Lecture Notes in Computer Science:
Credit Card Fraud Detection

Truc Huynh[1],

[1] Computer Science Department, Indiana University-Purdue University
Fort Wayne, IN, 46805,

huyntl02@pfw.edu

Abstract. The project focus on creating Fraud Detection Application to detect fraudulent credit card transactions. Thus, consumers and credit card companies are not paying for items that they did not purchase. According to Macaraeg (2019), the predicted worldwide non-cash transition growth from 2016 to 2020 is 12.7%. The increase in non-cash transactions leads to an increase in fraudulent transactions (Macaraeg, 2019). Even with EMV smart chips being implemented, the amount of money lost from credit card fraud is still very high. Therefore, implemented fraud detection (using data mining) is important.

Keywords:

Software: R Programming, R studio IDE

1. Introduction

1.1 Motivation

There are many applications out there that focused on detect fraudulent credit card transactions, but most do not cover all the issues. To mitigate the risk of fraud, using data mining is one of them.

Fraud detection is an interesting data mining project because it fights against criminal issues. Thus, my application can be used by credit card companies to stop fraudulent transactions at the time that transition occurs.

1.2 Goal

Create a meaningful subset of "Fraud" and "Non-Fraud" transactions to train the data.

Design and test possible predictive model to find the interesting pattern to prevent fraud-transactions

Learn new methods (Machine Learning Algorithm, Clustering, Decision Tree…) and apply them to solve the problem.

1.3 Challenges

Data is highly unbalanced (99.83% non-fraud vs. 0.17% fraud). This makes it hard to subset and train data (sampling).

Much research out there on fraud detection, not sure which one is good for reviewing.

Data has been applied principal component analysis so that it is hard to understand, and approach.

1.4 Data Description

My dataset and has been used for many online types of research about design credit card fraud detection applications. Therefore, I found this raw data in many articles online.

After carefully reviewing each research, I believed each author has different approaches to detect fraud patterns.

The variables are named v1 to v28 to maintain the privacy of the credit card users. The data set owner has applied principal component analysis (PCA) to the original features to reduce the features, convert them into numerical features, and hide the original features (Machine Learning Group, 2018).

Data can be download at kaggle.com

1.5 Approach

The learning goals

Sampling data, selecting variables

Data pre-processing for sequence information

Selection of useful attributes

Goals matched with DM methods

Selection of data model(s), and method(s)

Review and ask for feedback to improve

Generate pattern (Data Mining)

Interpret the model(s) based on visualization

Integrate all discovered knowledge into reports, resolve any conflicts as needed

Final Review & Improve base on given feedback

1.6 **Result**

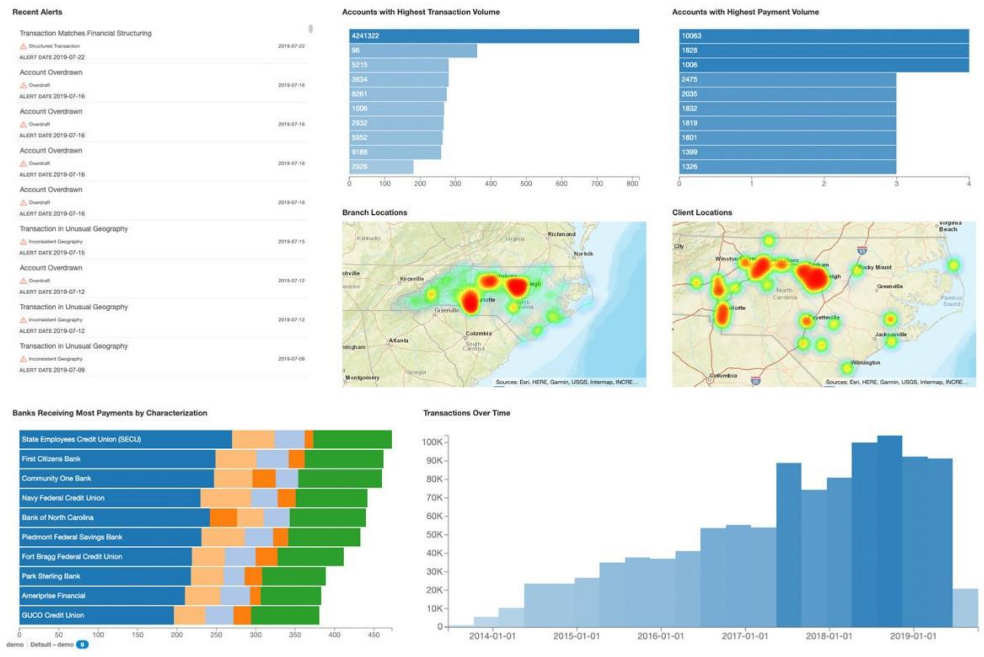
Working on this

2. Problem Statement (/Definition)

According to Machine Learning Group (2018), the datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

Since the data set is highly unbalanced, it would be highly skewed. If we use this data frame as the base for our predicted model our algorithms will probably overfit since it will "assume" that most transactions are not a fraud. Thus, create a meaningful subset is one of my goals.

Using the meaningful subset to train data. So that I can apply Data Visualization, Confusion Matrix, Clustering Methods, and Predictive Models to detect meaningful patterns in fraudulent credit card transactions.



Recent Alerts

Transaction Matches Financial Structuring
Structured Transaction 2019-07-22
ALERT DATE 2019-07-22

Account Overdrawn
Overdraft 2019-07-16
ALERT DATE 2019-07-16

Account Overdrawn
Overdraft 2019-07-16
ALERT DATE 2019-07-16

Account Overdrawn
Overdraft 2019-07-16
ALERT DATE 2019-07-16

Account Overdrawn
Overdraft 2019-07-16
ALERT DATE 2019-07-16

Transaction in Unusual Geography
Inconsistent Geography 2019-07-15
ALERT DATE 2019-07-15

Account Overdrawn
Overdraft 2019-07-12
ALERT DATE 2019-07-12

Transaction in Unusual Geography
Inconsistent Geography 2019-07-12
ALERT DATE 2019-07-12

Transaction in Unusual Geography
Inconsistent Geography 2019-07-09
ALERT DATE 2019-07-09

Image retrieved from visallo.com

3. Related Work

3.1 Patil. S., Nemade. V., & Soni, P. (Predictive Modelling)

Methods:

According to Patil, Nemade, & Soni. (2018), the proposed system is used to detect fraud on a real-time basis by analyzing incoming transactions. The system design consists of two components for fraud detection:

Designing a framework for data pre-processing.

Designing an analytical model for fraud prediction:

- Logistic regression.
- Decision tree: The decision tree uses the ID3 technique for building a decision tree by considering the entropy of the dataset
- Random Forest Decision Tree: are supervised learning algorithms used for both classification and regression problems. These two algorithms are best explained together because random forests are a bunch of decision trees combined.

Objectives:

According to Patil, Nemade, & Soni (2018), in the development of modern technology financial frauds are increasing significantly and hence fraud detection is a very important area. Fraud detection is very important to save the financial losses for the banks as they issue credit cards to the customer.

Without knowledge of cardholder use of the card information is credit card fraud. There are two types of fraud detection approaches misuse detection and anomaly detection.

More conversation can be retrieved from (PDF) Predictive Modelling for Credit Card Fraud Detection Using Data Analytics [4].

Result:

According to Patil, Nemade, & Soni (2018), the Logistic Regression Analytical Model: The optimal cut-off 0.18 is used by the Logistic Regression Analytical Model to give better performance.

Decision Tree Analytical model: To improve the performance of the decision tree, the most significant variable is taken from the trained model and the model is tuned with those most significant variables [4].

Random Forest Decision Tree Analytical model: If data points are nonlinear then the single line can be limited to the logistic regression as the outlier points are not handled effectively, in that case, the decision tree performs better [4].

Main Difference:

Using a framework for data pre-processing.

Method to train data for model prediction

The data mining methods are similar to what I plan (Logistic regression, decision tree…). Except I am going to use clustering, a subset (and/or sampling), and build models.

3.2 Gabriel Preda (Machine Learning)

Methods:

Methods are retrieved from Preda (2020):

- Random Forrest Model: For classification.
- AdaBoost model: This is used to boost the performance of any machine learning algorithm in credit, insurance, marketing, and sales.
- boost algorithm: Is used for gradient boosting on decision trees. Mostly used for search, recommendation systems, personal assistant, self-driving cars, and weather prediction…
- XG Boost algorithm: This is an implementation of gradient boosted decision trees designed for speed and performance.
- Light GBM algorithm: This is a gradient boosting framework that uses a tree-based learning algorithm.

Objectives:

Design a Predictive Model that can detect fraud transactions based on the train data.

Make sure true transaction is not rejected.

Make sure fraudulent transaction is not accepted.

(Preda, 2020)

Result:

According to Preda (2020), his work included investigated the data, checking for data unbalancing, visualizing the features, and understanding the relationship between different features. He then investigated two predictive models. The data was split into 3 parts, a train set, a validation set, and a test set. For the first three models, He only used the train and test set.

Preda started with RandomForrestClassifier, for which he obtained an AUC score of 0.85 when predicting the target for the test set [7].

He followed with an AdaBoostClassifier model, with a lower AUC score (0.83) for the prediction of the test set target values.

Then Preda followed with a CatBoostClassifier, with the AUC score after training 500 iterations 0.86 [7].

The author then experimented with an XGBoost model. In this case, He used the validation set for the validation of the training model. The best validation score obtained was 0.984 [7]. Then He used the model with the best training step, to predict the target value from the test data; the AUC score obtained was 0.974.

Preda then presented the data to a LightGBM model. The author used both train-validation split and cross-validation to evaluate the model effectiveness to predict 'Class' value, i.e. detecting if a transaction was fraudulent. [7] With the first method, the author obtained the values of AUC for the validation set around 0.974. For the test set, the score obtained was 0.946.

With the cross-validation, He obtained an AUC score for the test prediction of 0.93.

Main Difference:

The author is using the AdaBoost model, XGBoost model, CatBoostClassifier for gradient boosting on decision trees. I am going to use the Random Forest Tree, Decision Tree, Clustering, and Generalized Linear Model (GLM) Model to build my model. Since the data is making it up of various variables, it makes more sense to apply Random Forest and Decision Tree.

3.3 Pavan Sanagapati (Outliers)

Methods:

According to Sanagapati (2019), Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers (Sanagapati, 2019). Anomaly detection (also outlier detection) is the identification of rare items, events, or observations that raise suspicions by differing significantly from the majority of the data.

Method used:

- Isolation Forest algorithm: an unsupervised machine learning algorithm that identifies anomalies by isolating outliers in the data.
- Local Outlier Factor (LOF) algorithm: unsupervised anomaly detection method which computes the local density deviation of a given data point concerning its neighbors.
- Support Vector Machine (SVM) model: a supervised machine learning model that uses classification algorithms for two-group classification problems.

Objectives:

According to Sanagapati (2019), the goal is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. Also, Identifying whether a new transaction is fraudulent or not.

Result:

According to Sanagapati (2019):

The Isolation Forest detected 73 errors vs. the Local Outlier Factor detecting 97 errors vs. SVM detecting 8516 errors [8].

Isolation Forest (99.74%) is more accurate than LOF (99.65%) and SVM (70.09%) [8].

When comparing error precision & recall for 3 models, [8] the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around (27%) versus the LOF detection rate of just 2% and SVM of 0%.

So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.

We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense. We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases [8].

Main Difference:

The author using anomaly detection techniques, machine learning approaches, and Python for the analysis. I am going to use the decision-tree model and Generalized Linear Model(GLM) Model-logistic regression. I also using R Programming instead of Python.

4. Data

The detailed description of data

4.1 **Data source:**

Data is download into the data folder. Then we read the data using read.csv and store it in the data frame "R-data". Data is retrieved from "https://www.kaggle.com/mlg-ulb/creditcardfraud".

```r
Rdata <- read.csv("~/R/DataMining/FaultAnalyst.CreditCard/data/data.csv",
header=TRUE)

#Modify for mining
Rdata$hour_of_day <- (Rdata$Time/3600) %% 24 # convert to hours, then reduce
mod 24

#Data Preprocess & Transformation
Rdata$Class <- factor(ifelse(Rdata$Class == 0, "zero", "one")) # Easier for
mining data
```

4.2 Data size:

Number of rows

```r
nrow(Rdata)

## [1] 284807
```

Number of columns

```r
ncol(Rdata)

## [1] 32
```

4.3 Data attributes:

- Time: Number of seconds elapsed between this transaction and the first transaction in the dataset.
- V1 to V28: may be the result of a PCA Dimensionality reduction to protect user identities and sensitive features(v1-v28)
- Amount: Transaction amount
- Class: 1 for fraudulent transactions, 0 otherwise
- Let's check the original data attributes

```r
typeof(Rdata)

## [1] "list"
```

4.4 Main characteristics of the data:

Check the data characteristic using str() function:

```
str(Rdata)

## 'data.frame':    284807 obs. of  32 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      : Factor w/ 2 levels "one","zero": 2 2 2 2 2 2 2 2 2 2
...
##  $ hour_of_day: num  0 0 0.000278 0.000278 0.000556 ...
```

5. Data Exploration and Data Preprocess

5.1 Data Exploration:

Explore mean, standard deviation, correlation, and else using describe the () function of R.

```
#Explore the data
describe(Rdata)

##               vars      n     mean       sd   median  trimmed      mad     min
## Time             1 284807 94813.86 47488.15 84692.00 95361.03 63256.61    0.00
## V1               2 284807     0.00     1.96     0.02     0.22     1.77  -56.41
## V2               3 284807     0.00     1.65     0.07     0.07     1.04  -72.72
## V3               4 284807     0.00     1.52     0.18     0.09     1.39  -48.33
## V4               5 284807     0.00     1.42    -0.02    -0.06     1.19   -5.68
## V5               6 284807     0.00     1.38    -0.05    -0.03     0.97 -113.74
## V6               7 284807     0.00     1.33    -0.27    -0.18     0.83  -26.16
## V7             8 284807     0.00     1.24     0.04     0.01     0.83  -43.56
## V8             9 284807     0.00     1.19     0.02     0.06     0.38  -73.22
## V9            10 284807     0.00     1.10    -0.05    -0.03     0.92  -13.43
## V10           11 284807     0.00     1.09    -0.09    -0.06     0.71  -24.59
## V11           12 284807     0.00     1.02    -0.03    -0.01     1.11   -4.80
## V12           13 284807     0.00     1.00     0.14     0.10     0.75  -18.68
## V13           14 284807     0.00     1.00    -0.01     0.00     0.97   -5.79
## V14           15 284807     0.00     0.96     0.05     0.03     0.68  -19.21
## V15           16 284807     0.00     0.92     0.05     0.03     0.91   -4.50
## V16           17 284807     0.00     0.88     0.07     0.03     0.73  -14.13
## V17           18 284807     0.00     0.85    -0.07    -0.04     0.65  -25.16
## V18           19 284807     0.00     0.84     0.00     0.00     0.74   -9.50
## V19           20 284807     0.00     0.81     0.00     0.00     0.68   -7.21
## V20           21 284807     0.00     0.77    -0.06    -0.04     0.25  -54.50
## V21           22 284807     0.00     0.73    -0.03    -0.02     0.31  -34.83
## V22           23 284807     0.00     0.73     0.01     0.00     0.80  -10.93
## V23           24 284807     0.00     0.62    -0.01    -0.01     0.23  -44.81
## V24           25 284807     0.00     0.61     0.04     0.04     0.59   -2.84
## V25           26 284807     0.00     0.52     0.02     0.01     0.50  -10.30
## V26           27 284807     0.00     0.48    -0.05    -0.03     0.42   -2.60
## V27           28 284807     0.00     0.40     0.00     0.01     0.12  -22.57
## V28           29 284807     0.00     0.33     0.01     0.01     0.10  -15.43
## Amount        30 284807    88.35   250.12    22.00    41.64    29.98    0.00
## Class*        31 284807     2.00     0.04     2.00     2.00     0.00    1.00
## hour_of_day   32 284807    14.54     5.85    15.01    14.95     6.47    0.00
##                     max     range   skew kurtosis    se
## Time        172792.00 172792.00  -0.04    -1.29 88.98
## V1               2.45     58.86  -3.28    32.49  0.00
## V2              22.06     94.77  -4.62    95.77  0.00
## V3               9.38     57.71  -2.24    26.62  0.00
## V4              16.88     22.56   0.68     2.64  0.00
## V5              34.80    148.54  -2.43   206.90  0.00
## V6              73.30     99.46   1.83    42.64  0.00
## V7             120.59    164.15   2.55   405.60  0.00
## V8              20.01     93.22  -8.52   220.58  0.00
## V9              15.59     29.03   0.55     3.73  0.00
## V10             23.75     48.33   1.19    31.99  0.00
## V11             12.02     16.82   0.36     1.63  0.00
## V12              7.85     26.53  -2.28    20.24  0.00
## V13              7.13     12.92   0.07     0.20  0.00
## V14             10.53     29.74  -2.00    23.88  0.00
## V15              8.88     13.38  -0.31     0.28  0.00
## V16             17.32     31.44  -1.10    10.42  0.00
## V17              9.25     34.42  -3.84    94.80  0.00
## V18              5.04     14.54  -0.26     2.58  0.00
## V19              5.59     12.81   0.11     1.72  0.00
## V20             39.42     93.92  -2.04   271.01  0.00
## V21             27.20     62.03   3.59   207.28  0.00
## V22             10.50     21.44  -0.21     2.83  0.00
## V23             22.53     67.34  -5.88   440.08  0.00
## V24              4.58      7.42  -0.55     0.62  0.00
## V25              7.52     17.81  -0.42     4.29  0.00
## V26              3.52      6.12   0.58     0.92  0.00
## V27             31.61     54.18  -1.17   244.98  0.00
## V28             33.85     49.28  11.19   933.37  0.00
## Amount       25691.16  25691.16  16.98   845.07  0.47
## Class*           2.00      1.00 -24.00   573.87  0.00
## hour_of_day     24.00     24.00  -0.50    -0.37  0.01
```

5.2 Data Validation:

The amount and time attributes are not scaled with the rest of the features in the dataset. These can be scaled using a standard scaler. However, the classes are heavily skewed. I check the data for missing values by using **sum ()** and **mean ()** function. Then, I do a quick summary and subset the data for predictive modeling.

```
# check if data contain empty variable
sum(is.na(Rdata))

## [1] 0

mean(is.na(Rdata))

## [1] 0

#Explore the data
summary(Rdata)

##       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           hour_of_day  
##  Min.   :-15.43008   Min.   :    0.00   one :   492   Min.   : 0.00  
##  1st Qu.: -0.05296   1st Qu.:    5.60   zero:284315   1st Qu.:10.60  
##  Median :  0.01124   Median :   22.00                 Median :15.01  
##  Mean   :  0.00000   Mean   :   88.35                 Mean   :14.54  
##  3rd Qu.:  0.07828   3rd Qu.:   77.17                 3rd Qu.:19.33  
##  Max.   : 33.84781   Max.   :25691.16                 Max.   :24.00
```

5.3 Data Subset and Sampling:

5.3.1 Subset:

Split the original data into smaller subset include (test set and train set).

5.3.2 Sampling:

Create random samples so that we can achieve other mining tasks such as SVM algorithm and random forest which require a lot of resources to complete. the subset sample will include all the fraud transactions and 10,000 rows of normal transactions to test the accuracy of all the predictive models.

- Rdata: 284,807 observations of 31 variables (Validating)
- train: 199,364 observations of 31 variables (Training)
- train.subset: 7,344 observations of 31 variables (Training)
- cv: 85,443 observations of 31 variables (Testing)
- subsetData: 10,492 observations of 31 variables (Validating)
- cv.subset: 3,148 observations of 31 variables(Testing)
- fraud.data: 492 observations of 31 variables(Validating)

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

# Predictive Modeling
# Prepare Data for training
# Split data 70:30
Rdata$Class <- factor(Rdata$Class)

set.seed(1)
# Split data from vector data$Class into two sets in predefined ratio
while preserving
# relative ratios of different labels in data$Class. Used to split the
data used during
# classification into the train and test subsets.
split <- sample.split(Rdata$Class, SplitRatio = 0.7)

train <- subset(Rdata, split == T) # train data set of the original data

cv <- subset(Rdata, split == F) # test data set of the original data

# CREATE SMALL SUBSET of the ORIGINAL
# Collect all normal transaction in the original data set
data.class.0 <- subset(Rdata, Rdata$Class == 0)

# Collect all fraud transaction in the original data set
data.class.1 <- subset(Rdata, Rdata$Class == 1)

#fraud. data include all the fraud transaction
fraud.data <- data.class.1

# Get only 10,000 lines of the normal transaction in the original data
set
data.class.0 <- data.class.0[1:10000, ]

# Create the Subset Data
subsetData <- rbind(data.class.0, data.class.1)

rm(data.class.0,data.class.1) # Clean up/ un-use variable

set.seed(10)
split <- sample.split(subsetData$Class, SplitRatio = 0.7)
train.subset <- subset(subsetData, split == T)
cv.subset <- subset(subsetData, split == F)
```

5.4 Support Function (self-defined):

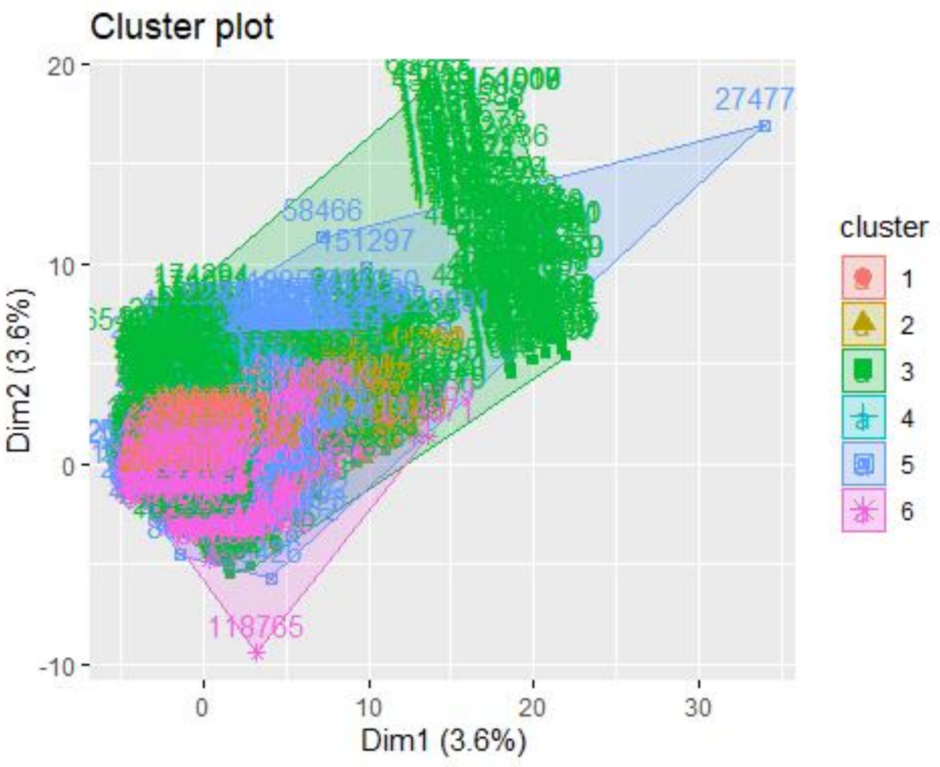
This function is to support the display description of the internal R function and print it out directly from the CRAN project library.

```
help_console <-
  function(topic,
           format = c("text", "html", "latex", "Rd"),
           lines = NULL,
           before = NULL,
           after = NULL) {
    format = match.arg(format)
    if (!is.character(topic))
      topic <- deparse(substitute(topic))
    helpfile = utils:::.getHelpFile(help(topic))

    hs <- capture.output(switch(
      format,
      text = tools:::Rd2txt(helpfile),
      html = tools:::Rd2HTML(helpfile),
      latex = tools:::Rd2latex(helpfile),
      Rd = tools:::prepare_Rd(helpfile)
    ))
    if (!is.null(lines))
      hs <- hs[lines]
    hs <- c(before, hs, after)
    cat(hs, sep = "\n")
    invisible(hs)
  }
```

5.5 Clustering for Utility:

Using Elbow method for clustering Utility.

```
trans_data<-Rdata
trans_data$Time<-NULL
trans_data$Amount<-NULL
trans_data$Class<-NULL

library(factoextra)  # library for get_dist

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

k2 <- kmeans(trans_data, centers = 6, nstart = 25)
str(k2)

## List of 9
##  $ cluster     : int [1:284807] 6 2 6 6 6 6 2 6 1 6 ...
##  $ centers     : num [1:6, 1:28] 0.176 1.448 -5.712 1.113 -7.348 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:6] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:28] "V1" "V2" "V3" "V4" ...
##  $ totss       : num 8752615
##  $ withinss    : num [1:6] 313549 1232869 1027592 708319 788641 ...
##  $ tot.withinss: num 6612492
##  $ betweenss   : num 2140122
##  $ size        : int [1:6] 16914 91942 5759 42725 3389 124078
##  $ iter        : int 5
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"

fviz_cluster(k2, data = trans_data)
```



Result: As there are too many dimensions, the clustering methods doesn't seem accurate at this point. I also try to reduce the dimension of the data. However, when applying the predictive model, the validation and test return with very low accuracy. Therefore, I will not apply the clustering methods.

6. Methodology (Proposed Methods/Approach)

6.1 Data Visualization

6.1.1 Methodology

- **Task Description:**

  Using data visualization to approach the problem. The purpose is to get a general idea about the data. Also supporting the following data mining Method in the following section.

- **Algorithm and Parameter:**

Simply convert the Time in the data set to a twenty-four hours' time system. The purpose is to estimate what time the fraud transaction amount usually occurs.

```
#Copy the Rdata to DisplayData
DisplayData<- Rdata

DisplayData$hour_of_day <- (DisplayData$Time/3600) %% 24 # convert to 
hours, then reduce mod 24
# to display only
DisplayData$Class <- factor(ifelse(DisplayData$Class == 0, "zero",
"one")) # creates issues later in caret if using 0, 1
```

Using geom_density() function of ggplot2 package to visualize the possibility of fraud transaction. From there we can summary some rules and evaluate the results. Details are given below.

```
# GEOM_DENSITY
  help_console('geom_density', "text", lines = 1:122, before = " ", after
  = " ")
##
## _S_m_o_o_t_h_e_d _d_e_n_s_i_t_y _e_s_t_i_m_a_t_e_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      Computes and draws kernel density estimate, which is a smoothed
##      version of the histogram. This is a useful alternative to the
##         histogram for continuous data that comes from an underlying
smooth
##      distribution.
##
## _U_s_a_g_e:
##
##      geom_density(
##        mapping = NULL,
##        data = NULL,
##        stat = "density",
##        position = "identity",
##        ...,
##        na.rm = FALSE,
##        orientation = NA,
##        show.legend = NA,
##        inherit.aes = TRUE,
##        outline.type = "upper"
##      )
##
##      stat_density(
##        mapping = NULL,
##        data = NULL,
##        geom = "area",
##        position = "stack",
##        ...,
##        bw = "nrd0",
##        adjust = 1,
##        kernel = "gaussian",
##        n = 512,
##        trim = FALSE,
##        na.rm = FALSE,
##        orientation = NA,
##        show.legend = NA,
##        inherit.aes = TRUE
##      )
##
## _A_r_g_u_m_e_n_t_s:
##
##  mapping: Set of aesthetic mappings created by 'aes()' or 'aes_()'. If
##           specified and 'inherit.aes = TRUE' (the default), it is
##           combined with the default mapping at the top level of the
##           plot. You must supply 'mapping' if there is no plot mapping.
##
##     data: The data to be displayed in this layer. There are three
##           options:
##
##           If 'NULL', the default, the data is inherited from the plot
##           data as specified in the call to 'ggplot()'.
##
##              A 'data.frame', or other object, will override the plot
data.
##           All objects will be fortified to produce a data frame. See
##           'fortify()' for which variables will be created.
##
##           A 'function' will be called with a single argument, the plot
##           data. The return value must be a 'data.frame', and will be
##           used as the layer data. A 'function' can be created from a
##           'formula' (e.g. '~ head(.x, 10)').
##
## position: Position adjustment, either as a string, or the result of a
##           call to a position adjustment function.
##
##      ...: Other arguments passed on to 'layer()'. These are often
##           aesthetics, used to set an aesthetic to a fixed value, like
##           'colour = "red"' or 'size = 3'. They may also be parameters
##           to the paired geom/stat.
##
##    na.rm: If 'FALSE', the default, missing values are removed with a
##           warning. If 'TRUE', missing values are silently removed.
##
## orientation: The orientation of the layer. The default ('NA')
##           automatically determines the orientation from the aesthetic
##           mapping. In the rare event that this fails it can be given
##               explicitly by setting 'orientation' to either '"x"' or
'"y"'.
##           See the _Orientation_ section for more detail.
##
## show.legend: logical. Should this layer be included in the legends?
##           'NA', the default, includes if any aesthetics are mapped.
##           'FALSE' never includes, and 'TRUE' always includes. It can
##           also be a named logical vector to finely select the
##           aesthetics to display.
##
## inherit.aes: If 'FALSE', overrides the default aesthetics, rather than
##                 combining with them. This is most useful for helper
functions
##           that define both data and aesthetics and shouldn't inherit
##           behaviour from the default plot specification, e.g.
##           'borders()'.
##
## outline.type: Type of the outline of the area; '"both"' draws both the
##           upper and lower lines, '"upper"'/'"lower"' draws the
##               respective lines only. '"full"' draws a closed polygon
around
##           the area.
##
## geom, stat: Use to override the default connection between
##           'geom_density' and 'stat_density'.
##
##       bw: The smoothing bandwidth to be used. If numeric, the standard
##           deviation of the smoothing kernel. If character, a rule to
##           choose the bandwidth, as listed in 'stats::bw.nrd()'.
##
##   adjust: A multiplicate bandwidth adjustment. This makes it possible
##           to adjust the bandwidth while still using the a bandwidth
##           estimator. For example, 'adjust = 1/2' means use half of the
##           default bandwidth.
##
##   kernel: Kernel. See list of available kernels in 'density()'.
##
##        n: number of equally spaced points at which the density is to
be
##           estimated, should be a power of two, see 'density()' for
##           details
##
##     trim: If 'FALSE', the default, each density is computed on the
full
##           range of the data. If 'TRUE', each density is computed over
##            the range of that group: this typically means the estimated
x
##              values will not line-up, and hence you won't be able to
stack
##           density values. This parameter only matters if you are
##           displaying multiple densities in one plot or if you are
##           manually adjusting the scale limits.
##
```

```
#GGPLOT
help_console('ggplot', "text", lines = 1:26, before = " ", after = " ")
##
## _C_r_e_a_t_e _a _n_e_w _g_g_p_l_o_t
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'ggplot()' initializes a ggplot object. It can be used to declare
##      the input data frame for a graphic and to specify the set of plot
##      aesthetics intended to be common throughout all subsequent layers
##      unless specifically overridden.
##
## _U_s_a_g_e:
##
##           ggplot(data = NULL, mapping = aes(), ..., environment =
parent.frame())
##
## _A_r_g_u_m_e_n_t_s:
##
##        data: Default dataset to use for plot. If not already a
data.frame,
##           will be converted to one by 'fortify()'. If not specified,
##           must be supplied in each layer added to the plot.
##
##  mapping: Default list of aesthetic mappings to use for plot. If not
##           specified, must be supplied in each layer added to the plot.
##
##      ...: Other arguments passed on to methods. Not currently used.
```

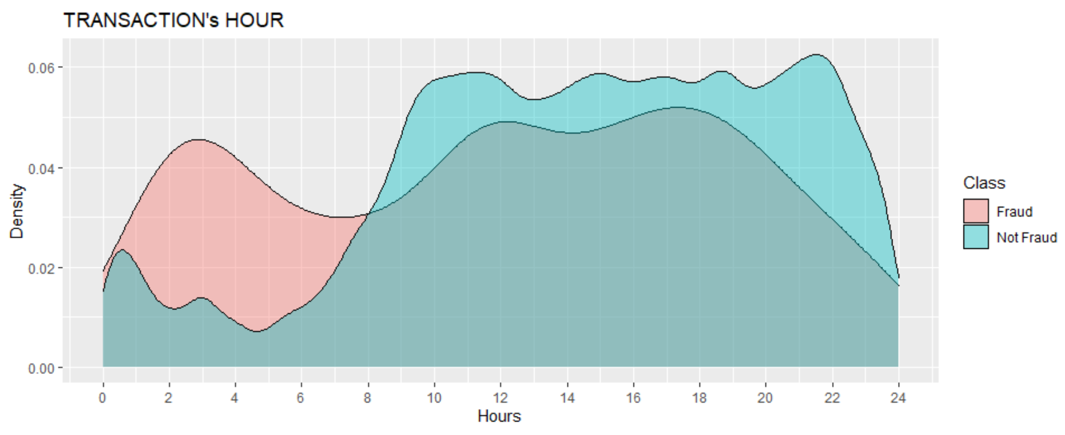
```
##
## environment: DEPRECATED. Used before tidy evaluation.
##
##
```

6.1.2 Transaction Hour Visualization:

- **Task Description:**

  Using Density Chart to visualize the pattern

```
  ggplot(Rdata, aes(x = hour_of_day, fill = Class)) +
    geom_density(alpha = 0.4) +
    scale_x_continuous(limits = c(0, 24), breaks = seq(0, 24, 2)) +
    labs(title = "TRANSACTION's HOUR",
         x = "Hours",
         y = "Density",
         col = "Class") +
    scale_fill_discrete(labels = c("Fraud", "Not Fraud"))
```



- **Evaluation:**

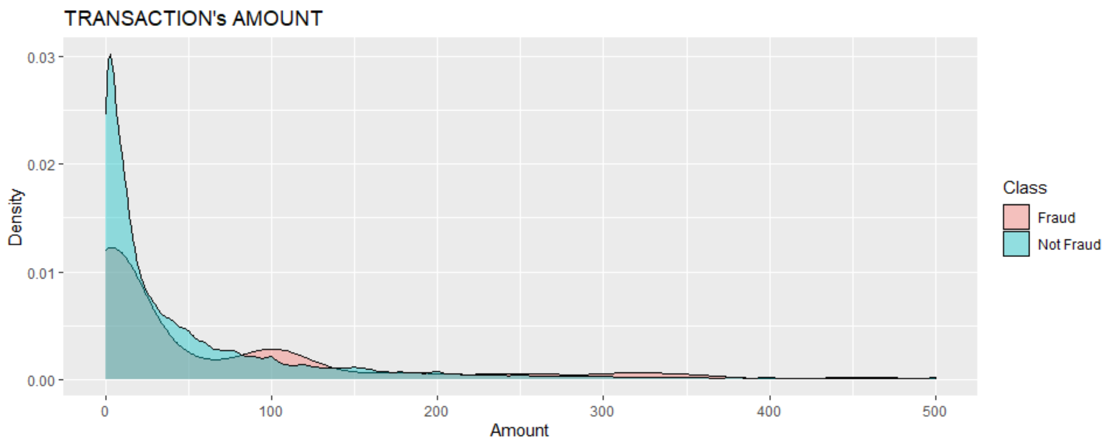
  According to the density-chart, fraud transactions happened from 0 to 8 AM (early morning and during sleep time) while non-fraud transactions happen during active-time. This is also common sense since humans usually purchase in the daytime, not when they sleep. Therefore, transactions that happen at night (peak at 3 PM) have more chances to happen in fraud-transactions.

6.1.3 Transaction Amount Visualization

- **Task Description:**

  Using Density Chart to visualize the pattern

```
ggplot(Rdata, aes(x = Amount, fill = Class)) +
  geom_density(alpha = 0.4) +
  scale_x_continuous(limits = c(0, 500), breaks = seq(0, 500, 100)) +
  labs(title = "TRANSACTION's AMOUNT",
       x = "Amount",
       y = "Density",
       col = "Class") +
  scale_fill_discrete(labels = c("Fraud", "Not Fraud"))
```



- **Evaluation**

According to the density-chart, a higher portion of the chart is at fraud-transaction. In the chart the fraud transaction value from 90 to over 100 and 300 to 350 is way more density than the nonfraud transaction. Therefore, fraud transactions may happen to be a large transaction (a higher proportion of fraudulent transactions take a very large value).

```
# Simple remove Display Data
rm(DisplayData)
```

6.2 Generalized Linear Model

6.2.1 Methodology

- **Task Description:**

In statistics, the generalized linear model (GLM) is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution.

The glm is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

We are going to use the glm() function in R to construct our Generalized Linear Predictive Model. Please read the description of the glm() function and its parameter to understand this function. While the main function is glm() has been displaying in greater detail, the support table() function and predict() function also describe in shorter detail.

- **Algorithm and Parameter:**

```
# GLM FUNCTION
help_console('glm',
             "text",
             lines = 1:110,
             before = "<blockquote>",
             after = "</blockquote>")
## <blockquote>
## _F_i_t_t_i_n_g _G_e_n_e_r_a_l_i_z_e_d _L_i_n_e_a_r _M_o_d_e_l_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'glm' is used to fit generalized linear models, specified by
##      giving a symbolic description of the linear predictor and a
##      description of the error distribution.
##
## _U_s_a_g_e:
##
##      glm(formula, family = gaussian, data, weights, subset,
##          na.action, start = NULL, etastart, mustart, offset,
##          control = list(...), model = TRUE, method = "glm.fit",
##            x = FALSE, y = TRUE, singular.ok = TRUE, contrasts = NULL,
...)
##
##      glm.fit(x, y, weights = rep.int(1, nobs),
##              start = NULL, etastart = NULL, mustart = NULL,
##              offset = rep.int(0, nobs), family = gaussian(),
##              control = list(), intercept = TRUE, singular.ok = TRUE)
##
##      ## S3 method for class 'glm'
##      weights(object, type = c("prior", "working"), ...)
##
## _A_r_g_u_m_e_n_t_s:
##
##  formula: an object of class '"formula"' (or one that can be coerced
to
##           that class): a symbolic description of the model to be
##           fitted.  The details of model specification are given under
##           'Details'.
##
##   family: a description of the error distribution and link function to
##           be used in the model.  For 'glm' this can be a character
##           string naming a family function, a family function or the
##           result of a call to a family function.  For 'glm.fit' only
##           the third option is supported.  (See 'family' for details of
##           family functions.)
##
##     data: an optional data frame, list or environment (or object
##           coercible by 'as.data.frame' to a data frame) containing the
##           variables in the model.  If not found in 'data', the
##           variables are taken from 'environment(formula)', typically
##           the environment from which 'glm' is called.
##
##  weights: an optional vector of 'prior weights' to be used in the
##           fitting process.  Should be 'NULL' or a numeric vector.
##
##   subset: an optional vector specifying a subset of observations to be
##           used in the fitting process.
##
## na.action: a function which indicates what should happen when the data
##               contain 'NA's.   The  default  is  set  by  the  'na.action'
setting
##           of 'options', and is 'na.fail' if that is unset.  The
##              'factory-fresh'  default  is  'na.omit'.   Another  possible
value
##           is 'NULL', no action.  Value 'na.exclude' can be useful.
##
##    start: starting values for the parameters in the linear predictor.
##
## etastart: starting values for the linear predictor.
##
##  mustart: starting values for the vector of means.
##
##   offset: this can be used to specify an _a priori_ known component to
##           be included in the linear predictor during fitting.  This
##           should be 'NULL' or a numeric vector of length equal to the
##           number of cases.  One or more 'offset' terms can be included
##           in the formula instead or as well, and if more than one is
##           specified their sum is used.  See 'model.offset'.
##
##  control: a list of parameters for controlling the fitting process.
##           For 'glm.fit' this is passed to 'glm.control'.
##
##    model: a logical value indicating whether _model frame_ should be
##           included as a component of the returned value.
##
##   method: the method to be used in fitting the model.  The default
##           method '"glm.fit"' uses iteratively reweighted least squares
##           (IWLS): the alternative '"model.frame"' returns the model
##           frame and does no fitting.
##
##           User-supplied fitting functions can be supplied either as a
##           function or a character string naming a function, with a
##           function which takes the same arguments as 'glm.fit'.  If
##           specified as a character string it is looked up from within
##           the 'stats' namespace.
##
##     x, y: For 'glm': logical values indicating whether the response
##             vector  and  model  matrix  used  in  the  fitting  process  should
be
##           returned as components of the returned value.
##
##           For 'glm.fit': 'x' is a design matrix of dimension 'n * p',
##           and 'y' is a vector of observations of length 'n'.
##
## singular.ok: logical; if 'FALSE' a singular fit is an error.
##
## contrasts: an optional list. See the 'contrasts.arg' of
##           'model.matrix.default'.
##
## intercept: logical. Should an intercept be included in the _null_
##           model?
##
##   object: an object inheriting from class '"glm"'.
##
##     type: character, partial matching allowed.  Type of weights to
##           extract from the fitted model object.  Can be abbreviated.
##
##          ...:  For  'glm':  arguments  to  be  used  to  form  the  default
'control'
```

```
##            argument if it is not supplied directly.
##
##            For 'weights': further arguments passed to or from other
##            methods.
## </blockquote>
# PREDICT FUNCTION
help_console(
  'predict',
  "text",
  lines = 1:28,
  before = " ",
  after = " "
)
##
## _M_o_d_e_l _P_r_e_d_i_c_t_i_o_n_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'predict' is a generic function for predictions from the results
##      of various model fitting functions.  The function invokes
##      particular _methods_ which depend on the 'class' of the first
##      argument.
##
## _U_s_a_g_e:
##
##      predict (object, ...)
##
## _A_r_g_u_m_e_n_t_s:
##
##   object: a model object for which prediction is desired.
##
##      ...: additional arguments affecting the predictions produced.
##
## _D_e_t_a_i_l_s:
##
##      Most prediction methods which are similar to those for linear
##      models have an argument 'newdata' specifying the first place to
##      look for explanatory variables to be used for prediction.  Some
##      considerable attempts are made to match up the columns in
##      'newdata' to those used for fitting, for example that they are of
##      comparable types and that any factors have the same level set in
##      the same order (or can be transformed to be so).
##
```

6.2.2 *Result:*

Build model and test train data frame.

```
#Base line accuracy
table(cv$Class)

##
##      0     1
## 85295   148

#Generalized Linear Model(GLM) Model: logistic regression
glm.model <- glm(Class ~ ., data = train, family = "binomial")
```

```
# Test the train model
glm.predict <- predict(glm.model, train, type = "response")

table(train$Class, glm.predict > 0.5)

##
##      FALSE   TRUE
##    0 198995     25
##    1    117    227
```

Result:

- Fraud: Find 227 out of 344 fraud case(227+117), accurate percentage is 67.96%.
- Non-Fraud: accurate percentage is 99.99%

In test 2 (test):

```
glm.predict <- predict(glm.model, cv, type = "response")
table(cv$Class, glm.predict > 0.5)

##
##     FALSE  TRUE
##    0 85279    16
##    1    69    79
```

Result:

- Fraud: Find 79 out of 148 fraud case(69+79), accurate percentage is 53.39%.
- Non-Fraud: accurate percentage is 99.99%

In test 3 (test):

```
glm.predict <- predict(glm.model, cv.subset, type = "response")
table(cv.subset$Class, glm.predict > 0.5)

##
##     FALSE TRUE
##    0  3000    0
##    1    58   90
```

Result:

- Fraud: Find 90 out of 148 fraud case(58+90), accurate percentage is 60.81%.
- Non-Fraud: accurate percentage is 100%

In test 4 (validate):

```
glm.predict <- predict(glm.model, Rdata, type = "response")
table(Rdata$Class, glm.predict > 0.5)

##
##      FALSE   TRUE
##   0 284274     41
##   1    186    306
```

Result:

- ○ Fraud: Find 306 out of 492 fraud case, accurate percentage is 62.19%.
- ○ Non-Fraud: accurate percentage is 99.99%

<u>In test 5 (validate):</u>

```
glm.predict <- predict(glm.model, subsetData, type = "response")
table(subsetData$Class, glm.predict > 0.5)

##
##     FALSE  TRUE
##   0 10000     0
##   1   186   306
```

Result:

- ○ Fraud: Find 306 out of 492 fraud case, accurate percentage is 62.19%.
- ○ Non-Fraud: accurate percentage is 100%

6.3 Predictive Model Using Decision Tree (Regression Trees)

6.3.1 Methodology:

- **Task Description:**

Decision tree learning is one of the predictive modeling approaches used in statistics, data mining, and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).

We are going to use the rpart()function of the rpart package in R to construct our Decision Tree and Predictive Model. Please read the description of the rpart function and its parameter to understand this function.

While the main function is rpart() have been displaying in greater detail, the support prp() function and predict() function also describe in shorter detail

- **Algorithm and Parameter:**

```
# RPART FUNCTION
help_console('rpart', "text", lines = 1:80, before = " ", after = " ")

##
## _R_e_c_u_r_s_i_v_e _P_a_r_t_i_t_i_o_n_i_n_g _a_n_d
_R_e_g_r_e_s_s_i_o_n _T_r_e_e_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      Fit a 'rpart' model
##
## _U_s_a_g_e:
##
##      rpart(formula, data, weights, subset, na.action = na.rpart,
method,
##            model = FALSE, x = FALSE, y = TRUE, parms, control, cost,
...)
##
## _A_r_g_u_m_e_n_t_s:
##
##  formula: a formula, with a response but no interaction terms.  If
this
##           a a data frame, that is taken as the model frame (see
##           'model.frame).'
##
##     data: an optional data frame in which to interpret the variables
##           named in the formula.
##
##  weights: optional case weights.
##
##   subset: optional expression saying that only a subset of the rows of
##           the data should be used in the fit.
##
## na.action: the default action deletes all observations for which 'y'
is
##           missing, but keeps those in which one or more predictors are
##           missing.
##
##   method: one of '"anova"', '"poisson"', '"class"' or '"exp"'.  If
##           'method' is missing then the routine tries to make an
##           intelligent guess.  If 'y' is a survival object, then
'method
##           = "exp"' is assumed, if 'y' has 2 columns then 'method =
##           "poisson"' is assumed, if 'y' is a factor then 'method =
##           "class"' is assumed, otherwise 'method = "anova"' is
assumed.
##           It is wisest to specify the method directly, especially as
##           more criteria may added to the function in future.
##
##           Alternatively, 'method' can be a list of functions named
##           'init', 'split' and 'eval'.  Examples are given in the file
##           'tests/usersplits.R' in the sources, and in the vignettes
##           'User Written Split Functions'.
##
##    model: if logical: keep a copy of the model frame in the result?
If
```

```
##            the input value for 'model' is a model frame (likely from an
##            earlier call to the 'rpart' function), then this frame is
##            used rather than constructing new data.
##
##        x: keep a copy of the 'x' matrix in the result.
##
##        y: keep a copy of the dependent variable in the result.  If
##            missing and 'model' is supplied this defaults to 'FALSE'.
##
##    parms: optional parameters for the splitting function.
##            Anova splitting has no parameters.
##            Poisson splitting has a single parameter, the coefficient of
##            variation of the prior distribution on the rates.  The
##            default value is 1.
##            Exponential splitting has the same parameter as Poisson.
##            For classification splitting, the list can contain any of:
##            the vector of prior probabilities (component 'prior'), the
##            loss matrix (component 'loss') or the splitting index
##            (component 'split').  The priors must be positive and sum to
##            1.  The loss matrix must have zeros on the diagonal and
##            positive off-diagonal elements.  The splitting index can be
##            'gini' or 'information'.  The default priors are
proportional
##            to the data counts, the losses default to 1, and the split
##            defaults to 'gini'.
##
##  control: a list of options that control details of the 'rpart'
##            algorithm.  See 'rpart.control'.
##
##     cost: a vector of non-negative costs, one for each variable in the
##            model. Defaults to one for all variables.  These are
scalings
##            to be applied when considering splits, so the improvement on
##            splitting on a variable is divided by its cost in deciding
##            which split to choose.
##
##      ...: arguments to 'rpart.control' may also be specified in the
##            call to 'rpart'.  They are checked against the list of valid
##

# PRP FUNCTION
help_console('prp', "text", lines = 1:30, before = " ", after = " ")

##
## _P_l_o_t _a_n _r_p_a_r_t _m_o_d_e_l.
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      Plot an 'rpart' model.
##
##      First-time users should use 'rpart.plot' instead, which provides
a
##      simplified interface to this function.
##
##      For an overview, please see the package vignette Plotting rpart
##      trees with the rpart.plot package.
##
##      The arguments of this function are a superset of those of
##      'rpart.plot' and some of the arguments have different defaults.
In
##      detail the different defaults are:
##
##                          |  'rpart.plot'  |   'prp'   |
##
##        'type'            |      '2'       |    '0'    |
##        'extra'           |    '"auto"'    |    '0'    |
##        'fallen.leaves'   |     'TRUE'     |  'FALSE'  |
##        'varlen'          |      '0'       |   '-8'    |
##        'faclen'          |      '0'       |    '3'    |
##        'box.palette'     |    '"auto"'    |    '0'    |
##
##      The defaults are different for historical reasons: for backwards
##      compatibility the defaults of 'prp' haven't changed, whereas the
##      defaults of 'rpart.plot' were changed when 'type="auto"' and
##      'box.palette' were introduced in version 2.0.0 of this package.
##
##
```

```
# PREDICT FUNCTION
help_console('predict', "text", lines = 1:28, before = " ", after = " ")
```

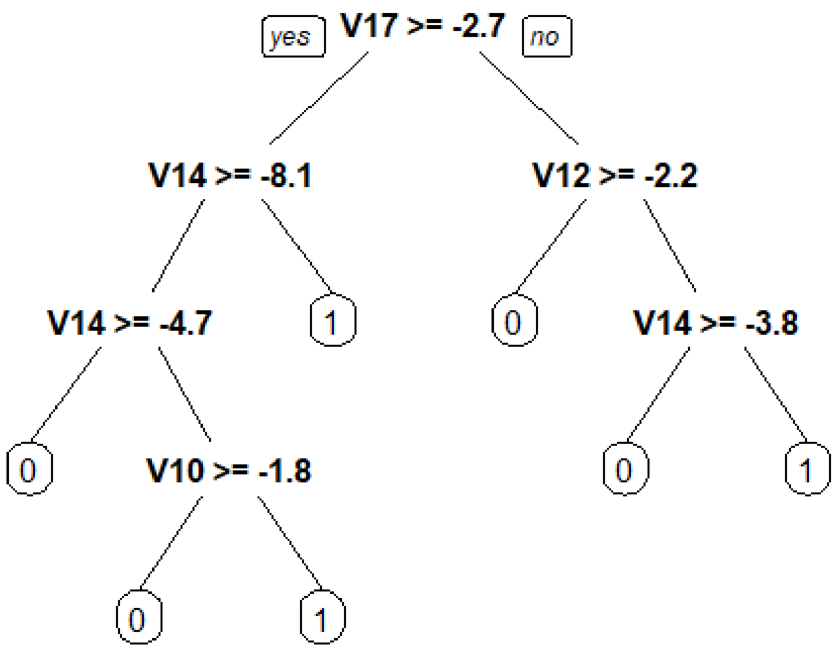
```
##
## _M_o_d_e_l _P_r_e_d_i_c_t_i_o_n_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'predict' is a generic function for predictions from the results
##      of various model fitting functions.  The function invokes
##      particular _methods_ which depend on the 'class' of the first
##      argument.
##
## _U_s_a_g_e:
##
##      predict (object, ...)
##
## _A_r_g_u_m_e_n_t_s:
##
##   object: a model object for which prediction is desired.
##
##      ...: additional arguments affecting the predictions produced.
##
## _D_e_t_a_i_l_s:
##
##      Most prediction methods which are similar to those for linear
##      models have an argument 'newdata' specifying the first place to
##      look for explanatory variables to be used for prediction.  Some
##      considerable attempts are made to match up the columns in
##      'newdata' to those used for fitting, for example that they are of
##      comparable types and that any factors have the same level set in
##      the same order (or can be transformed to be so).
##
```

6.3.2 *Result:*

- Decision tree model 1

Build the Decision tree Model using the train data subset from the original data. The method is Classification with the minimum number of the bucket is 20. Achieve the decision tree model with the accuracy is 99.93%.

```
#Decision tree model
tree.model <-
  rpart(Class ~ .,
        data = train,
        method = "class",
        minbucket = 20)
prp(tree.model)
```



```
tree.predict <- predict(tree.model, cv, type = "class")
confusionMatrix(as.factor(cv$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 85275    20
##          1    44   104
##
##                Accuracy : 0.9993
##                  95% CI : (0.999, 0.9994)
##     No Information Rate : 0.9985
##     P-Value [Acc > NIR] : 2.098e-09
##
##                   Kappa : 0.7643
##
##  Mcnemar's Test P-Value : 0.00404
##
##             Sensitivity : 0.9995
##             Specificity : 0.8387
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.7027
##              Prevalence : 0.9985
##          Detection Rate : 0.9980
##    Detection Prevalence : 0.9983
##       Balanced Accuracy : 0.9191
##
##        'Positive' Class : 0
##

# Result
# 99.93 % accuracy (best) using decision tree.
```

- Evaluation of Decision Tree model 1:

  In test 1, I simply test the model by compare the Class column in test(cv) dataset. I will also collect the mean in percentage of the comparison. The percentage is 99.93% which is match with the decision tree model accuracy percentage that found above. Prove that the decision tree model 1 is accuracy.

```
# This function simply test the accurate of the model by compare the
predicting model with the data
tree.predict <- predict(tree.model, cv, type = "class")
confusionMatrix(as.factor(cv$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 85275    20
##          1    44   104
##
##                Accuracy : 0.9993
##                  95% CI : (0.999, 0.9994)
##     No Information Rate : 0.9985
##     P-Value [Acc > NIR] : 2.098e-09
##
##                   Kappa : 0.7643
##
##  Mcnemar's Test P-Value : 0.00404
##
##             Sensitivity : 0.9995
##             Specificity : 0.8387
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.7027
##              Prevalence : 0.9985
##          Detection Rate : 0.9980
##    Detection Prevalence : 0.9983
##       Balanced Accuracy : 0.9191
##
##        'Positive' Class : 0
##

mean(tree.predict == cv$Class)

## [1] 0.999251
```

Result:

  - Achieve an overall accurate of 99.93 (test with the test data frame)

- Fraud: Find 104 out of 148 fraud case, accurate percentage is 70.27%
- Non-Fraud: accurate percentage is 99.99%

In test 2 (test):

```
# This function simply test the accurate of the model by compare the predicting
model with the data
tree.predict <- predict(tree.model, subsetData, type = "class")
confusionMatrix(as.factor(subsetData$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9998    2
##          1  119  373
##
##                Accuracy : 0.9885
##                  95% CI : (0.9862, 0.9904)
##     No Information Rate : 0.9643
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8545
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9882
##             Specificity : 0.9947
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.7581
##              Prevalence : 0.9643
##          Detection Rate : 0.9529
##    Detection Prevalence : 0.9531
##       Balanced Accuracy : 0.9915
##
##        'Positive' Class : 0
##

mean(tree.predict == subsetData$Class)

## [1] 0.9884674
```

Result:

- Achieve an overall accurate of 98.85 (test with the test data frame)
- Fraud: Find 373 out of 492 fraud case, accurate percentage is 75.81%.
- Non-Fraud: accurate percentage is 99.9999%

In test 3 (validation):

```
# This function simply test the accurate of the model by compare the
predicting model with the data
```

```
tree.predict <- predict(tree.model, fraud.data, type = "class")
confusionMatrix(as.factor(fraud.data$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1 119 373
##
##                Accuracy : 0.7581
##                  95% CI : (0.7178, 0.7953)
##     No Information Rate : 0.7581
##     P-Value [Acc > NIR] : 0.5246
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7581
##              Prevalence : 0.2419
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##

mean(tree.predict == fraud.data$Class)

## [1] 0.7581301
```

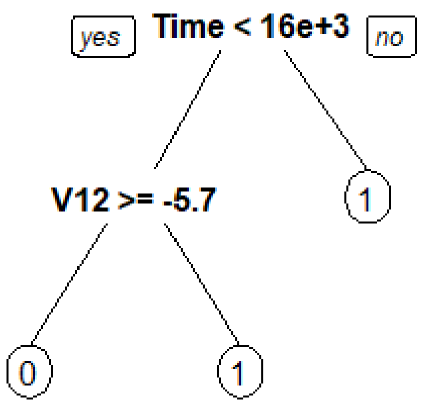
Result:

- Achieve an overall accurate of 75.81 (test with the ALL FRAUD data frame)
- Fraud: Find 373 out of 492 fraud case, accurate percentage is 75.81%.

- Decision tree model 2

Build the Decision tree Model using the train data subset from the subset data. The Method is Classification with the minimum number of bucket is 20.

```
#Decision tree model
tree.model.2 <-
  rpart(Class ~ .,
        data = train.subset,
        method = "class",
        minbucket = 20)
prp(tree.model.2)
```



```
tree.predict.2 <- predict(tree.model.2, cv.subset, type = "class")
confusionMatrix(as.factor(cv.subset$Class), tree.predict.2)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1    1  147
##
##                Accuracy : 0.9997
##                  95% CI : (0.9982, 1)
##     No Information Rate : 0.9533
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9964
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9997
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9932
##              Prevalence : 0.9533
##          Detection Rate : 0.9530
##    Detection Prevalence : 0.9530
##       Balanced Accuracy : 0.9998
##
##        'Positive' Class : 0
##
# Result
# 99.97 % accuracy (best) using the decision tree.
```

- Evaluation of Decision Tree model 2

Simply test the model by comparing the Class column in the test(cv. subset) dataset. I will also collect the mean in the percentage of the comparison. The percentage is 99.96823% which matches the decision tree model accuracy percentage found above. Prove that the decision tree model 2 is accurate.

In test 1(test):

```
# This function simply test the accurate of the model by compare the
predicting model with the data
tree.predict <- predict(tree.model.2, cv.subset, type = "class")
confusionMatrix(as.factor(cv.subset$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1    1  147
##
##                Accuracy : 0.9997
##                  95% CI : (0.9982, 1)
##     No Information Rate : 0.9533
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9964
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9997
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9932
##              Prevalence : 0.9533
##          Detection Rate : 0.9530
##    Detection Prevalence : 0.9530
##       Balanced Accuracy : 0.9998
##
##        'Positive' Class : 0
##

mean(tree.predict == cv.subset$Class)

## [1] 0.9996823
```

Result:

- Achieve an overall accurate of 99.97 (test with the test data frame)
- Fraud: Find 147 out of 148 fraud case, accurate percentage is 99.32%.
- Non-Fraud: accurate percentage is 100%

In Test 2 (test):

```
# This function simply test the accurate of the model by compare the predicting
model with the data
tree.predict <- predict(tree.model.2, Rdata, type = "class")
confusionMatrix(as.factor(Rdata$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0  10240 274075
##          1      3    489
##
##                Accuracy : 0.0377
##                  95% CI : (0.037, 0.0384)
##     No Information Rate : 0.964
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 1e-04
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.999707
##             Specificity : 0.001781
##          Pos Pred Value : 0.036016
##          Neg Pred Value : 0.993902
##              Prevalence : 0.035965
##          Detection Rate : 0.035954
##    Detection Prevalence : 0.998273
##       Balanced Accuracy : 0.500744
##
##        'Positive' Class : 0
##

mean(tree.predict == subsetData$Class)

## [1] 0.08065462
```

Result:

- Achieve an overall accurate of 0.0377 (test with the original data frame)
- Fraud: Find 489 out of 491 fraud case, accurate percentage is 99.59%
- Non-Fraud: accurate percentage is 3.601%

In Test 3 (validate):

```
# This function simply test the accurate of the model by compare the predicting
model with the data
tree.predict <- predict(tree.model.2, fraud.data, type = "class")
confusionMatrix(as.factor(fraud.data$Class), tree.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1   3 489
##                                           
##                Accuracy : 0.9939          
##                  95% CI : (0.9823, 0.9987)
##     No Information Rate : 0.9939          
##     P-Value [Acc > NIR] : 0.6472          
##                                           
##                   Kappa : 0               
##                                           
##  Mcnemar's Test P-Value : 0.2482          
##                                           
##             Sensitivity : 0.000000        
##             Specificity : 1.000000        
##          Pos Pred Value :      NaN        
##          Neg Pred Value : 0.993902        
##              Prevalence : 0.006098        
##          Detection Rate : 0.000000        
##    Detection Prevalence : 0.000000        
##       Balanced Accuracy : 0.500000        
##                                           
##        'Positive' Class : 0               
##                                           

mean(tree.predict == fraud.data$Class)

## [1] 0.9939024
```

Result:

- Achieve an overall accurate of 99.39 (test with the fraud data)
- Fraud: Find 489 out of 491 fraud case, accurate percentage is 99.59%

6.4 Classification and Regression analysis using Support-Vector Machines Model (SVMs)

6.4.1 Methodology

- **Task Description:**

  In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

  We are going to use the SVM()function in R to construct a classification model. Please read the description of the SVM() function and its parameter to understand this function.

While the main function is the SVM() have been displaying in greater detail, the support the confusion matrix() function and predict() function also describe in shorter detail

- **Algorithm and Parameter:**

```
# SVM FUNCTION
help_console('svm', "text", lines = 1:129, before = " ", after = " ")

##
## _S_u_p_p_o_r_t _V_e_c_t_o_r _M_a_c_h_i_n_e_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'svm' is used to train a support vector machine. It can be used
to
##      carry out general regression and classification (of nu and
##      epsilon-type), as well as density-estimation. A formula interface
##      is provided.
##
## _U_s_a_g_e:
##
##      ## S3 method for class 'formula'
##      svm(formula, data = NULL, ..., subset, na.action =
##      na.omit, scale = TRUE)
##      ## Default S3 method:
##      svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
##      "radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 /
ncol(x),
##      coef0 = 0, cost = 1, nu = 0.5,
##      class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon
= 0.1,
##      shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
##      ..., subset, na.action = na.omit)
##
## _A_r_g_u_m_e_n_t_s:
##
##  formula: a symbolic description of the model to be fit.
##
##     data: an optional data frame containing the variables in the
model.
##           By default the variables are taken from the environment
which
##           'svm' is called from.
##
##        x: a data matrix, a vector, or a sparse matrix (object of class
##           'Matrix' provided by the 'Matrix' package, or of class
##           'matrix.csr' provided by the 'SparseM' package, or of class
##           'simple_triplet_matrix' provided by the 'slam' package).
##
##        y: a response vector with one label for each row/component of
##           'x'. Can be either a factor (for classification tasks) or a
##           numeric vector (for regression).
##
##    scale: A logical vector indicating the variables to be scaled. If
##           'scale' is of length 1, the value is recycled as many times
##           as needed.  Per default, data are scaled internally (both
'x'
##           and 'y' variables) to zero mean and unit variance. The
center
```

```
##           and scale values are returned and used for later
predictions.
##
##     type: 'svm' can be used as a classification machine, as a
##           regression machine, or for novelty detection.  Depending of
##           whether 'y' is a factor or not, the default setting for
##           'type' is 'C-classification' or 'eps-regression',
##           respectively, but may be overwritten by setting an explicit
##           value.
##           Valid options are:
##
##             • 'C-classification'
##
##             • 'nu-classification'
##
##             • 'one-classification' (for novelty detection)
##
##             • 'eps-regression'
##
##             • 'nu-regression'
##
##   kernel: the kernel used in training and predicting. You might
##           consider changing some of the following parameters,
depending
##           on the kernel type.
##
##           linear: u'*v
##
##           polynomial: (gamma*u'*v + coef0)^degree
##
##           radial basis: exp(-gamma*|u-v|^2)
##
##           sigmoid: tanh(gamma*u'*v + coef0)
##
##   degree: parameter needed for kernel of type 'polynomial' (default:
3)
##
##    gamma: parameter needed for all kernels except 'linear' (default:
##           1/(data dimension))
##
##    coef0: parameter needed for kernels of type 'polynomial' and
##           'sigmoid' (default: 0)
##
##     cost: cost of constraints violation (default: 1)-it is the
##           'C'-constant of the regularization term in the Lagrange
##           formulation.
##
##       nu: parameter needed for 'nu-classification', 'nu-regression',
##           and 'one-classification'
##
## class.weights: a named vector of weights for the different classes,
##           used for asymmetric class sizes. Not all factor levels have
##           to be supplied (default weight: 1). All components have to
be
##           named. Specifying '"inverse"' will choose the weights
##           _inversely_ proportional to the class distribution.
##
## cachesize: cache memory in MB (default 40)
##
## tolerance: tolerance of termination criterion (default: 0.001)
##
##  epsilon: epsilon in the insensitive-loss function (default: 0.1)
##
```

```
## shrinking: option whether to use the shrinking-heuristics (default:
##           'TRUE')
##
##    cross: if a integer value k>0 is specified, a k-fold cross
##           validation on the training data is performed to assess the
##           quality of the model: the accuracy rate for classification
##           and the Mean Squared Error for regression
##
##   fitted: logical indicating whether the fitted values should be
##           computed and included in the model or not (default: 'TRUE')
##
## probability: logical indicating whether the model should allow for
##           probability predictions.
##
##      ...: additional parameters for the low level fitting function
##           'svm.default'
##
##   subset: An index vector specifying the cases to be used in the
##           training sample.  (NOTE: If given, this argument must be
##           named.)
##
## na.action: A function to specify the action to be taken if 'NA's are
##           found. The default action is 'na.omit', which leads to
##           rejection of cases with missing values on any required
##           variable. An alternative is 'na.fail', which causes an error
##           if 'NA' cases are found. (NOTE: If given, this argument must
##           be named.)
##
```

```
# PREDICT FUNCTION
help_console('predict', "text", lines = 1:28, before = " ", after = " ")
```

```
##
## _M_o_d_e_l _P_r_e_d_i_c_t_i_o_n_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'predict' is a generic function for predictions from the results
##      of various model fitting functions.  The function invokes
##      particular _methods_ which depend on the 'class' of the first
##      argument.
##
## _U_s_a_g_e:
##
##      predict (object, ...)
##
## _A_r_g_u_m_e_n_t_s:
##
##   object: a model object for which prediction is desired.
##
##      ...: additional arguments affecting the predictions produced.
##
## _D_e_t_a_i_l_s:
##
##      Most prediction methods which are similar to those for linear
##      models have an argument 'newdata' specifying the first place to
##      look for explanatory variables to be used for prediction.  Some
##      considerable attempts are made to match up the columns in
##      'newdata' to those used for fitting, for example that they are of
##      comparable types and that any factors have the same level set in
##      the same order (or can be transformed to be so).
##
```

6.4.2 Result

Build the SVM Model using the train data subset from the subset data. Match the prediction Model with the test data to test the model.

```
svm.model <- svm(Class ~ ., data = train.subset, kernel = "radial", cost = 1,
gamma = 0.1)
svm.predict <- predict(svm.model, cv.subset)
confusionMatrix(cv.subset$Class, svm.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1   44  104
##
##                Accuracy : 0.986
##                  95% CI : (0.9813, 0.9898)
##     No Information Rate : 0.967
##     P-Value [Acc > NIR] : 1.362e-11
##
##                   Kappa : 0.8183
##
##  Mcnemar's Test P-Value : 9.022e-11
##
##             Sensitivity : 0.9855
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7027
##              Prevalence : 0.9670
##          Detection Rate : 0.9530
##    Detection Prevalence : 0.9530
##       Balanced Accuracy : 0.9928
##
##        'Positive' Class : 0
##
```

In test 1 (test):

```
svm.predict <- predict(svm.model, cv.subset)
confusionMatrix(cv.subset$Class, svm.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1   44  104
##
##                Accuracy : 0.986
##                  95% CI : (0.9813, 0.9898)
##     No Information Rate : 0.967
##     P-Value [Acc > NIR] : 1.362e-11
```

```
##
##                  Kappa : 0.8183
##
##  Mcnemar's Test P-Value : 9.022e-11
##
##             Sensitivity : 0.9855
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7027
##              Prevalence : 0.9670
##          Detection Rate : 0.9530
##    Detection Prevalence : 0.9530
##       Balanced Accuracy : 0.9928
##
##        'Positive' Class : 0
##

mean(svm.predict == cv.subset$Class)

## [1] 0.9860229
```

Result:

- Achieve an overall accurate of 0.986 (test with the sample test data frame)
- Fraud: Find 104 out of 148 fraud case, accurate percentage is 70.27%.
- Non-Fraud: accurate percentage is 100%

In test 2, achieve an overall accurate of 33.07% (test).

```
svm.predict <- predict(svm.model, cv)
confusionMatrix(cv$Class, svm.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 28126 57169
##          1    16   132
##
##                Accuracy : 0.3307
##                  95% CI : (0.3276, 0.3339)
##     No Information Rate : 0.6706
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0011
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.999431
##             Specificity : 0.002304
##          Pos Pred Value : 0.329750
##          Neg Pred Value : 0.891892
##              Prevalence : 0.329366
##          Detection Rate : 0.329179
##    Detection Prevalence : 0.998268
##       Balanced Accuracy : 0.500868
##
```

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

mean(svm.predict == cv$Class)

## [1] 0.3307234
```

Result:

- Achieve an overall accurate of 0.3307 (test with the test data frame)
- Fraud: Find 132 out of 148 fraud case, accurate percentage is 89.18%.
- Non-Fraud: accurate percentage is 32.97%

In test 3, achieve an accurate of 33.07% (validation).

```
svm.predict <- predict(svm.model, Rdata)
confusionMatrix(Rdata$Class, svm.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0  93673 190642
##          1     45    447
##
##                Accuracy : 0.3305
##                  95% CI : (0.3287, 0.3322)
##     No Information Rate : 0.6709
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0012
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.999520
##             Specificity : 0.002339
##          Pos Pred Value : 0.329469
##          Neg Pred Value : 0.908537
##              Prevalence : 0.329058
##          Detection Rate : 0.328900
##    Detection Prevalence : 0.998273
##       Balanced Accuracy : 0.500930
##
##        'Positive' Class : 0
##

mean(svm.predict == Rdata$Class)

## [1] 0.3304694
```

Result:

- Achieve an overall accurate of 0.3307 (test with the test data frame)
- Fraud: Find 132 out of 148 fraud case, accurate percentage is 89.18%.
- Non-Fraud: accurate percentage is 32.97%

6.5 Predictive Model Using Random Forest

6.5.1 Methodology

- **Task Description**

  Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.

  We are going to use the randomForest()function of package randomForest in R to construct a decision tree. Please read the description of the randomForest() function and its parameter to understand this function.

  While the main function is randomForest() have been displayed in greater detail, the support confusion matrix() function and predict() function also describe in shorter detail

- **Algorithm and Parameter:**

```
# SVM FUNCTION
help_console('rf', "text", lines = 1:129, before = " ", after = " ")

##
## _T_h_e _F _D_i_s_t_r_i_b_u_t_i_o_n
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      Density, distribution function, quantile function, and random
##      generation for the F distribution with 'df1' and 'df2' degrees of
##      freedom (and optional non-centrality parameter 'ncp').
##
## _U_s_a_g_e:
##
##      df(x, df1, df2, ncp, log = FALSE)
##      pf(q, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
##      qf(p, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
##      rf(n, df1, df2, ncp)
##
## _A_r_g_u_m_e_n_t_s:
##
##     x, q: vector of quantiles.
##
##        p: vector of probabilities.
##
##        n: number of observations. If 'length(n) > 1', the length is
##           taken to be the number required.
##
## df1, df2: degrees of freedom.  'Inf' is allowed.
```

```
##
##      ncp: non-centrality parameter. If omitted the central F is
##           assumed.
##
## log, log.p: logical; if TRUE, probabilities p are given as log(p).
##
## lower.tail: logical; if TRUE (default), probabilities are P[X <= x],
##           otherwise, P[X > x].
##
## _D_e_t_a_i_l_s:
##
##      The F distribution with 'df1 =' n1 and 'df2 =' n2 degrees of
##      freedom has density
##
##      f(x) = Gamma((n1 + n2)/2) / (Gamma(n1/2) Gamma(n2/2))
##          (n1/n2)^(n1/2) x^(n1/2 - 1)
##          (1 + (n1/n2) x)^-(n1 + n2)/2
##
##      for x > 0.
##
##      It is the distribution of the ratio of the mean squares of n1 and
##      n2 independent standard normals, and hence of the ratio of two
##      independent chi-squared variates each divided by its degrees of
##      freedom.  Since the ratio of a normal and the root mean-square of
##      m independent normals has a Student's t_m distribution, the
square
##      of a t_m variate has a F distribution on 1 and m degrees of
##      freedom.
##
##      The non-central F distribution is again the ratio of mean squares
##      of independent normals of unit variance, but those in the
##      numerator are allowed to have non-zero means and 'ncp' is the sum
##      of squares of the means.  See Chisquare for further details on
##      non-central distributions.
##
## _V_a_l_u_e:
##
##      'df' gives the density, 'pf' gives the distribution function 'qf'
##      gives the quantile function, and 'rf' generates random deviates.
##
##      Invalid arguments will result in return value 'NaN', with a
##      warning.
##
##      The length of the result is determined by 'n' for 'rf', and is
the
##      maximum of the lengths of the numerical arguments for the other
##      functions.
##
##      The numerical arguments other than 'n' are recycled to the length
##      of the result.  Only the first elements of the logical arguments
##      are used.
##
## _N_o_t_e:
##
##      Supplying 'ncp = 0' uses the algorithm for the non-central
##      distribution, which is not the same algorithm used if 'ncp' is
##      omitted.  This is to give consistent behaviour in extreme cases
##      with values of 'ncp' very near zero.
##
##      The code for non-zero 'ncp' is principally intended to be used
for
##      moderate values of 'ncp': it will not be highly accurate,
##      especially in the tails, for large values.
```

```
##
## _S_o_u_r_c_e:
##
##      For the central case of 'df', computed _via_ a binomial
##      probability, code contributed by Catherine Loader (see 'dbinom');
##      for the non-central case computed _via_ 'dbeta', code contributed
##      by Peter Ruckdeschel.
##
##      For 'pf', _via_ 'pbeta' (or for large 'df2', _via_ 'pchisq').
##
##      For 'qf', _via_ 'qchisq' for large 'df2', else _via_ 'qbeta'.
##
## _R_e_f_e_r_e_n_c_e_s:
##
##      Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
##      Language_.  Wadsworth & Brooks/Cole.
##
##      Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) _Continuous
##      Univariate Distributions_, volume 2, chapters 27 and 30.  Wiley,
##      New York.
##
## _S_e_e _A_l_s_o:
##
##      Distributions for other standard distributions, including
'dchisq'
##      for chi-squared and 'dt' for Student's t distributions.
##
## _E_x_a_m_p_l_e_s:
##
##      ## Equivalence of pt(.,nu) with pf(.^2, 1,nu):
##      x <- seq(0.001, 5, len = 100)
##      nu <- 4
##      stopifnot(all.equal(2*pt(x,nu) - 1, pf(x^2, 1,nu)),
##                ## upper tails:
##                all.equal(2*pt(x,     nu, lower=FALSE),
##                            pf(x^2, 1,nu, lower=FALSE)))
##
##      ## the density of the square of a t_m is 2*dt(x, m)/(2*x)
##      # check this is the same as the density of F_{1,m}
##      all.equal(df(x^2, 1, 5), dt(x, 5)/x)
##
##      ## Identity:  qf(2*p - 1, 1, df) == qt(p, df)^2  for  p >= 1/2
##      p <- seq(1/2, .99, length = 50); df <- 10
##      rel.err <- function(x, y) ifelse(x == y, 0, abs(x-
y)/mean(abs(c(x,y))))
##      quantile(rel.err(qf(2*p - 1, df1 = 1, df2 = df), qt(p, df)^2),
.90)  # ~= 7e-9
##

# PREDICT FUNCTION
help_console('predict', "text", lines = 1:28, before = " ", after = " ")

##
## _M_o_d_e_l _P_r_e_d_i_c_t_i_o_n_s
##
## _D_e_s_c_r_i_p_t_i_o_n:
##
##      'predict' is a generic function for predictions from the results
##      of various model fitting functions.  The function invokes
##      particular _methods_ which depend on the 'class' of the first
##      argument.
##
## _U_s_a_g_e:
```

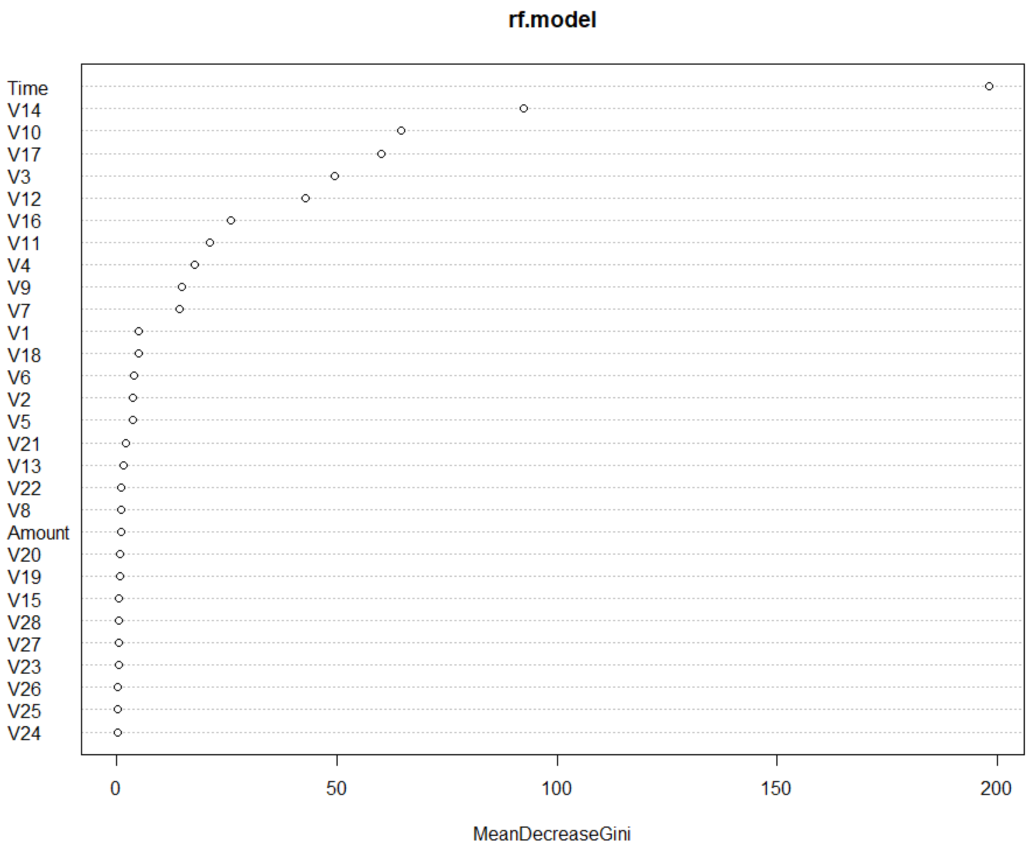
```
##
##      predict (object, ...)
##
## _A_r_g_u_m_e_n_t_s:
##
##   object: a model object for which prediction is desired.
##
##      ...: additional arguments affecting the predictions produced.
##
## _D_e_t_a_i_l_s:
##
##      Most prediction methods which are similar to those for linear
##      models have an argument 'newdata' specifying the first place to
##      look for explanatory variables to be used for prediction.  Some
##      considerable attempts are made to match up the columns in
##      'newdata' to those used for fitting, for example that they are of
##      comparable types and that any factors have the same level set in
##      the same order (or can be transformed to be so).
##
```

6.5.2 Result

Build the Random Forest Decision Tree Model using the train data subset from the subset data.

```
set.seed(100)
rf.model <- randomForest(Class ~ ., data = train.subset,
                         ntree = 2000, nodesize = 20)

rf.predict <- predict(rf.model, cv.subset)
confusionMatrix(cv.subset$Class, rf.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1    0  148
##
##                Accuracy : 1
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.953
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.000
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 1.000
##              Prevalence : 0.953
##          Detection Rate : 0.953
##    Detection Prevalence : 0.953
##       Balanced Accuracy : 1.000
##
##        'Positive' Class : 0
##
```

```r
varImpPlot(rf.model)
```



In test 1, achieve an accurate of 100% (test).

```r
rf.predict <- predict(rf.model, cv.subset)
confusionMatrix(cv.subset$Class, rf.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1    0  148
##
##                Accuracy : 1
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.953
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.000
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 1.000
##              Prevalence : 0.953
##          Detection Rate : 0.953
##    Detection Prevalence : 0.953
##       Balanced Accuracy : 1.000
##
##        'Positive' Class : 0
##

mean(rf.predict == cv.subset$Class)

## [1] 1
```

Result:

- Achieve an overall accurate of 100% (test with the test data frame)
- Fraud: Find 148 out of 148 fraud case, accurate percentage is 100%.
- Non-Fraud: accurate percentage is 100%

In test 2, achieve an overall accurate of 3.82% (test).

```
rf.predict <- predict(rf.model, cv)
confusionMatrix(cv$Class, rf.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0  3119 82176
##          1     0   148
##
##                Accuracy : 0.0382
##                  95% CI : (0.037, 0.0395)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 1e-04
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.001798
##          Pos Pred Value : 0.036567
##          Neg Pred Value : 1.000000
##              Prevalence : 0.036504
##          Detection Rate : 0.036504
##    Detection Prevalence : 0.998268
##       Balanced Accuracy : 0.500899
##
##        'Positive' Class : 0
##

mean(rf.predict == cv$Class)

## [1] 0.03823602
```

Result:

- Achieve an overall accurate of 0.0382 (test with the test data frame).
- Fraud: Find 148 out of 148 fraud case, accurate percentage is 100%.
- Non-Fraud: accurate percentage is 0.037%.

In test 3, achieve an overall accurate of 3.77% (validate)

```
rf.predict <- predict(rf.model, Rdata)
confusionMatrix(Rdata$Class, rf.predict)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0  10259 274056
##          1      2    490
##
##                Accuracy : 0.0377
##                  95% CI : (0.037, 0.0384)
##     No Information Rate : 0.964
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 1e-04
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.999805
##             Specificity : 0.001785
##          Pos Pred Value : 0.036083
##          Neg Pred Value : 0.995935
##              Prevalence : 0.036028
##          Detection Rate : 0.036021
##    Detection Prevalence : 0.998273
##       Balanced Accuracy : 0.500795
##
##        'Positive' Class : 0
##

mean(rf.predict == Rdata$Class)

## [1] 0.03774135
```

Result:

- Achieve an overall accurate of 0.0377 (test with the entire data frame).
- Fraud: Find 490 out of 492 fraud case, accurate percentage is 99.59%.
- Non-Fraud: accurate percentage is 0.036%.

In test 4, achieve an overall accurate of 99.59% (validate)

```
rf.predict <- predict(rf.model, fraud.data)
confusionMatrix(fraud.data$Class, rf.predict)

## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0   0   0
##          1   2 490
##
##                Accuracy : 0.9959
##                  95% CI : (0.9854, 0.9995)
##     No Information Rate : 0.9959
##     P-Value [Acc > NIR] : 0.6767
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.000000
##             Specificity : 1.000000
##          Pos Pred Value :      NaN
##          Neg Pred Value : 0.995935
##              Prevalence : 0.004065
##          Detection Rate : 0.000000
##    Detection Prevalence : 0.000000
##       Balanced Accuracy : 0.500000
##
##        'Positive' Class : 0
##
```

```
mean(rf.predict == fraud.data$Class)

## [1] 0.995935
```

Result:

- Achieve an overall accurate of 99.59.0377 (test with the entire data frame).
- Fraud: Find 490 out of 492 fraud case, accurate percentage is 99.59%.

7. Conclusion

Many algorithms and data-mining methods have been testing and applied to the training dataset (fraud credit card). I took different approach to analyze interesting pattern in the data set.

I also take different approach for the training data (split the data, create factor value, get sample from the original data.). Un-success approaches include:

- Clustering Models using k means.
- DBSCAN
- Dimension Reduction
- Elbow method

- Shilhouette method

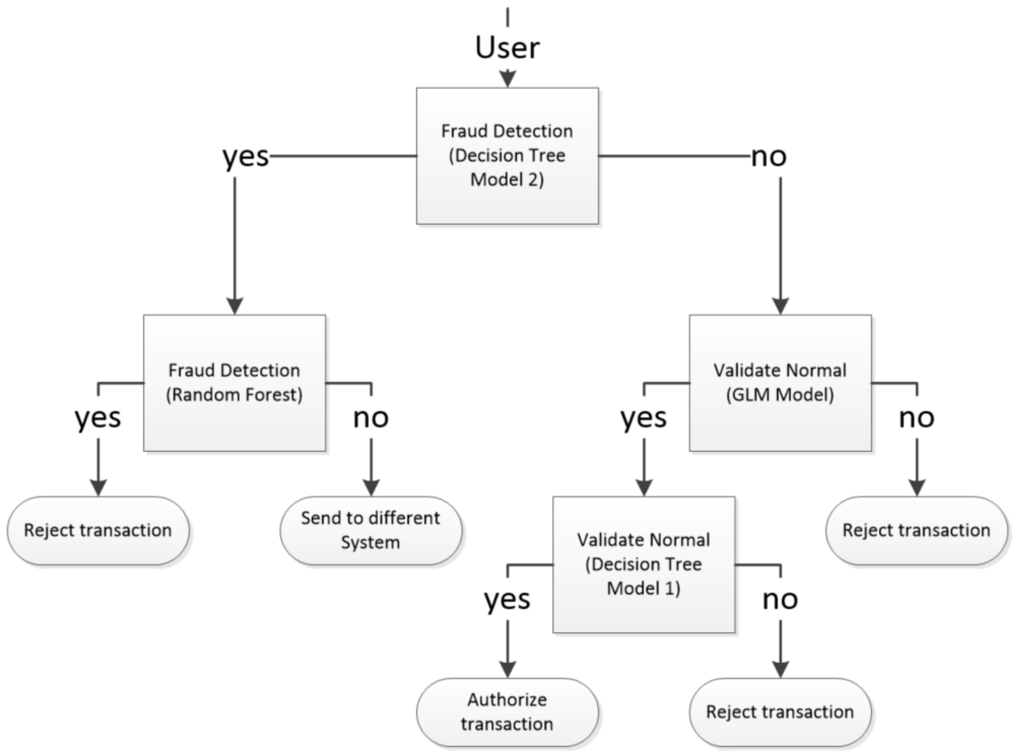
Success approaches include:

- Random Forest Predictive Model (99.56 % average on fraud transactions)
- Regression Decision Tree Predictive Model (models 1 with 99.9999% on non-fraud transactions and model 2 with 99.59% test case on fraud transactions)
- Support-Vector Machines Models (89.18 % average on fraud transactions)
- Generalized Linear Model (99.988 % average on non-fraud transactions)
- Density-graph, that hypothesis can be made from there.

In general, the research model will be able to recognize and detect fraud and non-fraud transactions in the data set. In other to use it properly for a fraud detection application, choosing what models for what purpose is importance (Random Forest to detect fraud (99.59%); GLM and Decision Tree model 1 (99.999 %) to verify non-fraud transaction). Please refer to the result table:

Result	**Fraud Transaction**			**Non-Fraud Transaction**		
	Test 1	Test 2	Validate	Test 1	Test 2	Validate
GLM Model	67.96%	62.19 %	62.19 %	100 %	99.99%	99.99%
Decision Tree 1	70.27%	75.81%	75.81%	99.99%	99.99%	99.99%
Decision Tree 2	99.32%	99.59%	99.59%	100%	3.601%	3.79%
SVM Model	70.27%	89.18%	89.18%	100%	32.97%	32.97%
Random Forest	100%	100%	99.59%	100%	3.7%	3.6%

There are no perfect models that can predict all the fraud and non-fraud transaction in one algorithm (with respect to the accuracy percentage). My theory is a combine models and use them to vote to decide if transaction is fraud. The model on the same category can be swapped (i.e., Random Forest and Decision Tree Model 2 can be swap). Diagram:


User
Fraud Detection (Decision Tree Model 2)
yes
no
Fraud Detection (Random Forest)
yes
no
Reject transaction
Send to different System
Validate Normal (GLM Model)
yes
no
Reject transaction
Validate Normal (Decision Tree Model 1)
yes
no
Authorize transaction
Reject transaction

References

Kurgan, L., & Musilek, P.(n.d.) A survey of Knowledge Discovery and Data Mining process models, The Knowledge Engineering Review, Vol. 21:1, 1–24. Retrieved from http://biomine.cs.vcu.edu/papers/KER-KDDM2006.pdf [1]

Macaraeg, R., (2019, Sep. 5)Credit Card Fraud Detection: Staying Vigilant in the Virtual World. Retrieved from: https://towardsdatascience.com/credit-card-fraud-detection-a1c7e1b75f59 [2]

Machine Learning Group,(2018)Credit Card Fraud Detection: Anonymized credit card transactions labeled as fraudulent or genuine. Retrieved from: https://www.kaggle.com/mlg-ulb/creditcardfraud [3]

Patil. S., Nemade. V., & Soni, P. (2018) International Conference on Computational Intelligence and Data Science. Predictive Modelling For Credit Card Fraud Detection Using Data Analytics. Retrieved from https://www.researchgate.net/publication/325663203_Predictive_Modelling_For_Credit_Card_Fraud_Detection_Using_Data_Analytic [4]

Pozzolo A., Caelen O., Johnson R. & Bontempi G.(2015) Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE [5]

Pozzolo A.; Olivier C.; Yann-Ael B.; Serge; Gianluca B.(2014) Learned lessons in credit card fraud detection from a practitioner perspective, Expert systems with applications,41,10,4915-4928, Pergamon [6]

Preda. G., (2020) Credit Card Fraud Detection Predictive Models. Retrieved from https://www.kaggle.com/gpreda/credit-card-fraud-detection-predictive-models [7]

Sanagapati. P., (2019) Anomaly Detection - Credit Card Fraud Analysis. Retrieved from https://www.kaggle.com/pavansanagapati/anomaly-detection-credit-card-fraud-analysis [8]

Visallo (n.d.) Credit Card Fraud Detection 2019 Retrieved from https://www.visallo.com/blog/credit-card-fraud-detection-2019/ [9]