Credit Card Fraud

Dhaval Patel

Joshua Monteiro

Visesh Jaiswal

Wei-Tung Lin

University of Massachusetts – Dartmouth

Abstract

This report provides some analysis about the credit card fraud. With the advantage of science and technology, we can use different models to know which factors are important in this issue, and use some models to predict the outcome. In this report, we will execute some different kinds of models, including Naïve Bayes Prediction, decision tree, random forest, and XGBoost. We want to get the best model in order to give this company some suggestions, and the company can modify the strategy in order to get the best result.

Keywords: *forecast, Naïve Bayes, decision tree, random forest, XGBoost, Data Balancement*

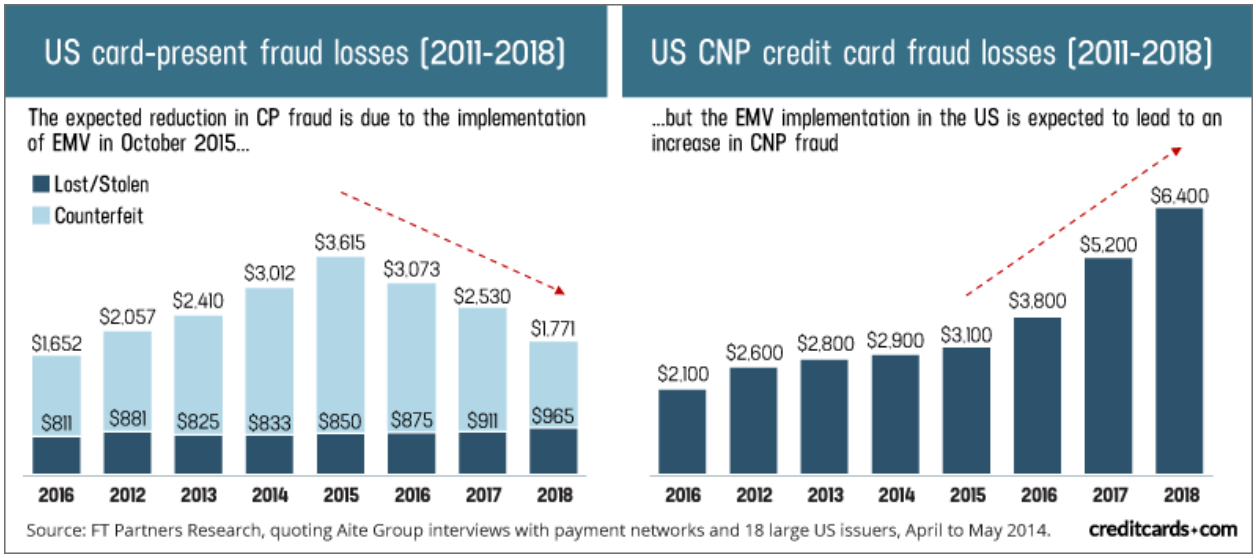
Credit Card Fraud

Introduction

Nowadays there are billions upon billions of credit card transactions. With the increase in technology and increased variety of goods out there, these transactions are not going to slow down any time soon. A simple example involves looking at the order processes of many companies. Amazon is a perfect scenario. You can store credit cards so easily and purchase items literally in just a few clicks. With the relative ease and frequency of these transactions, it is evident that some problems could arise. One major problem is credit card fraud. Credit card fraud is theft of one's credit card (usually taken in terms of stealing their identity and making some sort of transaction for your own personally use). Over the years, there have been many plenty of different features and mechanisms incorporated to try and decrease the likelihood of credit card fraud.

Consumer teachings, EMV implementation (card chips), and more advanced card material are a few of the features that has helped decrease credit card fraud. It is important to note that this has only helped with present credit cards (those being used in person). As previously described ordering processes such as Amazon's, do not need a credit card to physically be present to make purchases. Many organizations and companies have been studying these different aspects of credit cards. One investigation (below), demonstrates that even as present-credit card fraud has decreased, the overall not present credit card fraud is expected to

increase.



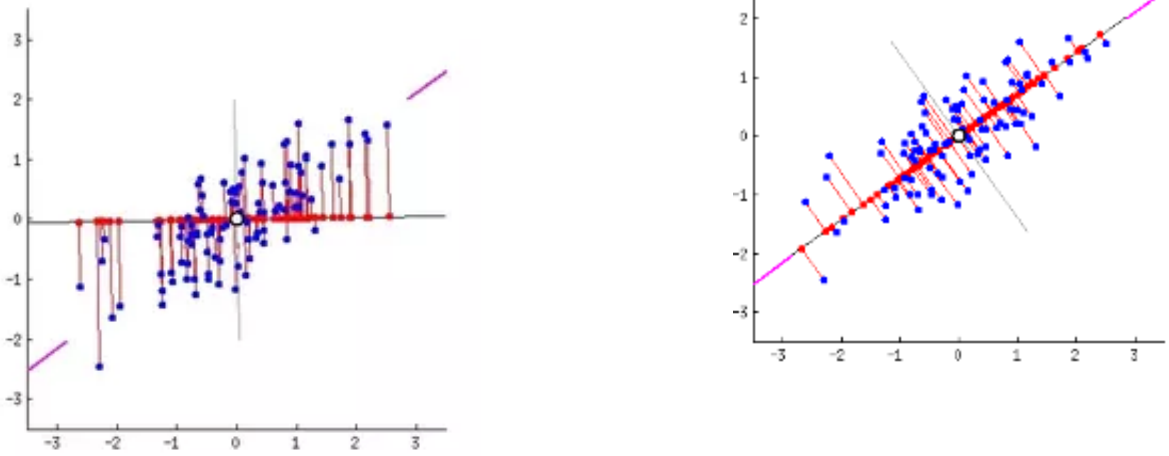
(Red Ventures Company, 2019).

With this in mind, credit card companies have to be more diligent. Security is one part of the discussion. The other important component involves; their ability to accurately detect credit card fraud in the first place. This will be a crucial aspect of their business processes. This is where we come in. Based on information that can be gathered from the everyday transactions, we are going to try and predict whether credit card fraud is present. For credit card companies, there is plenty of value in this. Overall, we will help establish which models are most accurate, key elements that need to be understood, and which variables are the most important.

Data

Now we begin with the data itself. The data is a set of 284,807 credit card transactions that took place in Europe on September 2013. This is just for a two day span, so you can see how vast credit card transactions really are. Compared to the billions of transactions in the world, this is a decent sample size (not too big when taken in reality). There are a total of 31 variables in this

data set. Some of these include elements such as time, amount spent, and so on. One factor that has to be considered is the confidentiality of the information. To avoid leaking important information, some of these variables can not be explicitly stated. That is why the data itself has been converted. All of the data has already been converted using PCA (principal component analysis). PCA is a tool that reduces a large data set into a more meaningful data set by emphasizing the variation between the values. This is done by adjusting the dimensions of the variables. None of the values are actually (physically) changed, instead the dimensions are altered in which it brings out the most variance among the variables. Then, the observations (or values) are converted to match that specific dimension. Below, an example is shown to help illustrate the conversion.



The left is an original plot of two variables. The right is the same thing except the dimension has shifted. As you can see in the graphs, this does two things. The first is that the variation among predicted data point increases (red dots). They variation can be seen in which the red dots are farther apart in the right graph. The more variation between these means it is much easier to find the strong patterns (easy to distinguish physically). The other component involves the residual

error (red lines between the actual values [blue dots] and line of best fit). Notice how the residual error decreases in the right graph.

Overall, the example above involves just two variables (two dimensions). Our data set has 31 variables, so you can so how complex it can get (trying to find the most useful dimensions or even eliminate useless dimensions). All of the values have been converted using these principles. These PCA transformations also explain that these values have been normalized (and converted to numerical variables if they were not already). That is the initial criteria as variables have to be numerical for PCA transformations to occur and so scale can not impact these dimensions. As the concepts are understood, we can see how this connects to the confidentiality piece. As Kaggle explains (where we acquired the data set), "due to confidentiality issues, we cannot provide the original features and more background information about the data" (Machine Learning Group, 2018). Since all the values have been converted, the original nature of the variables can not be deciphered. Hence the data can be analyzed more efficiently and without any confidentiality problems. Note: the credit card companies would have first-hand access to this specific information.

Project methodology

In this report, there are several different methodologies involved. We want to find the best accuracy rate by these models. Set seeds (1234) have been used to keep the data set consistent. Since these are based on predictive simulations, we will only focus on the testing data. The models are in the following:

(1) Naïve Bayes

Naïve Bayes is a supervised machine learning method. Naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

(2) Decision Tree

Decision trees are versatile Machine Learning algorithm that can perform both classification and regression tasks. They are very powerful algorithms, capable of fitting complex datasets. Besides, decision trees are fundamental components of random forests, which are among the most potent Machine Learning algorithms available today.

(3) Random Forest

Random forest is an ensemble of decision trees. The basic principle behind ensemble methods is that a group of “weak learners” can come together to form a “strong learner.” As a result, random forest grow many decision trees.

(4) XGBOOST

XGBoost is short for extreme gradient boosting. This is an efficient implementation of the gradient boosting framework. The package includes efficient linear model solver and tree learning algorithms. The package can automatically do parallel computation on a single machine which could be more than 10 times faster than existing gradient boosting packages. It supports various objective functions, including regression, classification and ranking.

Discussion on result

This report will discuss these kinds of model respectively.

(1) Naïve Bayes

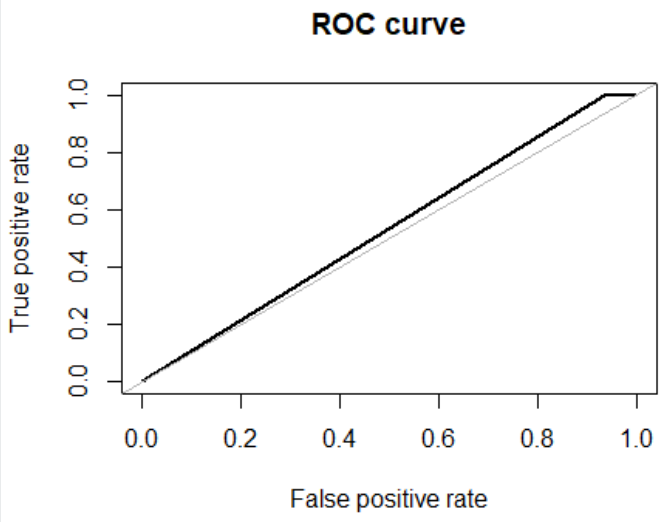
Here we divided our dataset into two parts, that is training and testing. As PCA has already been applied to the dataset, we don't need to worry about the correlation between variables (maximum variation among them within the dimension). 80% of data is training and 20% is testing.

```
set.seed(1234)

Naive_Model <- naiveBayes(Class ~ .,data = train)

#Make predictions in test data

Naive_Pred <- predict(Naive_Model,test,type = "class")

#Performance Matrics

confusionMatrix(Naive_Pred,test$Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 55435    12
##          1  1242    87
##
##                Accuracy : 0.9779
##                  95% CI : (0.9767, 0.9791)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.119
##
```

```
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.97809
##             Specificity : 0.87879
##          Pos Pred Value : 0.99978
##          Neg Pred Value : 0.06546
##              Prevalence : 0.99826
##          Detection Rate : 0.97638
##    Detection Prevalence : 0.97659
##       Balanced Accuracy : 0.92844
##
##        'Positive' Class : 0
##
```

After running the naive bayes model, we get an accuracy of 97.79 %. But as we can see the false accuracy is very low (0.065 %). This is because the data is highly imbalanced in which a small proportion of the transactions were fraud. This will result in a biased prediction with respect to

no fraud (the majority of the data). We will discuss this more in detail later (discussion of Smote).



Overall, the ROC helps express this problem. The area under the curve (AUC) was 0.533 which is very low (maximum value of 1). With respect to the specificity, the prediction has trouble classify those incidents of fraud.Therefore this model wouldn't be as useful to credit card companies; at least until the data has been balanced.

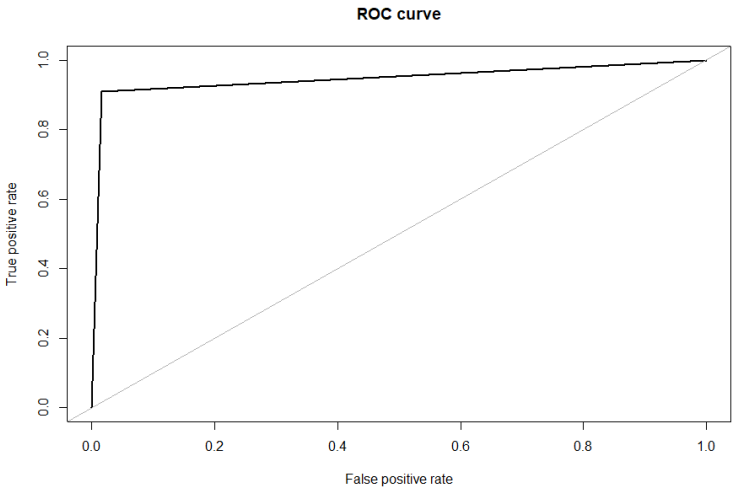
(2) Decision Tree

It is Recursive partitioning algorithm. Decision trees are built up of two types of nodes: decision nodes, and leaves.

At a leaf node we return the majority value of training data routed to the leaf node as a classification decision, or return the mean-value of outcomes as a regression estimate.

Initially we were getting accuracy around 90% after considering six key variables which has more importance are shown below, we improved our accuracy.

```
> as.data.frame(Tree_Model$variable.importance)
    Tree_Model$variable.importance
V14                       71565.92
V10                       64362.26
V11                       59806.10
V12                       59573.39
V17                       59088.14
V16                       53508.25
>
```




```
> #Model performance
> Tree_Pred <- predict(Tree_Model,test_Bal,type="class")
> confusionMatrix(Tree_Pred,test_Bal$Class)
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 16949   283
         0  2892 29038

               Accuracy : 0.9354
                 95% CI : (0.9332, 0.9376)
    No Information Rate : 0.5964
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8629

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8542
            Specificity : 0.9903
         Pos Pred Value : 0.9836
         Neg Pred Value : 0.9094
             Prevalence : 0.4036
         Detection Rate : 0.3448
   Detection Prevalence : 0.3505
      Balanced Accuracy : 0.9223

       'Positive' Class : 1

> fancyRpartPlot(Tree_Model)
> #Variable importance
> (Tree_Model$variable.importance)
     V14      V10      V11      V12      V17      V16
71565.92 64362.26 59806.10 59573.39 59088.14 53508.25
```

we got an accuracy of 93.54 % but as we look our confusion matrix, out of 19841 times the class was “1”, our model misclassified it wrong 2892 times. we need to increase the accuracy of class ‘1’.so let’s run random forest and check out it’s result. So model still needs to be improved and that's why we’ll train our model with random forest.

(3) Random Forest

In terms of the random forest, we also need to change our data become more balance in order to find the best model. In our result, it can easily be seen that the accuracy rate is 0.9935 (99.35%), this is a very high value.

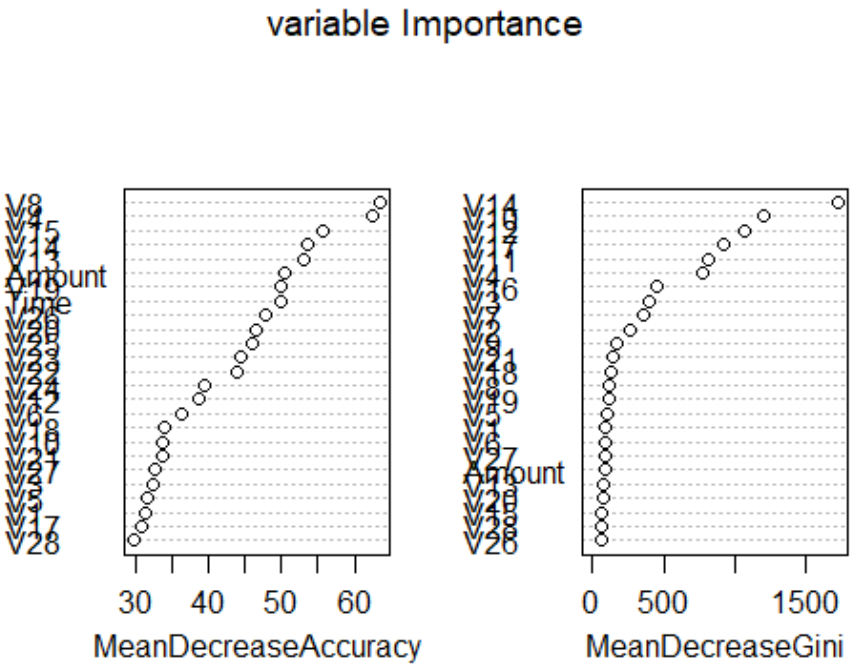
```
confusionMatrix(table(RF_Pred, test_Bal$Class))

## Confusion Matrix and Statistics
##
##
## RF_Pred    0    1
##       0 2961   29
##       1    4 2063
##
##                Accuracy : 0.9935
##                  95% CI : (0.9908, 0.9955)
##     No Information Rate : 0.5863
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9865
##
##  Mcnemar's Test P-Value : 2.943e-05
##
##             Sensitivity : 0.9987
##             Specificity : 0.9861
##          Pos Pred Value : 0.9903
##          Neg Pred Value : 0.9981
##              Prevalence : 0.5863
##          Detection Rate : 0.5855
##    Detection Prevalence : 0.5913
##       Balanced Accuracy : 0.9924
##
##        'Positive' Class : 0
##
```

In more detail, we can see the confusion matrix to know that the relationship between the reference and prediction. In the result, we can know that this model accuracy is 0.9935, it is high percentage. Comparing this model and decision tree, we can find that the accuracy is increased significantly, from 93% to 99%. In other words, this model can

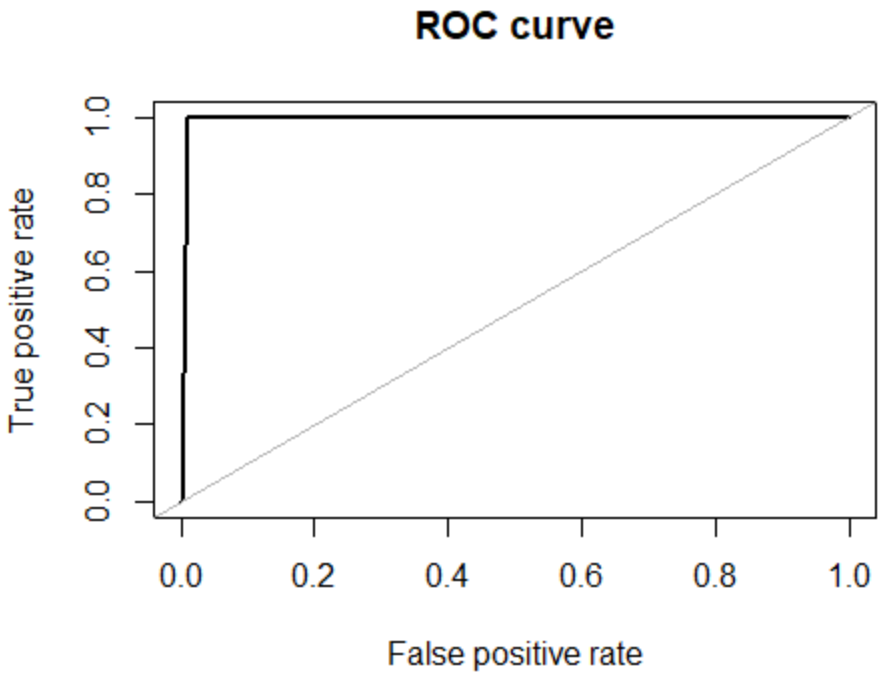
help us to predict precisely.

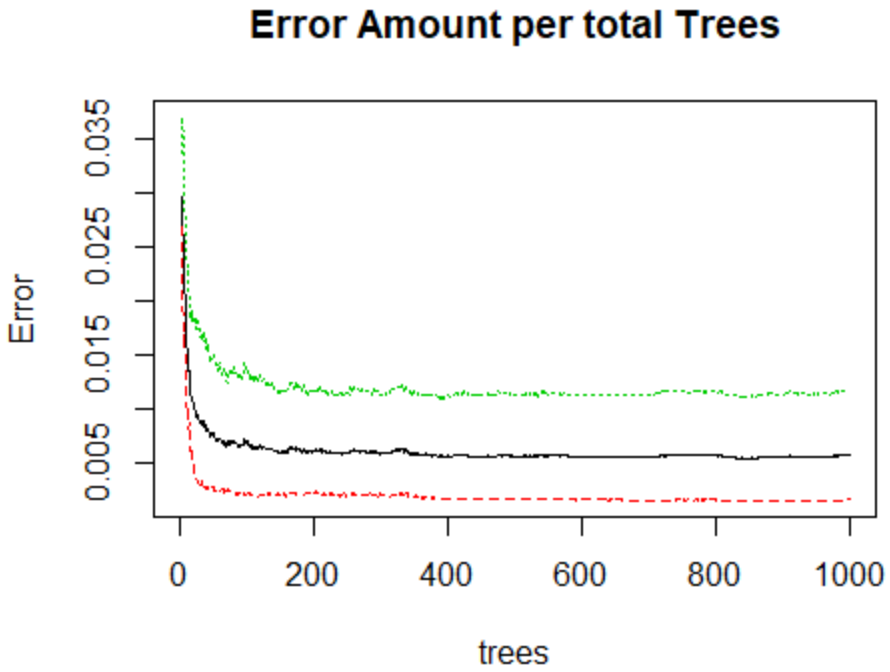
```
varImp(RF_Model)
```

```
##                0        1
## Time   38.16931 38.16931
## V1     24.99748 24.99748
## V2     21.79065 21.79065
## V3     25.16780 25.16780
## V4     47.57253 47.57253
## V5     24.92915 24.92915
## V6     28.32272 28.32272
## V7     20.21532 20.21532
## V8     45.28883 45.28883
## V9     20.99026 20.99026
## V10    24.83478 24.83478
## V11    21.40550 21.40550
## V12    26.71966 26.71966
## V13    38.09146 38.09146
## V14    40.74739 40.74739
## V15    39.14913 39.14913
## V16    15.78169 15.78169
## V17    23.04926 23.04926
## V18    20.83362 20.83362
## V19    38.22018 38.22018
## V20    36.79732 36.79732
## V21    27.31745 27.31745
## V22    31.39665 31.39665
## V23    34.89464 34.89464
## V24    26.60129 26.60129
## V25    33.67717 33.67717
## V26    36.08773 36.08773
## V27    27.19390 27.19390
## V28    25.90612 25.90612
## Amount 39.24439 39.24439
```



Moreover, in terms of VarImp, we can know that the most important value is V4 in the class number is 0 and 1. However, when we use VarImpPlot, the result will become a few differences. In the VarImpPlot, it involves two different groups of important variables, the first group is “MeanDecreaseAccuracy” and another one is “MeanDecreaseGini”. Before we discuss these two groups, we need to understand that what are differences between them. First, the mean decrease in accuracy is usually described as "the decrease in model accuracy from permuting the values in each feature". In other words, we change any one variable and give it the random forest, when the decrease is the highest, it means this variable will affect model significantly. Thus, In this case, “V8” will play an important role when we give random forest to this variable. On the other hand, the mean Decrease in Gini is the average (mean) of a variable’s total decrease in node impurity, weighted by the proportion of samples reaching that node in each individual decision tree in the random forest. This is effectively a measure of how important a variable is for estimating the value of the target variable across all of the trees that make up the forest. A higher Mean Decrease in Gini indicates higher variable importance. Variables are sorted and displayed in the Variable Importance Plot created for the Random Forest by this measure. The most important variables to the model will be highest in the plot and have the largest Mean Decrease in Gini Values, conversely, the least important variable will be the lowest in the plot, and have the smallest Mean Decrease in Gini values. As a result, we can find that “V14” is the most important variable for mean decrease gini.



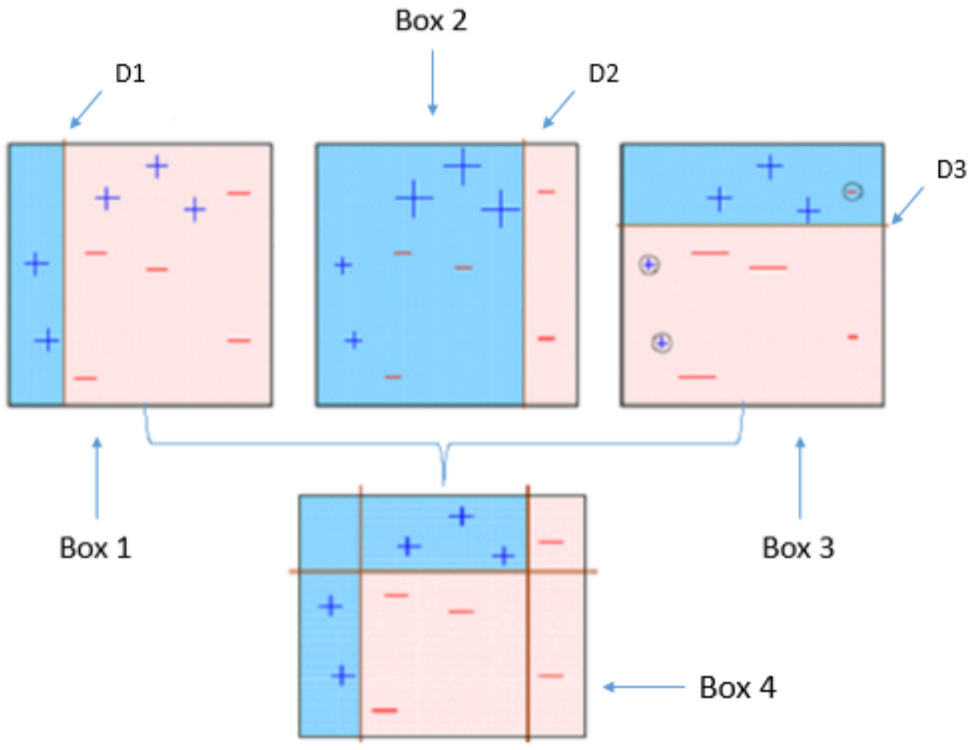


Last, we can see the ROC curve to understand the area under the curve (AUC),

when AUC is higher (0.994), it means this model is much better. After we know the ROC and AUC, we also need to check the error amount per total tree. We can clearly understand know that all of these three values are very low. Basically the variation in error becomes stable at around 400 trees. In other words, this model is very good due to the low error and high accuracy.

(4) XGBOOST:

XGBoost (Extreme Gradient Boosting) is an optimized distributed gradient boosting library. Yes, it uses gradient boosting (GBM) framework at core. Boosting is a sequential process; i.e., trees are grown using the information from a previously grown tree one after the other. This process slowly learns from data and tries to improve its prediction in subsequent



iterations.

Four classifiers (in 4 boxes), shown above, are trying hard to classify + and - classes as homogeneously as possible. Let's understand this picture well.

1. **Box 1:** The first classifier creates a vertical line (split) at D1. It says anything to the left of D1 is + and anything to the right of D1 is -. However, this classifier misclassifies three + points.
2. **Box 2:** The next classifier says don't worry I will correct your mistakes. Therefore, it gives more weight to the three + misclassified points (see bigger size of +) and creates a vertical line at D2. Again it says, anything to right of D2 is - and left is +. Still, it makes mistakes by incorrectly classifying three - points.
3. **Box 3:** The next classifier continues to bestow support. Again, it gives more weight to the three - misclassified points and creates a horizontal line at D3. Still, this classifier fails to classify the points (in circle) correctly.
4. Remember that each of these classifiers has a misclassification error associated with them.
5. Boxes 1,2, and 3 are weak classifiers. These classifiers will now be used to create a strong classifier Box 4.
6. **Box 4:** It is a weighted combination of the weak classifiers. As you can see, it does good job at classifying all the points correctly.

That's the basic idea behind boosting algorithms.

We have converted the factor variable into numeric as discussed before that XGBoost algorithm works in that way later we are converting our training and testing samples in the form of matrix.

```
parameters <- list(
  # General Parameters
  booster            = "gbtree",
   # Booster Parameters
  eta                = 0.3,
  gamma              = 0,
  max_depth          = 6,
  min_child_weight   = 1,
  subsample          = 1,
  colsample_bytree   = 1,
  objective          = "binary:logistic",
  seed               = 1234)
```

we are setting the parameters for the XGBoost model. A tree is grown one after other and attempts to reduce misclassification rate in subsequent iterations.• Then we are using ‘eta’ which

controls the learning rate, i.e., the rate at which our model learns patterns in data. After every round, it shrinks the feature weights to reach the best optimum.The value of eta is 0.3. the 3rd parameter we are setting withe value of 0 is gamma which It controls regularization. Higher the value, higher the regularization. Then max_depth parameter that controls the depth of the tree. Larger the depth, more complex the model. min_child_weight, subsample, colsample_bytree: these three parameters having value of 1 are used for: blocking the potential feature interactions to prevent overfitting. Should be tuned using CV, controlling the number of samples (observations) supplied to a tree, controlling the number of features (variables) supplied to a tree respectively. lastly objective which is used as 'binary:logistic' - logistic regression for binary classification. It returns class probabilities.

```
xgb_cv <- xgb.cv(params = parameters,
                 data = xgb_train_smote,
                 nrounds = 200,
                 nfold = 5,
                 showsd = T,
                 stratified = T,
                 print_every_n = 10,
                 early_stopping_rounds = 20,
                 maximize = F)
```

we are cross validating first to get the best round for our model.We are getting train error as almost zero in teh last round teht is hundred so we'ii be using nrounds = 158 in our model.This value can change as w run it multiple times, but it will be close to 158.

```
#training our model
xgb_model <- xgb.train(params = parameters,
                       data = xgb_train_smote,
                       nrounds = 158,
                       nthreads=1,
                       print_every_n = 10,
                       early_stopping_rounds = 10,
                       watchlist = list(train = xgb_train_smote),
                       maximize = F,
                       eval_metric = "error")
```

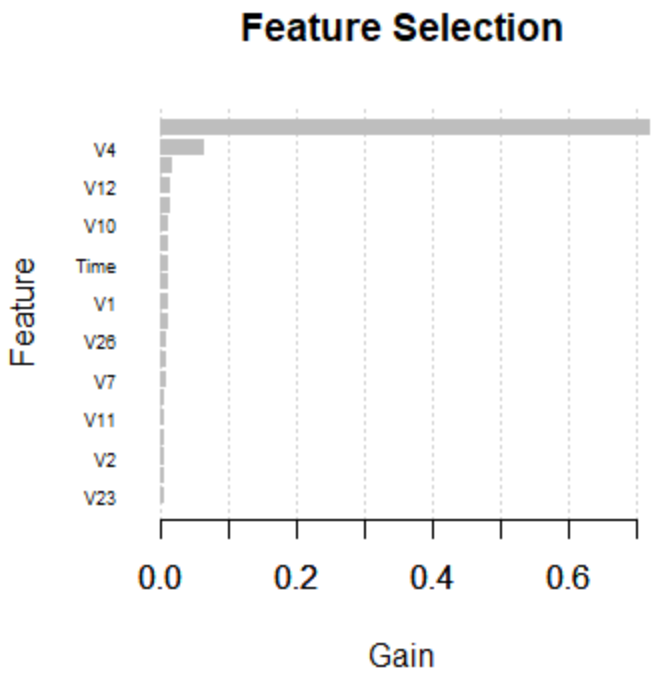
so we are training our model here we can see as the model iterates every time the train error is decreasing and finally at 82th iteration we are getting train error to be almost zero. let's go ahead and check the confusion matrix and accuracy of model testing against the test data.

```
#CM and accuracy
xgb_pred <- predict(xgb_model,xgb_test_smote)
xgb_pred <- ifelse(xgb_pred > 0.5,1,0)
confusionMatrix(factor(xgb_pred),factor(Class_test_smote))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 24585     3
##          1    15 24720
##
##                Accuracy : 0.9996
##                  95% CI : (0.9994, 0.9998)
##     No Information Rate : 0.5012
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9993
##
##  Mcnemar's Test P-Value : 0.009522
##
##             Sensitivity : 0.9994
##             Specificity : 0.9999
##          Pos Pred Value : 0.9999
##          Neg Pred Value : 0.9994
##              Prevalence : 0.4988

##          Detection Rate : 0.4984
##    Detection Prevalence : 0.4985
##       Balanced Accuracy : 0.9996
##
##        'Positive' Class : 0
```

As we can see we are getting an accuracy of 99.95 % overall and for class 1 that is the “fraud detected” , we are getting an accuracy of 99.89 %.

```
#view variable importance plot
mat <- xgb.importance (feature_names = colnames(train_smote),model =
xgb_model)
xgb.plot.importance (importance_matrix = mat[1:20],main = "Feature
Selection",xlab="Gain",ylab="Feature")
```



Here is a plot of variable importance. we can see that v14 is the most important variable which has been used to train our XGBoost model.

Conclusion: Summary of findings and recommendations

In summary, we make the table to let us easily compare these models differences.

	NB	Tree	RF	XGBOOST
Accuracy Rate	0.9779	0.9423	0.9935	0.9996
AUC	0.533	0.949	0.994	0.999

The two most important components credit card companies have to recognize is that V14 is the most important variable in detecting credit card fraud and that there are accurate algorithms that can be used to predict With high accuracy values (all over 97%), the prediction models can easily be attained. The only skeptical part involves the data balancing. In order for these models to be effective, the data based on fraud had to be over proportioned. Overall, this

means for these models to work, the creation of near data points (nearest neighbor creations under Smote) need to be accurate. This is easier said than done. For example transactions across different countries can make it much harder to create this data balancement.

The unfortunate thing is that the more fraud there is, the easier it will be for credit card companies to detect them. However the philosophy is to mitigate the amount of credit card fraud. This means that the more effective credit card companies get in decreasing the likelihood of credit card fraud, the harder it will be for them to detect credit card fraud itself. That is because they will have less samples and a prediction with even more bias towards non-fraudulent transactions. All together when you compare the different models, XGBoost appears to be the most accurate. The fact that it can incorporate multiple iterations within the model is a main reason as to why it is so accurate. This is a little more beneficial against the ideology of data imbalancement.

References

Kumar, Rishav. “Understanding Principal Component Analysis.” *Medium*, Medium, 10 Apr. 2018, medium.com/@aptrishu/understanding-principle-component-analysis-e32be0253ef0.

Machine Learning Group. “Credit Card Fraud Detection.” *Kaggle*, 23 Mar. 2018, www.kaggle.com/mlg-ulb/creditcardfraud.

Rai, Bharatendra. “Business Analytics & Data Mining, POM-681”. *University of Massachusetts: Dartmouth. Charlton College of Business.* Dartmouth, Massachusetts. Fall 2019. Lecture.

Ray, Sunil, and Business Analytics. “6 Easy Steps to Learn Naive Bayes Algorithm (with Code in Python).” *Analytics Vidhya*, 3 Sept. 2019, www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/.

Steele, Jason. “Credit Card Fraud and ID Theft Statistics.” CreditCards.com, *Red Ventures Company*, 10 June 2019, www.creditcards.com/credit-card-news/credit-card-security-id-theft-fraud-statistics-1276.php.