly be reduced, although the last statement will be tested during a later section.

The following script will be used to create our new dataset called `sampled_credit` which will be the result of combining the ROS and RUS method to produce a balanced dataset.

```
n_new <- nrow(train_data)
fraction_fraud_new <- 0.50

sampling_result <- ovun.sample(Class ~.,
                               data = train_data,
                               method = "both",
```

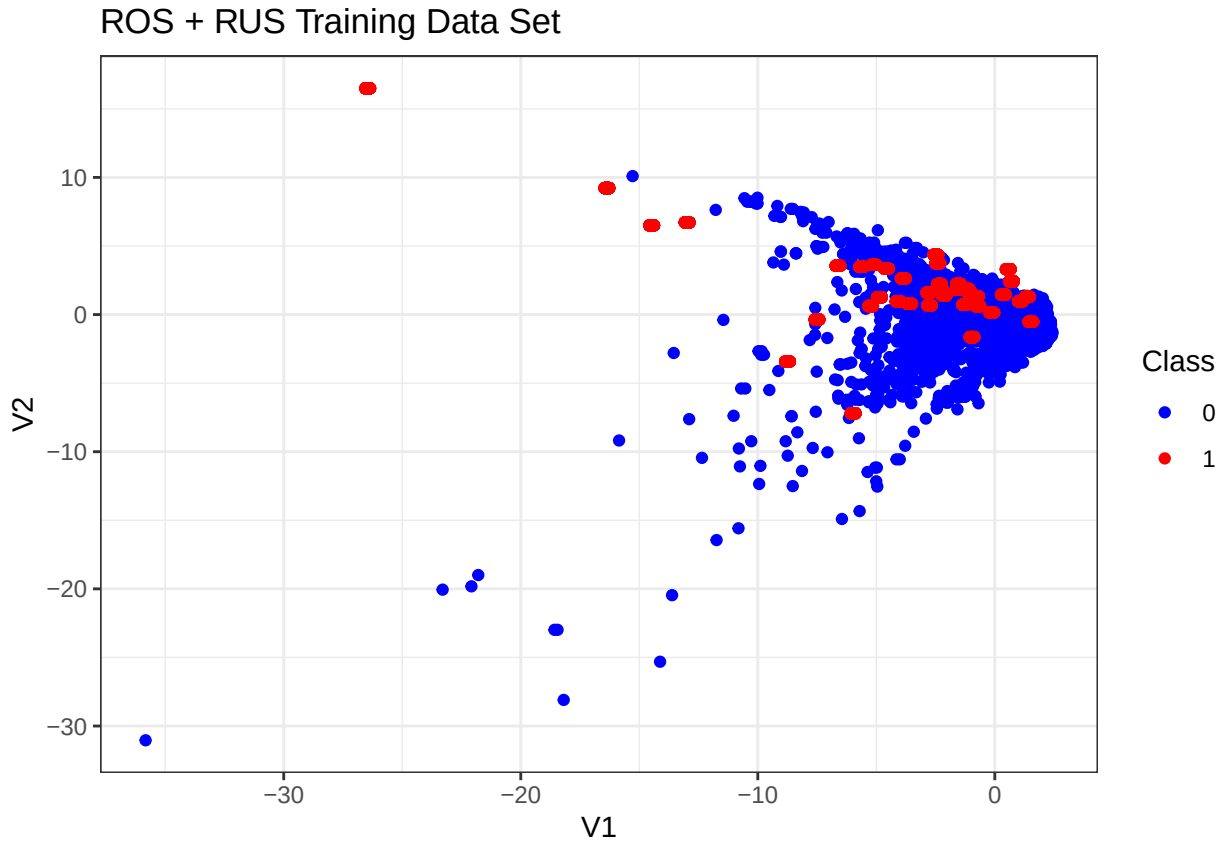
```
                                N = n_new,
                                p = fraction_fraud_new,
                                seed = 2019)

sampled_credit <- sampling_result$data
```

Our resulting dataset, like the previous methods, should now contain only a slight difference between the number of fraudulent cases and non-fraudulent cases.

```
table(sampled_credit$Class)
```

```
##
##     0     1
## 11430 11354
```

```
ggplot(data = sampled_credit, aes(x = V1, y = V2, col = Class)) +
  geom_point(position = position_jitter(width = 0.1)) +
  ggtitle("ROS + RUS Training Data Set") +
  theme_bw() +
  scale_color_manual(values = c('blue', 'red'))
```



We can observe the contents of our dataset with the following script.

```r
summary(sampled_credit)
```

```
##       Time              V1                  V2                  V3
##  Min.   :    77   Min.   :-35.9051   Min.   :-31.0414   Min.   :-30.1773
##  1st Qu.: 48884   1st Qu.: -2.7560   1st Qu.: -0.2240   1st Qu.: -4.4909
##  Median : 79340   Median : -0.9646   Median :  0.7814   Median : -1.1433
##  Mean   : 89259   Mean   : -2.0422   Mean   :  1.0580   Mean   : -2.6180
##  3rd Qu.:132881   3rd Qu.:  0.8884   3rd Qu.:  1.8087   3rd Qu.:  0.2869
##  Max.   :172760   Max.   :  2.4047   Max.   : 16.4975   Max.   :  3.9350
##        V4                 V5                  V6                  V7
##  Min.   :-4.6950   Min.   :-31.3568   Min.   :-8.9984   Min.   :-31.1973
##  1st Qu.:-0.2161   1st Qu.: -1.3959   1st Qu.:-1.3265   1st Qu.: -3.0060
##  Median : 0.9353   Median : -0.4685   Median :-0.5871   Median : -0.7662
##  Mean   : 1.6925   Mean   : -1.0223   Mean   :-0.5624   Mean   : -2.2019
##  3rd Qu.: 3.0242   3rd Qu.:  0.3819   3rd Qu.: 0.1101   3rd Qu.:  0.2476
##  Max.   :11.8448   Max.   : 12.4779   Max.   :20.3795   Max.   : 29.2059
##        V8                 V9                 V10                 V11
##  Min.   :-27.1569   Min.   :-9.4626   Min.   :-22.18709   Min.   :-3.663
##  1st Qu.: -0.2095   1st Qu.:-1.6784   1st Qu.: -3.53865   1st Qu.:-0.150
##  Median :  0.1463   Median :-0.5599   Median : -1.01021   Median : 1.168
##  Mean   : -0.1893   Mean   :-0.9610   Mean   : -2.41699   Mean   : 1.679
##  3rd Qu.:  0.6492   3rd Qu.: 0.2353   3rd Qu.: -0.04169   3rd Qu.: 3.102
##  Max.   : 10.2509   Max.   : 8.9112   Max.   : 13.64446   Max.   :10.545
##       V12                V13                V14                 V15
##  Min.   :-15.0227   Min.   :-3.48015   Min.   :-15.06637   Min.   :-3.70440
##  1st Qu.: -4.5403   1st Qu.:-0.72179   1st Qu.: -5.21014   1st Qu.:-0.53001
##  Median : -0.8033   Median : 0.03881   Median : -1.42562   Median : 0.05091
##  Mean   : -2.6010   Mean   : 0.02402   Mean   : -2.93133   Mean   : 0.06064
##  3rd Qu.:  0.2214   3rd Qu.: 0.79517   3rd Qu.:  0.05602   3rd Qu.: 0.63833
##  Max.   :  3.5092   Max.   : 3.72783   Max.   :  5.45563   Max.   : 5.50112
##       V16                V17                V18                 V19
##  Min.   :-11.3502   Min.   :-21.7102   Min.   :-8.8595   Min.   :-3.9069
##  1st Qu.: -3.2580   1st Qu.: -5.2119   1st Qu.:-1.8382   1st Qu.:-0.3086
##  Median : -0.7847   Median : -0.6099   Median :-0.4139   Median : 0.2974
##  Mean   : -1.8636   Mean   : -2.8831   Mean   :-0.9642   Mean   : 0.4692
##  3rd Qu.:  0.2355   3rd Qu.:  0.2239   3rd Qu.: 0.3517   3rd Qu.: 1.0283
##  Max.   :  7.0591   Max.   :  5.4862   Max.   : 2.9751   Max.   : 4.4751
##       V20                V21                V22                 V23
##  Min.   :-25.22235   Min.   :-14.0316   Min.   :-4.1847   Min.   :-19.33125
##  1st Qu.: -0.22048   1st Qu.: -0.1494   1st Qu.:-0.3650   1st Qu.: -0.23510
##  Median : -0.01851   Median :  0.1424   Median : 0.0578   Median : -0.03298
##  Mean   :  0.06452   Mean   :  0.2773   Mean   : 0.1080   Mean   : -0.02631
##  3rd Qu.:  0.36117   3rd Qu.:  0.6142   3rd Qu.: 0.6076   3rd Qu.:  0.20384
##  Max.   : 15.81505   Max.   : 11.9157   Max.   : 3.6360   Max.   : 11.46671
##       V24                V25                V26                 V27
##  Min.   :-2.417821   Min.   :-2.96761   Min.   :-1.726928   Min.   :-7.26348
##  1st Qu.:-0.338152   1st Qu.:-0.36394   1st Qu.:-0.329869   1st Qu.:-0.09078
##  Median : 0.005053   Median : 0.02980   Median :-0.006168   Median : 0.02707
##  Mean   :-0.014999   Mean   :-0.06531   Mean   : 0.043693   Mean   : 0.06701
##  3rd Qu.: 0.349677   3rd Qu.: 0.32260   3rd Qu.: 0.345217   3rd Qu.: 0.39234
##  Max.   : 3.949245   Max.   : 3.62465   Max.   : 3.004455   Max.   : 6.22814
##       V28                Amount       Class
##  Min.   :-6.10516   Min.   :   0.0   0:11430
```

```
##  1st Qu.:-0.10667   1st Qu.:   2.0   1:11354
##  Median : 0.01981   Median :  24.0
##  Mean   :-0.02829   Mean   : 105.2
##  3rd Qu.: 0.14111   3rd Qu.: 124.2
##  Max.   :14.04630   Max.   :7879.4
```

3.4 Synthetic Minority Oversampling Technique (SMOTE)

The Synthetic Minority Oversampling Technique (SMOTE) is a type of oversampling method. The theory basis is that the feature space of minority class instances is similar to that of the majority class. The mathematical approach is as follows: For each instance x_i in the minority class, SMOTE searches its k nearest neighbors and one neighbor is randomly selected as x_θ (we call instances x_i and $x\theta$ seed sample). Then a random number between [0,1] λ is generated.

The new data set entry x_{new} is then generated as follows:

$$x_{new} = x_i + (x_\theta - x_i) * \lambda$$

Using the SMOTE technique, we will generate a new dataset using the following script.

```
# Set the number of fraud and legitimate cases and the desired
#percentage of legimate cases

n0 <- 22745 #No. of Non-fraudulent cases
n1 <- 39 #No. of fraudulent cases
r0 <- 0.6 #Proportion to convert minority cases

#Calculate the value for the dup_size parameter of SMOTE
ntimes <- ((1 - r0) / r0) * (n0 / n1) - 1

smote_output = SMOTE(X = train_data[, -c(1,31)],
                     target = train_data$Class,
                     K = 5,
                     dup_size = ntimes)

credit_smote <- smote_output$data

colnames(credit_smote)[30] <- "Class" #Fix corresponding columnd name
```

Likewise, we can find that the dataset is has been balanced in accordance to the algorithm. The visualization can give a clearer idea on how the minority class (Fraudulent cases) has been re-balanced alongside with the majority class (Non-fraudulent cases).

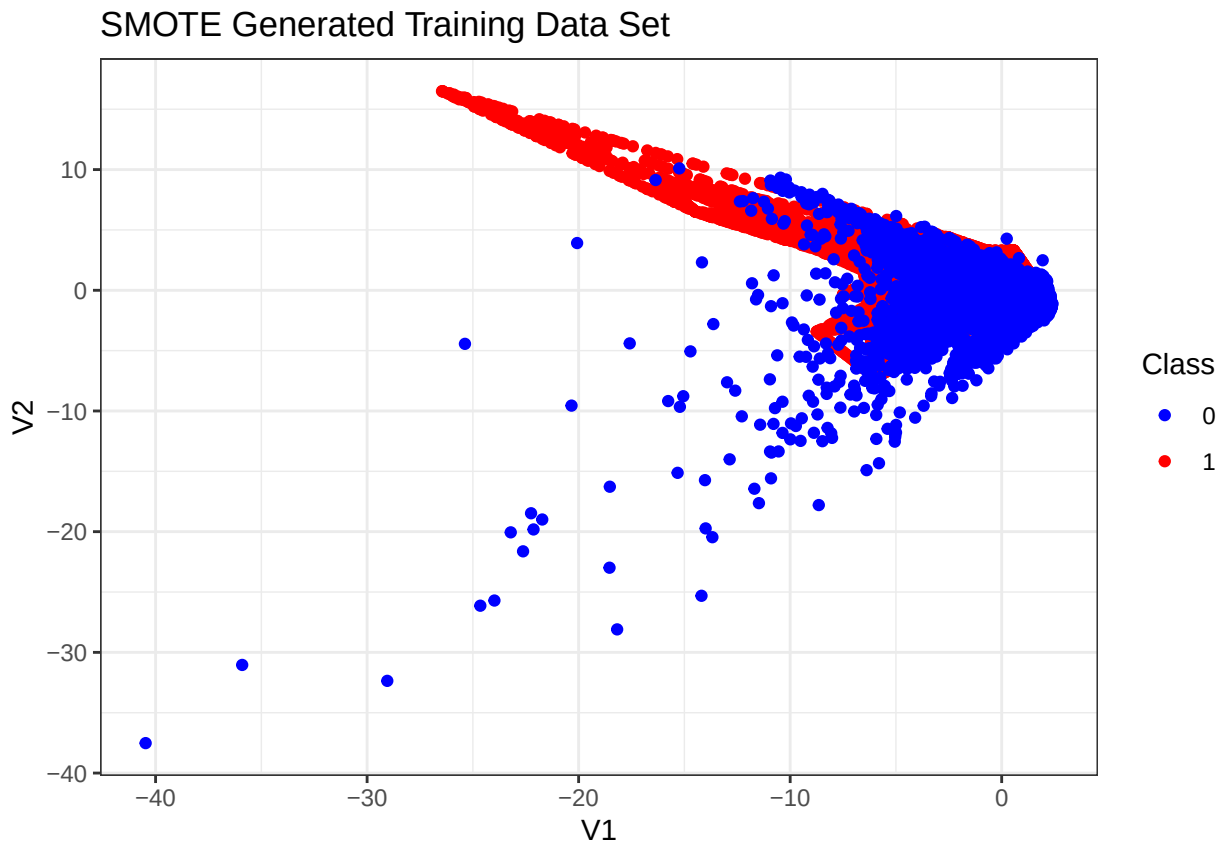
```
table(credit_smote$Class)
```

```
##
##     0     1
## 22745 15132
```

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

```
##
##         0         1
## 0.6004963 0.3995037
```

```
ggplot(credit_smote, aes(x = V1, y = V2, color = Class)) +
  geom_point() +
  ggtitle("SMOTE Generated Training Data Set") +
  theme_bw() +
  scale_color_manual(values = c('blue','red'))
```



Finally, the summary of the generated dataset are as follows.

```
summary(credit_smote)
```

```
##        V1                  V2                  V3                  V4
##  Min.   :-40.4701   Min.   :-37.5204   Min.   :-30.1773   Min.   :-5.0712
##  1st Qu.: -2.3937   1st Qu.: -0.2551   1st Qu.: -3.2894   1st Qu.:-0.3602
##  Median : -0.7597   Median :  0.6917   Median : -1.0300   Median : 0.7445
##  Mean   : -1.3617   Mean   :  0.7630   Mean   : -1.8970   Mean   : 1.2637
##  3rd Qu.:  1.0432   3rd Qu.:  1.5262   3rd Qu.:  0.4706   3rd Qu.: 2.4977
##  Max.   :  2.4047   Max.   : 16.4975   Max.   :  3.9350   Max.   :11.8448
##        V5                  V6                  V7                  V8
##  Min.   :-31.3568   Min.   :-8.9984   Min.   :-31.1973   Min.   :-27.15691
##  1st Qu.: -1.2391   1st Qu.:-1.2701   1st Qu.: -2.3032   1st Qu.: -0.19616
##  Median : -0.4113   Median :-0.5968   Median : -0.5570   Median :  0.11702
```

```
##  Mean   : -0.6728   Mean   :-0.4981   Mean   : -1.5364   Mean   : -0.02386
##  3rd Qu.:  0.3418   3rd Qu.: 0.1144   3rd Qu.:  0.2715   3rd Qu.:  0.60281
##  Max.   : 15.8488   Max.   :20.3795   Max.   : 30.8977   Max.   : 10.72473
##        V9                V10                 V11                V12
##  Min.   :-9.4626   Min.   :-22.18709   Min.   :-4.2248   Min.   :-15.0227
##  1st Qu.:-1.4567   1st Qu.: -3.24262   1st Qu.:-0.2782   1st Qu.: -3.7179
##  Median :-0.4417   Median : -0.69433   Median : 0.8488   Median : -0.5765
##  Mean   :-0.7022   Mean   : -1.83467   Mean   : 1.3216   Mean   : -2.0556
##  3rd Qu.: 0.2716   3rd Qu.:  0.04556   3rd Qu.: 2.6299   3rd Qu.:  0.3061
##  Max.   : 9.2724   Max.   : 15.23603   Max.   :10.5453   Max.   :  4.4063
##       V13                V14                 V15                V16
##  Min.   :-3.61769   Min.   :-15.0664   Min.   :-4.00556   Min.   :-11.3502
##  1st Qu.:-0.62113   1st Qu.: -4.7917   1st Qu.:-0.46670   1st Qu.: -2.6929
##  Median : 0.03007   Median : -0.6620   Median : 0.09599   Median : -0.5143
##  Mean   : 0.02175   Mean   : -2.3569   Mean   : 0.06361   Mean   : -1.4986
##  3rd Qu.: 0.67142   3rd Qu.:  0.2022   3rd Qu.: 0.63389   3rd Qu.:  0.2747
##  Max.   : 3.85601   Max.   :  6.9918   Max.   : 5.68590   Max.   :  7.0591
##       V17                V18                V19                V20
##  Min.   :-21.7102   Min.   :-8.8595   Min.   :-4.0385   Min.   :-25.22235
##  1st Qu.: -4.1244   1st Qu.:-1.3770   1st Qu.:-0.2576   1st Qu.: -0.20491
##  Median : -0.4918   Median :-0.3426   Median : 0.2595   Median : -0.01371
##  Mean   : -2.2786   Mean   :-0.7523   Mean   : 0.4032   Mean   :  0.04939
##  3rd Qu.:  0.1829   3rd Qu.: 0.3027   3rd Qu.: 0.9649   3rd Qu.:  0.27040
##  Max.   :  7.7666   Max.   : 4.0713   Max.   : 4.4751   Max.   : 15.81505
##       V21                V22                V23                 V24
##  Min.   :-14.0316   Min.   :-8.59364   Min.   :-20.75736   Min.   :-2.513206
##  1st Qu.: -0.1496   1st Qu.:-0.36516   1st Qu.: -0.19736   1st Qu.:-0.276494
##  Median :  0.1430   Median : 0.04211   Median : -0.02619   Median : 0.017555
##  Mean   :  0.2995   Mean   : 0.05771   Mean   : -0.01347   Mean   :-0.007691
##  3rd Qu.:  0.5165   3rd Qu.: 0.52112   3rd Qu.:  0.18344   3rd Qu.: 0.334399
##  Max.   : 22.6149   Max.   : 4.35963   Max.   : 16.72282   Max.   : 3.949245
##       V25                 V26                 V27                 V28
##  Min.   :-4.129885   Min.   :-1.726928   Min.   :-7.26348   Min.   :-8.338123
##  1st Qu.:-0.335058   1st Qu.:-0.306115   1st Qu.:-0.07976   1st Qu.:-0.066241
##  Median :-0.007437   Median : 0.006815   Median : 0.02773   Median : 0.020126
##  Mean   :-0.043923   Mean   : 0.036714   Mean   : 0.07481   Mean   : 0.008729
##  3rd Qu.: 0.301964   3rd Qu.: 0.314991   3rd Qu.: 0.29141   3rd Qu.: 0.143440
##  Max.   : 4.808122   Max.   : 3.119295   Max.   : 7.12634   Max.   :14.046301
##      Amount             Class
##  Min.   :   0.000   Length:37877
##  1st Qu.:   2.783   Class :character
##  Median :  22.720   Mode  :character
##  Mean   :  96.094
##  3rd Qu.: 107.631
##  Max.   :7879.420
```

4.0 Generating Confusion Matrices via Classification from Sub Datasets

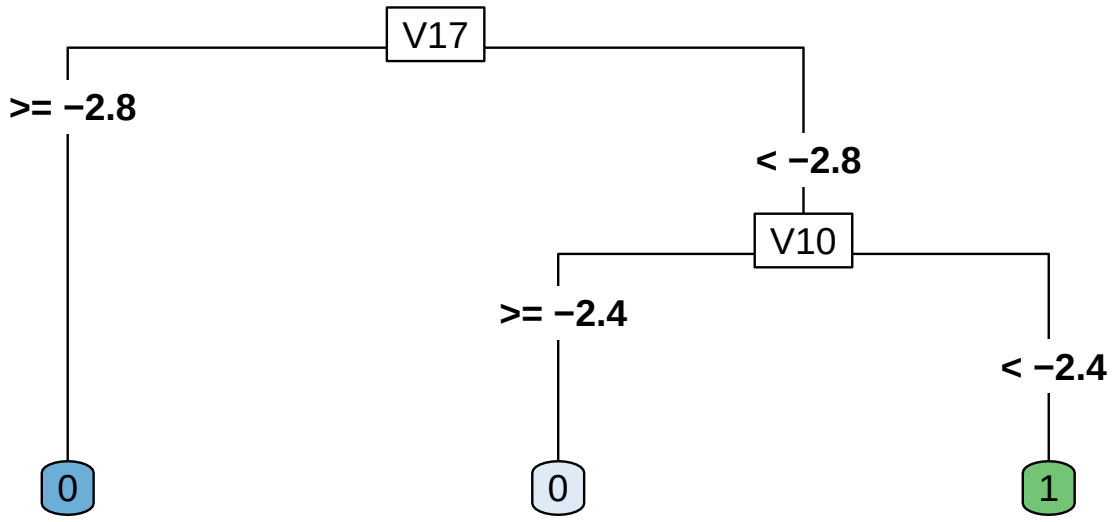
At this point, we have a series of datasets available. To give a summary, the following are a description of the datasets that are now provided within the codes that will be executed in this section.

`credit_card` = 100% of the dataset

`final_credit_sample` = 90% of the dataset (Used for final analysis)
`sampled_credit` = 10% of the dataset used for training
`train_data` = 80% of sampled_credit
`test_data` = 20% of sampled_credit
`oversample_credit` = train_data balanced with ROS
`undersampled_credit` = train_data balanced with RUS
`sampled_credit` = train_data balanced with ROS and RUS
`credit_smote` = train_data balanced with SMOTE

4.1 Decision tree without data balancing

We will first generate the decision tree without any sort of data balancing done on the training data that was sampled. The folowing is the plot that is generated from the dataset.



The resulting confusion matrix when applied to the test set is as follows:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5684    1
##          1    2    9
##
##                Accuracy : 0.9995
##                  95% CI : (0.9985, 0.9999)
##     No Information Rate : 0.9982
```

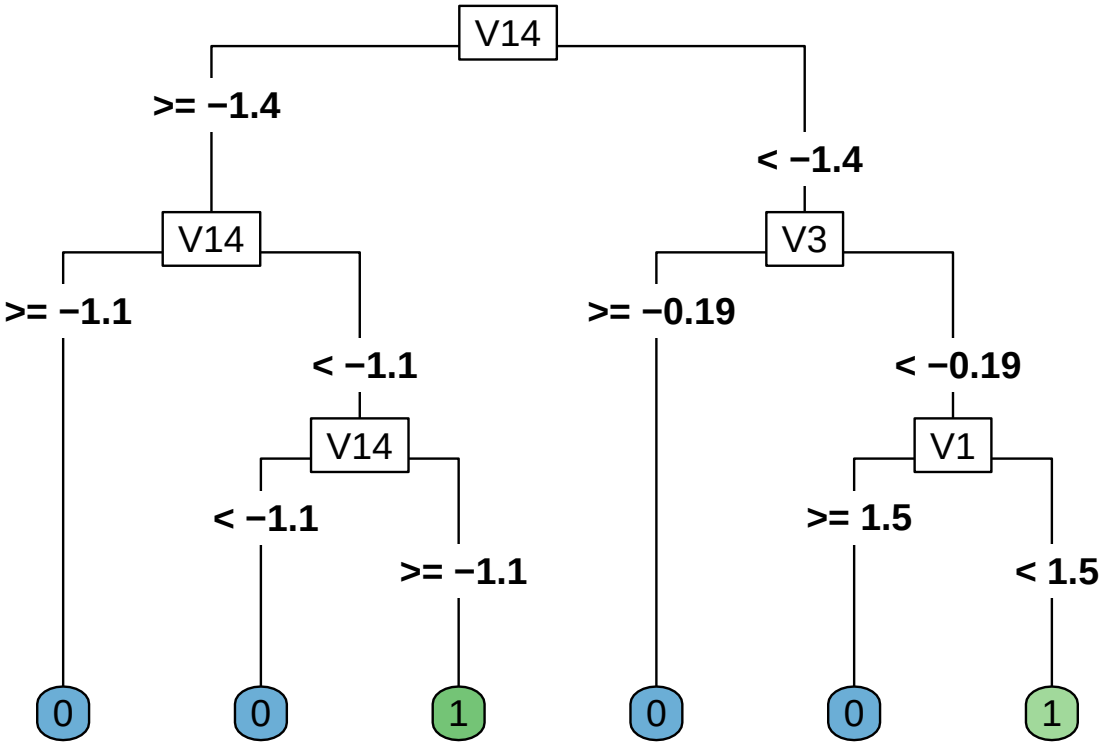
```
##     P-Value [Acc > NIR] : 0.01029
##
##                   Kappa : 0.8569
##
##  Mcnemar's Test P-Value : 1.00000
##
##             Sensitivity : 0.9996
##             Specificity : 0.9000
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.8182
##              Prevalence : 0.9982
##          Detection Rate : 0.9979
##    Detection Prevalence : 0.9981
##       Balanced Accuracy : 0.9498
##
##        'Positive' Class : 0
##
```

Applying it to the rest of the dataset gives the following confusion matrix.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 255826    165
##          1     58    278
##
##                Accuracy : 0.9991
##                  95% CI : (0.999, 0.9992)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7133
##
##  Mcnemar's Test P-Value : 1.263e-12
##
##             Sensitivity : 0.9998
##             Specificity : 0.6275
##          Pos Pred Value : 0.9994
##          Neg Pred Value : 0.8274
##              Prevalence : 0.9983
##          Detection Rate : 0.9980
##    Detection Prevalence : 0.9987
##       Balanced Accuracy : 0.8137
##
##        'Positive' Class : 0
##
```

4.2 Decision tree with ROS balancing

Next, the dataset that was balanced with ROS will be used to generate our classifer.



The resulting confusion matrix when applied to the test set is as follows.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5582    1
##          1  104    9
##
##                Accuracy : 0.9816
##                  95% CI : (0.9777, 0.9849)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1436
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98171
##             Specificity : 0.90000
##          Pos Pred Value : 0.99982
##          Neg Pred Value : 0.07965
##              Prevalence : 0.99824
##          Detection Rate : 0.97999
##    Detection Prevalence : 0.98016
##       Balanced Accuracy : 0.94085
```

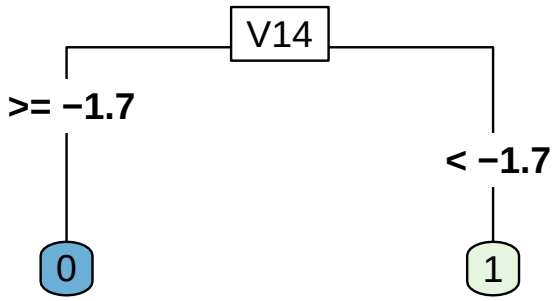
```
##
##        'Positive' Class : 0
##
```

The corresponding model when applied to the rest of the dataset is as follows.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 251006     72
##          1   4878    371
##
##                Accuracy : 0.9807
##                  95% CI : (0.9801, 0.9812)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1276
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98094
##             Specificity : 0.83747
##          Pos Pred Value : 0.99971
##          Neg Pred Value : 0.07068
##              Prevalence : 0.99827
##          Detection Rate : 0.97924
##    Detection Prevalence : 0.97952
##       Balanced Accuracy : 0.90920
##
##        'Positive' Class : 0
##
```

4.3 Decision tree with RUS balancing

The next dataset to be used for classifying will be the dataset balanced with RUS.



This led to the following confusion matrix:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5504    1
##          1  182    9
##
##                Accuracy : 0.9679
##                  95% CI : (0.963, 0.9723)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0865
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.96799
##             Specificity : 0.90000
##          Pos Pred Value : 0.99982
##          Neg Pred Value : 0.04712
##              Prevalence : 0.99824
##          Detection Rate : 0.96629
##    Detection Prevalence : 0.96647
##       Balanced Accuracy : 0.93400
```

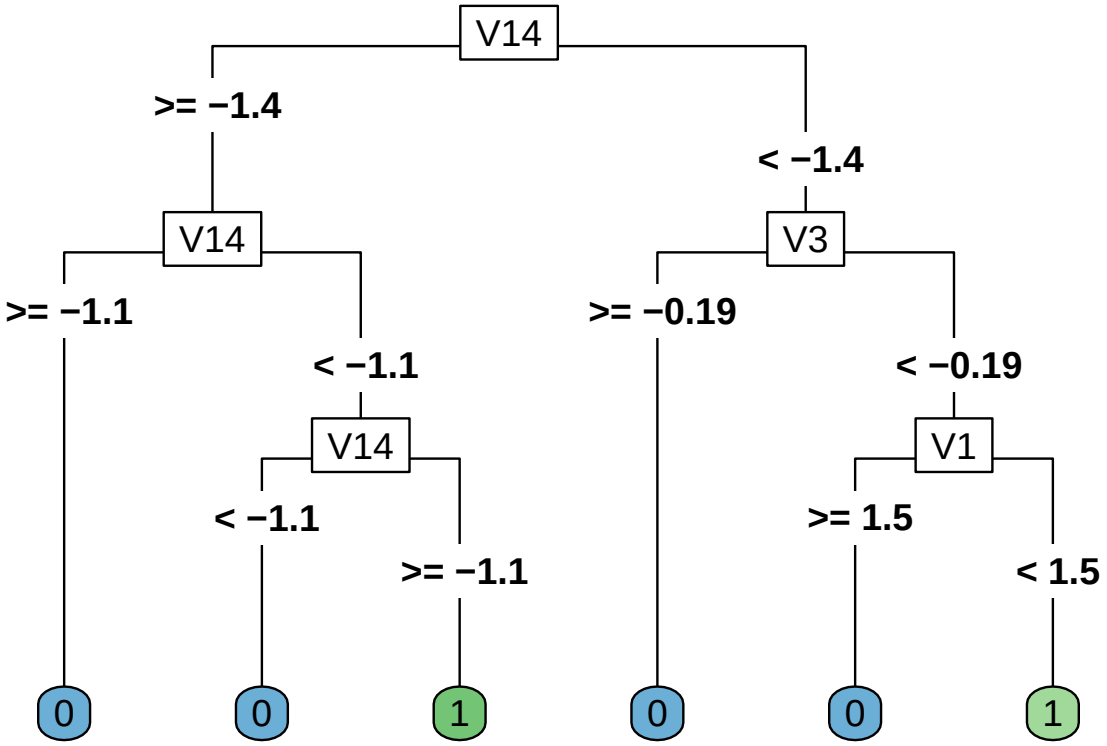
```
##
##        'Positive' Class : 0
##
```

Applying the decision tree to the rest of the dataset yields the following confusion matrix.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 247719     60
##          1   8165    383
##
##                Accuracy : 0.9679
##                  95% CI : (0.9672, 0.9686)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0822
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.96809
##             Specificity : 0.86456
##          Pos Pred Value : 0.99976
##          Neg Pred Value : 0.04481
##              Prevalence : 0.99827
##          Detection Rate : 0.96642
##    Detection Prevalence : 0.96665
##       Balanced Accuracy : 0.91633
##
##        'Positive' Class : 0
##
```

4.4 Decision tree with ROS + RUS balancing

The decision tree generated with the ROS+RUS dataset is then given in the following figure.



Likewise, this is the resulting confusion matrix.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5582    1
##          1  104    9
##
##                Accuracy : 0.9816
##                  95% CI : (0.9777, 0.9849)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1436
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98171
##             Specificity : 0.90000
##          Pos Pred Value : 0.99982
##          Neg Pred Value : 0.07965
##              Prevalence : 0.99824
##          Detection Rate : 0.97999
##    Detection Prevalence : 0.98016
##       Balanced Accuracy : 0.94085
```

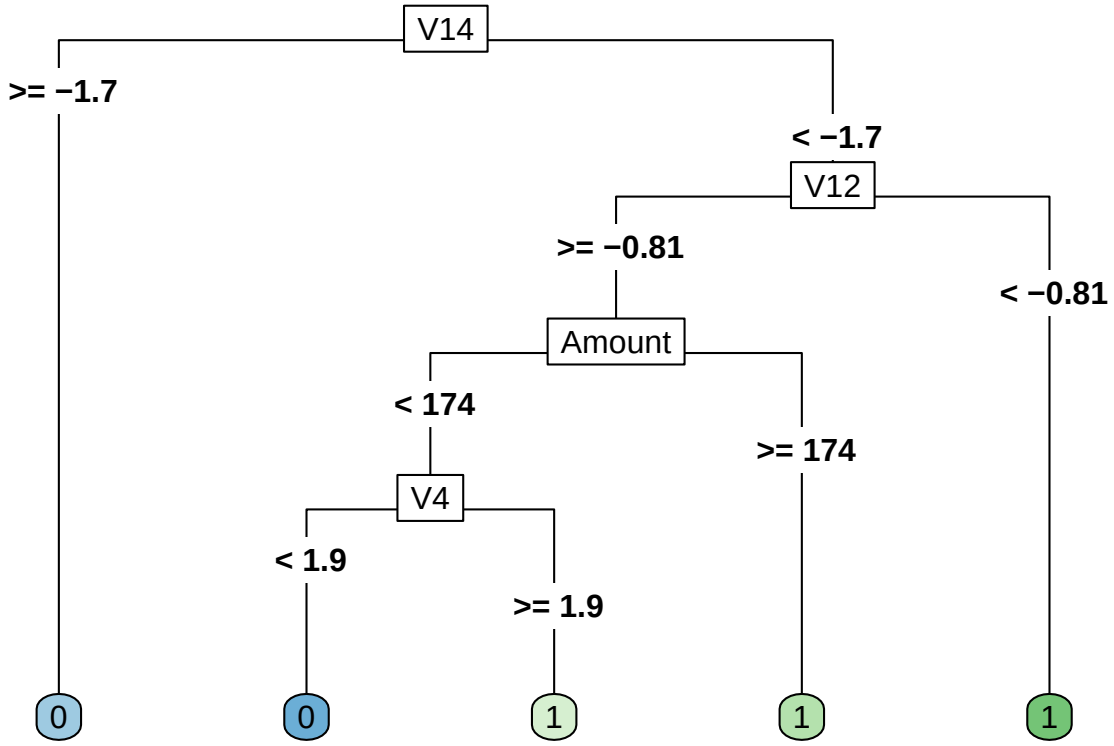
```
##
##        'Positive' Class : 0
##
```

Applying the model to the dataset yields the following result.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 250972     72
##          1   4912    371
##
##                Accuracy : 0.9806
##                  95% CI : (0.98, 0.9811)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1268
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98080
##             Specificity : 0.83747
##          Pos Pred Value : 0.99971
##          Neg Pred Value : 0.07023
##              Prevalence : 0.99827
##          Detection Rate : 0.97911
##    Detection Prevalence : 0.97939
##       Balanced Accuracy : 0.90914
##
##        'Positive' Class : 0
##
```

4.5 Decision tree wih SMOTE balancing

For the dataset balanced using the SMOTE, the following illustration shows the decision tree generated.



The corresponding confusion matrix is as follows.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5626    1
##          1   60    9
##
##                Accuracy : 0.9893
##                  95% CI : (0.9863, 0.9918)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2255
##
##  Mcnemar's Test P-Value : 1.118e-13
##
##             Sensitivity : 0.9894
##             Specificity : 0.9000
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.1304
##              Prevalence : 0.9982
##          Detection Rate : 0.9877
##    Detection Prevalence : 0.9879
##       Balanced Accuracy : 0.9447
```

```
##
##        'Positive' Class : 0
##
```

On the rest of the dataset, the following confusion matrix is obtained.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 253376     63
##          1   2508    380
##
##                Accuracy : 0.99
##                  95% CI : (0.9896, 0.9904)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2258
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9902
##             Specificity : 0.8578
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.1316
##              Prevalence : 0.9983
##          Detection Rate : 0.9885
##    Detection Prevalence : 0.9887
##       Balanced Accuracy : 0.9240
##
##        'Positive' Class : 0
##
```

5.0 Data Balancing Techniques on the full dataset.

Sections 3 and 4, our methodology described taking a small subset of the dataset, generating a training and test set from the subset, and perfoming data balancing techniques on the training set. We observed that there is a plausibility in improving the effect of detecting True Negatives, but the kappa value is considerably very low.

In this section, data balancing techniques will be performed on the full dataset before performing the typical 80-20 split for training and testing.

5.1 Baseline Dataset Classification

While it is true that performing classification on the base dataset alone might not yield a reliable model and may contain a very large bias towards the non-fraudulent cases, we will still generate it to use it as another baseline model along with the model generated with ZeroR in Section 2.0.

Our obtained training and testing dataset will consist of the following distribution.

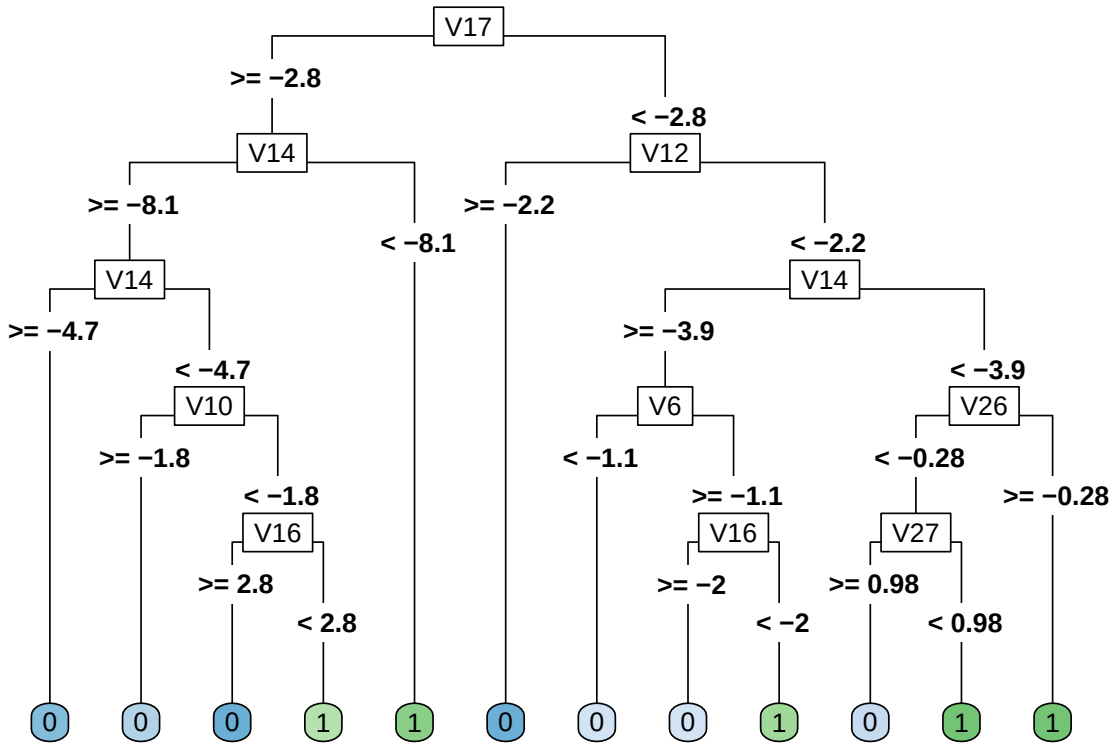
```
table(training_credit_vanilla$Class)
```

```
##
##      0      1
## 227452    394
```

```
table(testing_credit_vanilla$Class)
```

```
##
##     0     1
## 56863    98
```

The resulting classification model and confusion matrix are as follows.




```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56858    26
##          1     5    72
##
##                Accuracy : 0.9995
##                  95% CI : (0.9992, 0.9996)
##     No Information Rate : 0.9983
```

```
##     P-Value [Acc > NIR] : 2.496e-15
##
##                   Kappa : 0.8226
##
##  Mcnemar's Test P-Value : 0.000328
##
##             Sensitivity : 0.9999
##             Specificity : 0.7347
##          Pos Pred Value : 0.9995
##          Neg Pred Value : 0.9351
##              Prevalence : 0.9983
##          Detection Rate : 0.9982
##    Detection Prevalence : 0.9986
##       Balanced Accuracy : 0.8673
##
##        'Positive' Class : 0
##
```

5.2 Balanced Complete Dataset with ROS

Now, ROS will be applied into the dataset before obtaining the training and testing data.

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

Applying ROS before obtaining the training and testing data set will yield to the following distribution for the training and testing data respectively.

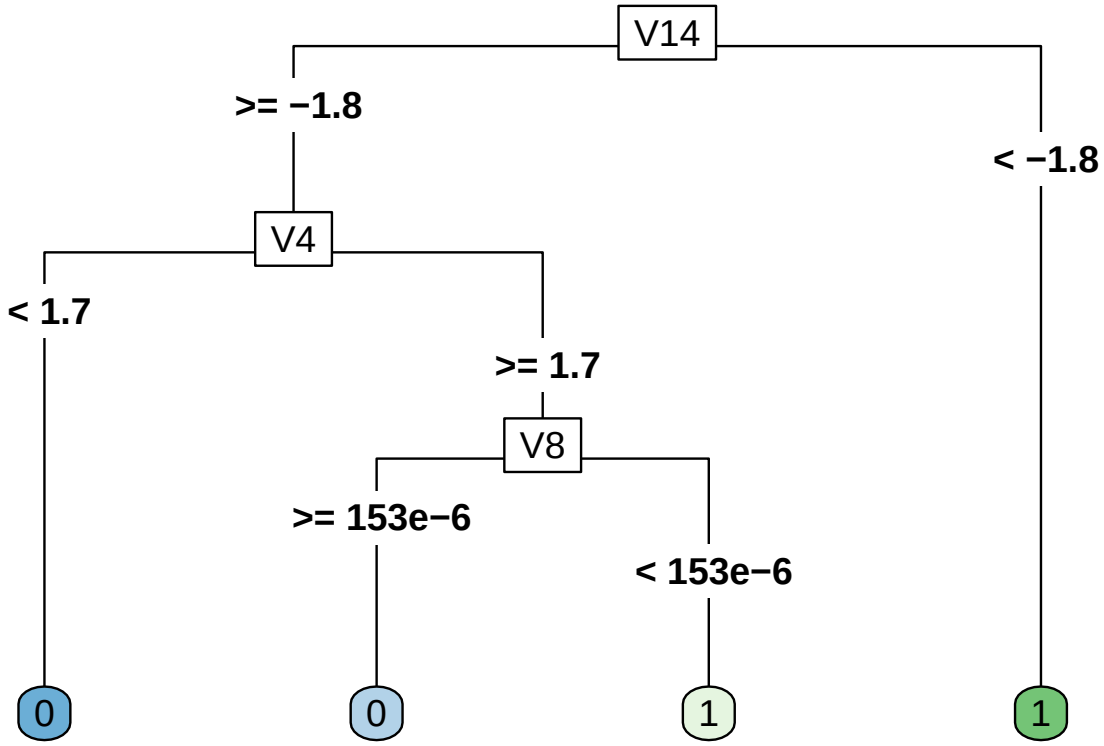
```
table(training_credit_ros$Class)
```

```
##
##      0      1
## 227452 227452
```

```
table(testing_credit_ros$Class)
```

```
##
##     0     1
## 56863 56863
```

The corresponding classifier model and confusion matrix obtained upon testing are as follows.




```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 53671  4247
##          1  3192 52616
##
##                Accuracy : 0.9346
##                  95% CI : (0.9331, 0.936)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8692
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9439
##             Specificity : 0.9253
##          Pos Pred Value : 0.9267
##          Neg Pred Value : 0.9428
##              Prevalence : 0.5000
##          Detection Rate : 0.4719
##    Detection Prevalence : 0.5093
##       Balanced Accuracy : 0.9346
##
##        'Positive' Class : 0
```

```
##
```

5.3 Balanced Complete Dataset with RUS

The RUS method will now be applied on the dataset. Again, with the vast amount of data we have, it might be possible that important data might be lost, but there's a possibility that significant ones can still remain.

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

Applying RUS to the complete dataset gives us the following distribution for our training and test data respectively.

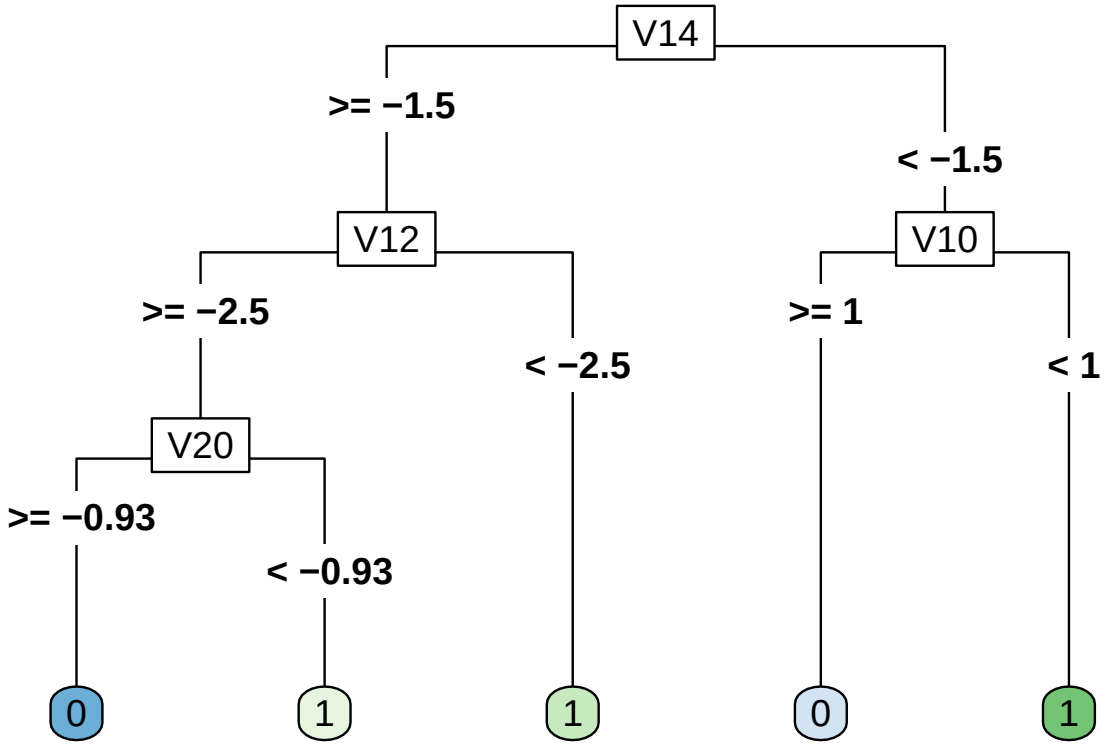
```
table(training_credit_rus$Class)
```

```
##
##   0   1
## 394 394
```

```
table(testing_credit_rus$Class)
```

```
##
##  0  1
## 98 98
```

The resulting computational model and confusion matrix are as follows.




```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 88 11
##          1 10 87
##
##                Accuracy : 0.8929
##                  95% CI : (0.8409, 0.9324)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7857
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8980
##             Specificity : 0.8878
##          Pos Pred Value : 0.8889
##          Neg Pred Value : 0.8969
##              Prevalence : 0.5000
##          Detection Rate : 0.4490
##    Detection Prevalence : 0.5051
##       Balanced Accuracy : 0.8929
##
##        'Positive' Class : 0
```

##

5.4 Balanced Complete Dataset with ROS + RUS

We will now observe the impact of combining ROS and RUS with the dataset before deriving the training and test data respectively.

```
##
##      0      1
## 142430 142377
```

Doing so, first of all, will provide us with the distributions in our training and test data set.

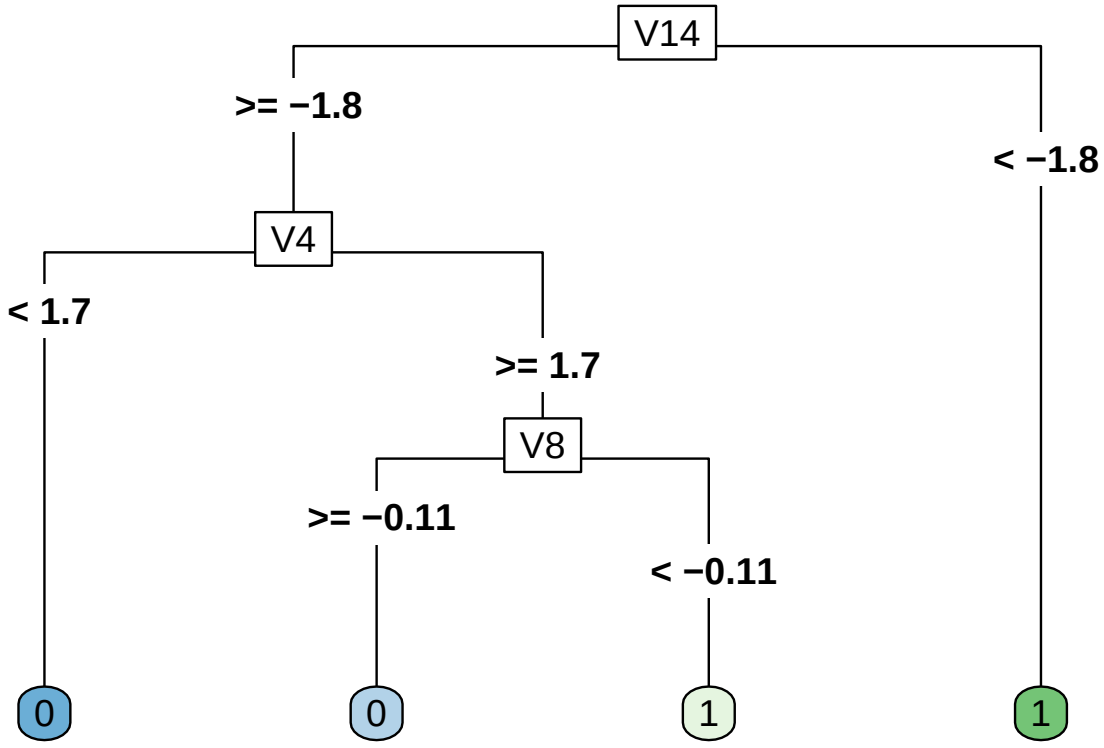
```
table(training_credit_both$Class)
```

```
##
##      0      1
## 113944 113902
```

```
table(testing_credit_both$Class)
```

```
##
##     0     1
## 28486 28475
```

This leads to the following classification model and resulting confusion matrix.



```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 27133  2226
##          1  1353 26249
##
##                Accuracy : 0.9372
##                  95% CI : (0.9351, 0.9391)
##     No Information Rate : 0.5001
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8743
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9525
##             Specificity : 0.9218
##          Pos Pred Value : 0.9242
##          Neg Pred Value : 0.9510
##              Prevalence : 0.5001
##          Detection Rate : 0.4763
##    Detection Prevalence : 0.5154
##       Balanced Accuracy : 0.9372
##
##        'Positive' Class : 0
##
```

5.5 Balanced Complete Dataset with SMOTE

We will see if SMOTE truly is the most effective in enabling us to detect the True Negatives if we use it to balance our dataset.

```
##
##      0      1
## 284315 189420
```

Applying SMOTE to our entire dataset allows us to obtain the following distribution for the obtained training and test data respectively.

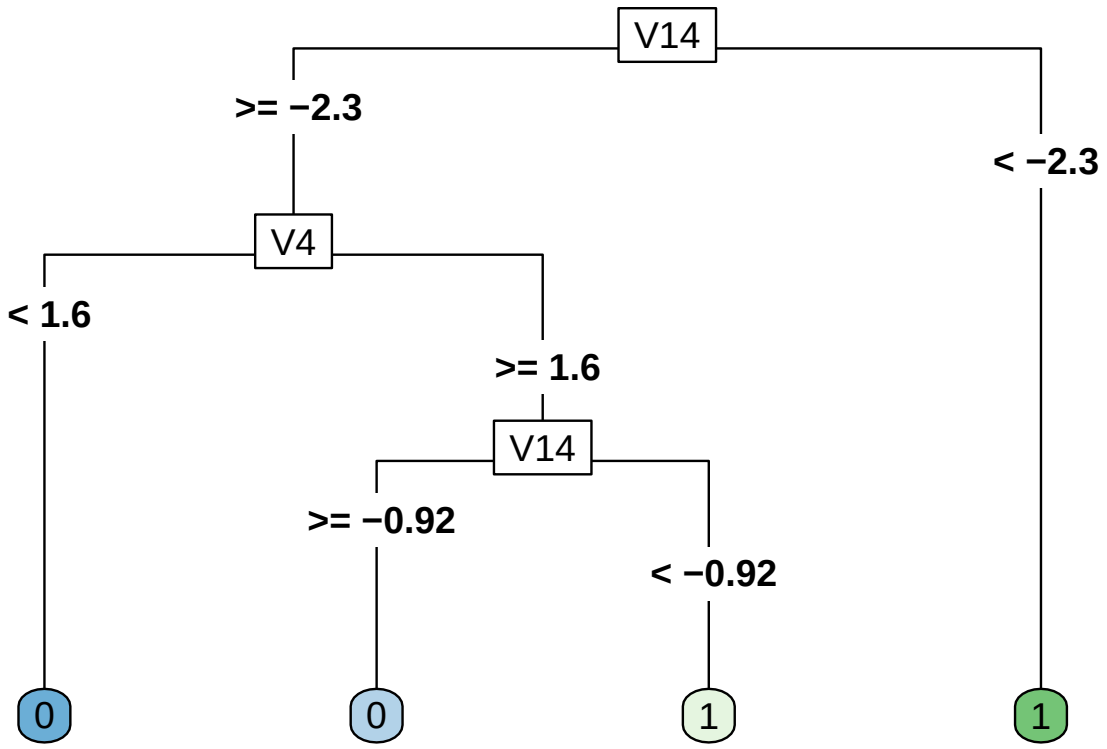
```
table(training_credit_smote$Class)
```

```
##
##      0      1
## 227452 151536
```

```
table(testing_credit_smote$Class)
```

```
##
##     0     1
## 56863 37884
```

The classification model and resulting confusion matrix now looks like the followig.




```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 55500  3462
##          1  1363 34422
##
##                Accuracy : 0.9491
##                  95% CI : (0.9477, 0.9505)
##     No Information Rate : 0.6002
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8929
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9760
##             Specificity : 0.9086
##          Pos Pred Value : 0.9413
##          Neg Pred Value : 0.9619
##              Prevalence : 0.6002
##          Detection Rate : 0.5858
##    Detection Prevalence : 0.6223
##       Balanced Accuracy : 0.9423
```

```
##
##        'Positive' Class : 0
##
```

6.0 Results and Analysis

The aim of this project is to determine the effect of data balancing techniques with regards to properly detecting the True Negative cases (Fraudulent Transactions) while minimizing negative effects on the True Positive Rates (Non-Fraudulent Transactions).

Since, for imbalanced datasets, accuracy is not a valid form of evaluation, and true to the objective of the project, we can compare the following measures in identifying the performance for each classification model resulting from obtaining data balancing techniques. These measures are:

`TNR` = The True Negative Rate, or the rate of which fraudulent cases are identified out of the total number of actual fraudulent cases. In the confusion matrices, this is referred to as the `Specificity`.

`TPR` = The True Positive Rate, or the rate of which non-fraudulent cases are identified out of the total number of actual non-fraudulent cases. In the confusion matrices, this is referred to as the `Sensitivity`, or otherwise known as the `Recall`.

`Kappa` = The Kappa statistic or Cohen's Kappa is used to measure inter-rater reliability for categorical items.[14,15] The higher the kappa, the better. The aim is to at least achieve a Kappa of 0.8.[16]

The following table shows the results from applying data balancing techniques to only a subset of the dataset.

Model	TNR	TPR	Kappa
Sub DS No Balancing	0.6275	0.9999	0.7133
Sub DS ROS	0.8375	0.9809	0.1276
Sub DS RUS	0.8646	0.9681	0.0820
Sub DS ROS+RUS	0.8375	0.9808	0.1268
Sub DS SMOTE	0.8578	0.9902	0.2258

Without any data balancing, TNR was only 0.628, with a Kappa of 0.713. There was an improvement in the TNR when data balancing was done, with RUS producing the highest TNR of 0.865, but this is not a reliable scale of measure as the Kappa value is only 0.082, which is considered poor.[16] In fact, all the Kappa values, with the exception of the SMOTE technique, were all considered poor, with ROS placing at 0.128 and the ROS+RUS technique obtaining a Kappa of 0.127.

This means to say that although the sub data set is balanced, the number of instances was too small which could have led to the classifiers obtaining a very low Kappa rating due to the lack of Non-Fraudulent patterns in contrast to the scale of the complete dataset. At the end of the day, final data set still ended up being too imbalanced for a balanced training data set to have a reliable performance on the complete data set.

However, performing data balancing techniques on the full dataset yielded better results, which are displayed in the succeeding table.

Model	TNR	TPR	Kappa
Full DS No Balancing	0.7347	0.9999	0.8266
Full DS ROS	0.9253	0.9439	0.8692
Full DS RUS	0.8878	0.8980	0.7857
Full DS ROS+RUS	0.9218	0.9525	0.8743
Full DS SMOTE	0.9086	0.9760	0.8929

In contrast to the full dataset, which was very imbalanced, the TNR was pretty low due to a lack of instances of fraudulent transactions despite the respectable Kappa. The imbalance also resulted in a .99 TPR which was obtained due to being heavily biased towards categorizing cases as positive. We saw earlier in section 2.0 that considering the imbalanced data's accuracy for predicting non-fraudulent cases is pretty unreliable.

With the balanced datasets, on the other hand, performing ROS obtained the highest TNR, which was at .925 but the TPR is only at .944, with a Kappa of 0.869. The Kappa, while respectable, means that the capability of retaining knowledge about the non-fraudulent transactions was affected. Using RUS reduced the Kappa considerably and it did not perform as well in both TNR and TPR due to, as mentioned in an earlier hypothesis, the possibility of of important data being lost. However, combining ROS and RUS led to a respectable percentage in the TNR while mantaining a TPR detection of over 95% (0.952 in the table) with an even better Kappa than the ROS method. The SMOTE continued to perform consistently as it did with being used to only a small portion of the dataset as previously mentioned by retaining 97% (0.976 in the table) of the correct classifications of non-fraudulent cases, but the TNR is lower than ROS and ROS+RUS techniques, with only a TNR or .909. Nonetheless, the Kappa statistic is an impressive 0.893 for the SMOTE method.

7.0 Conclusions and Recommendations

To support the statement made by the publishers of the dataset, Area Under the Precision-Recall Curve (AUPRC) may be used as the measure for heavily imbalanced data as demonstrated in the earlier procedures, but a balanced dataset can actually provide a considerable classification model given the proper instances and balance. Perhaps data that spans a longer period of time can be considered regarding transactions that have been made in the place of using data balancing techniques can actually lead to classifiers performing better due to a possibility for a more set entry in the dataset in lieu of a series of estimates.

Nonetheless, data balancing techniques show promise only if it is conducted to the entire dataset before splitting the data into the training and testing dataset. Unfortunately, applying data balancing techniques to only a small portion of the dataset extracted as the training set was not enough to overcome the imbalanced nature of the complete dataset in this experiment, but the approach can be tweaked for future experiments that aims to find approaches that can act as fast and not computationally expensive in resources to make an approach more cost-efficient.

A notable finding is that many of the classifiers found the attribute `V14` prominent as a basis for classification. Further analysis of the dataset and more sophisticated techniques like Deep Learning may be applied for possibly better results. Unfortunately, this was not possible due to the hardware limitations present, but it could be something to consider. Likewise, other data balancing techniques like the ADASYN (Adaptive Synthetic), which is an extension of the SMOTE technique, and other variations can be explored in future projects.

Bibliography

[1] “What is Identity Crime” https://www.afp.gov.au/what-we-do/crime-types/fraud/identity-crime

[2] “Fraud The Facts” https://www.ukfinance.org.uk/system/files/Fraud%20The%20Facts%202019%20-%20FINAL%20ONLINE.pdf

[3] “Credit card fraud happens, but don’t let it happen to you” https://news.abs-cbn.com/business/02/25/19/credit-card-fraud-happens-but-dont-let-it-happen-to-you

[4] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015

[5] Dal Pozzolo, Andrea; Caelen, Olivier; Le Borgne, Yann-Ael; Waterschoot, Serge; Bontempi, Gianluca. Learned lessons in credit card fraud detection from a practitioner perspective, Expert systems with applications,41,10,4915-4928,2014, Pergamon

[6] Dal Pozzolo, Andrea; Boracchi, Giacomo; Caelen, Olivier; Alippi, Cesare; Bontempi, Gianluca. Credit card fraud detection: a realistic modeling and a novel learning strategy, IEEE transactions on neural networks and learning systems,29,8,3784-3797,2018,IEEE

[7] Dal Pozzolo, Andrea Adaptive Machine learning for credit card fraud detection ULB MLG PhD thesis (supervised by G. Bontempi)

[8] Carcillo, Fabrizio; Dal Pozzolo, Andrea; Le Borgne, Yann-Aël; Caelen, Olivier; Mazzer, Yannis; Bontempi, Gianluca. Scarff: a scalable framework for streaming credit card fraud detection with Spark, Information fusion,41, 182-194,2018,Elsevier

[9] Carcillo, Fabrizio; Le Borgne, Yann-Aël; Caelen, Olivier; Bontempi, Gianluca. Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization, International Journal of Data Science and Analytics, 5,4,285-300,2018,Springer International Publishing

[10] Chawla N., Bowyer K., Hall L., Kegelmeyer P. SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research (2002).

[11] Blagus, R., Lusa, L. SMOTE for high-dimensional class-imbalanced data. BMC Bioinformatics 14, 106 (2013). https://doi.org/10.1186/1471-2105-14-106

[12] Brownlee, J. "Random Oversampling and Undersampling for Imbalanced Classification". https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

[13] Pykes, K. Cohen's Kappa; Understanding Cohen's Kappa coefficient https://towardsdatascience.com/cohens-kappa-9786ceceab58

[14] Cohen J (1960) A coefficient of agreement for nominal scales. Educational and Psychological Measurement 20:37-46.

[15] Cohen J (1968) Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. Psychological Bulletin 70:213-220.

[16] Altman DG (1991) Practical statistics for medical research. London: Chapman and Hall.