

Credit Card Fraud Detection

A report by

Karthik Prasad - 17MIS1020

Vamsi Krishna – 17MIS1100

Submitted to

Prof. Bharadwaja Kumar

M. Tech. Software Engineering (5-year Integrated Programme)

School of Computing Science and Engineering



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ABSTRACT

Now-a-days the usage of credit cards has dramatically increased. As credit-card becomes the most popular mode of payment for both online as well as regular purchase, cases of fraud associated with it are also rising. In online payment mode, attackers need only little information for doing fraudulent transactions (secure code, card number, expiration date etc.). It is important that credit card companies can recognize fraudulent transactions so that customers are not charged for items that they did not purchase. So, to address this issue a real time dataset is taken from Kaggle and used for doing a comparative study by building different models that predict fraudulent transactions from given transactions. Our end goal is to give a model that predicts the majority fraud cases from vast data. Usage of various **Machine Learning** algorithms such as Random Forest, Decision Tree, etc., can be used. But there are some inconsistencies that are to be taken care while classifying. One of them includes *Imbalanced Data*. Usage of **SMOTE** can overcome this issue.

I. INTRODUCTION

Our work focuses majorly on detecting fraudulent credit card transactions among the given data. End goal is to implement some Machine Learning Algorithms, Visualizations (if any) on the dataset and do the necessary classification (Binary Classification). But there are some inconsistencies that are to be taken care while classifying. Some of them include *Imbalanced Data, Hyper Parameter Tuning*.

Imbalanced classification is a supervised learning problem where one class outnumbers other classes by a large proportion. This problem is faced more frequently in binary classification problems than multi-level classification problems. Neglecting this becomes a big challenge because ML algorithms assume that the data set has balanced class distributions. ML algorithms struggle with accuracy because of the unequal distribution in dependent variable(s). This causes the performance of existing classifiers to get biased towards the majority class. The algorithms are accuracy driven i.e., aim to minimize the overall error to which the minority class contributes very little.

Our dataset contains 492 frauds out of 284,807 transactions. This is like 0.172% of all transactions. Ignoring this and split the data based on the class label (1/0) will lead the algorithm to focus mainly on majority class label (non-fraud). Sampling methods such as **SMOTE, Cost Sensitive Learning** are used to overcome this issue. This improves the minority class data (fraud) with some data (respective methodology). A good choice of hyperparameters can really make an algorithm shine. Hyper parameters must be tuned accordingly such that the accuracy of our model increases. Taking the above-mentioned topics into consideration, implementing an accurate model that classifies Fraud/Non-Fraud Transactions will be the end goal.

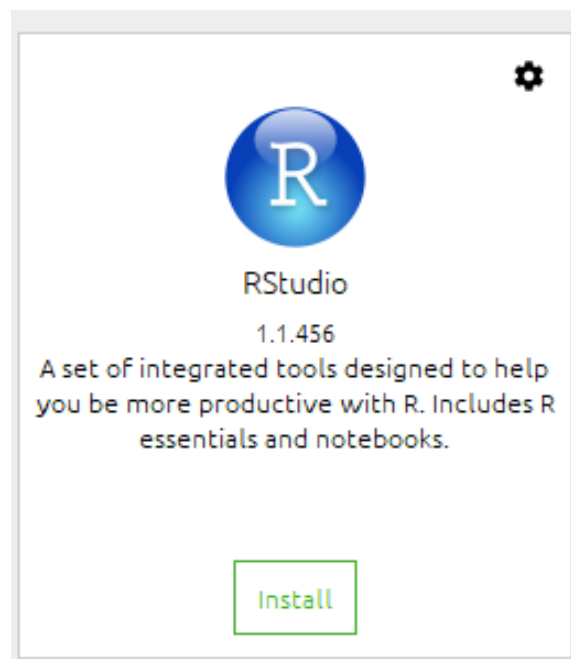
II. APPLICATIONS & TOOLS

Applications:

- It can detect fraudulent cases from a pool of transactions and can save several banks by not losing money, customers & their trust.
- We can restrict misuse of **information** with credit card fraud, information is taken from the victim and used by the fraudster without authorization from the victim.
- Credit card fraud costs consumers and the financial company billions of dollars annually, and fraudsters continuously try to find new rules and tactics to commit illegal actions. Thus, fraud detection systems have become essential for **banks** and financial institution, to minimize their losses.
- This gives an added level of **security** for users so that they can't go bankrupted.

Tools:

Most of project was developed R-Studio. As, boosting algorithms will be taking too much time for training the model, we have utilized GPU from Kaggle notebook.



III. METHODOLOGY

There are mainly 4 modules followed while doing the work:

1. **Exploratory Data Analysis**
2. **Data Cleaning**
3. **Model Building**
4. **Model Evaluation**

3.1 Exploratory Data Analysis:

Exploring the dataset to get a clear understanding on the parameters in the data and will be useful for further analysis on choosing the model. Here are some results after analysing the dataset.

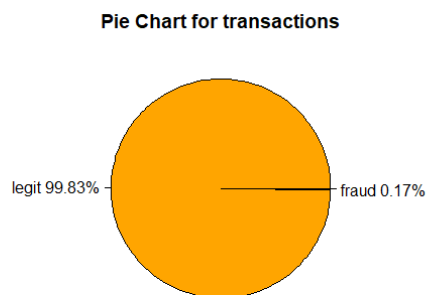


Figure 1 Pie chart showing imbalance in the data.

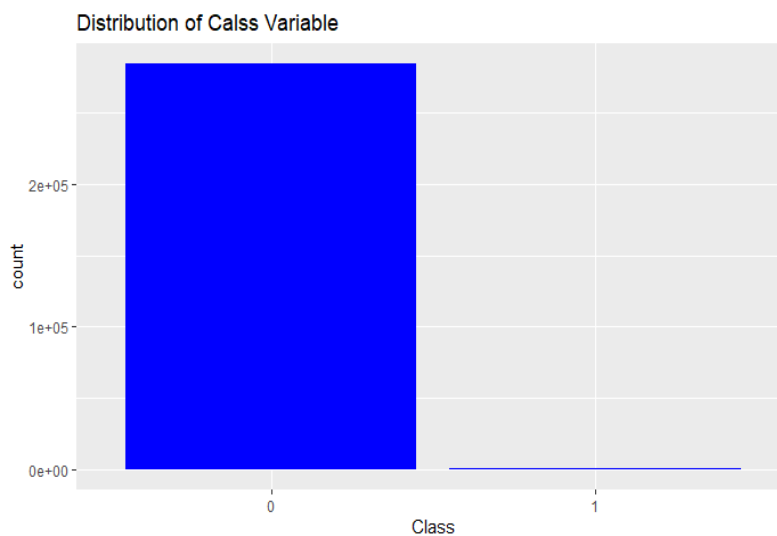


Figure 2 Bar Plot showing imbalance in the data.

By observing the below scatter plot (Figure 3), all the fraud cases are outliers, and they occur in very less amount cases.

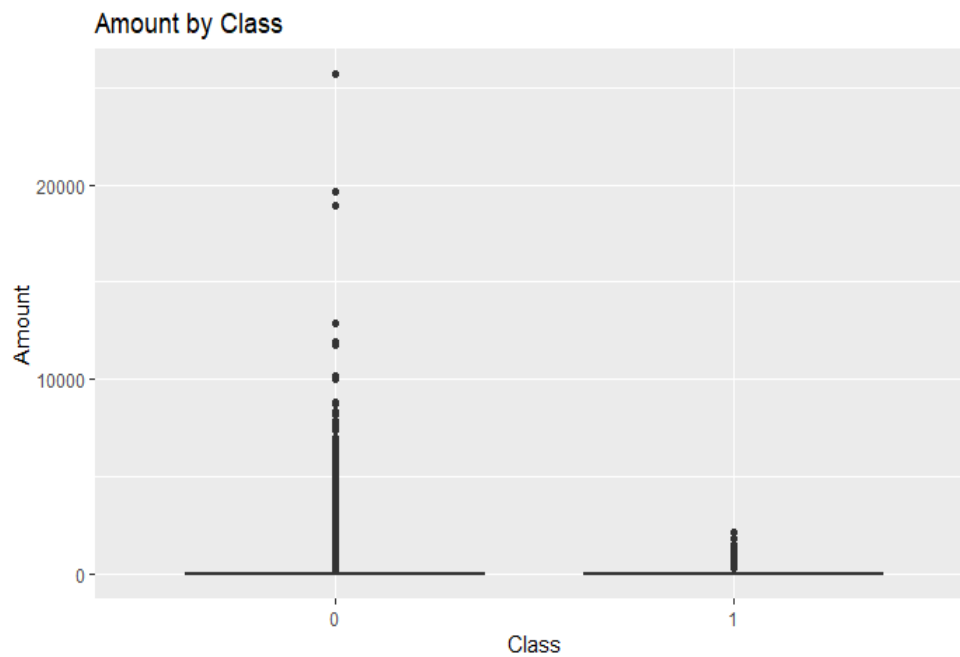


Figure 3 Scatter plot denoting fraud cases with less amount

Below plot (Figure 4) shows the ratio of Fraud and Normal cases in each feature.

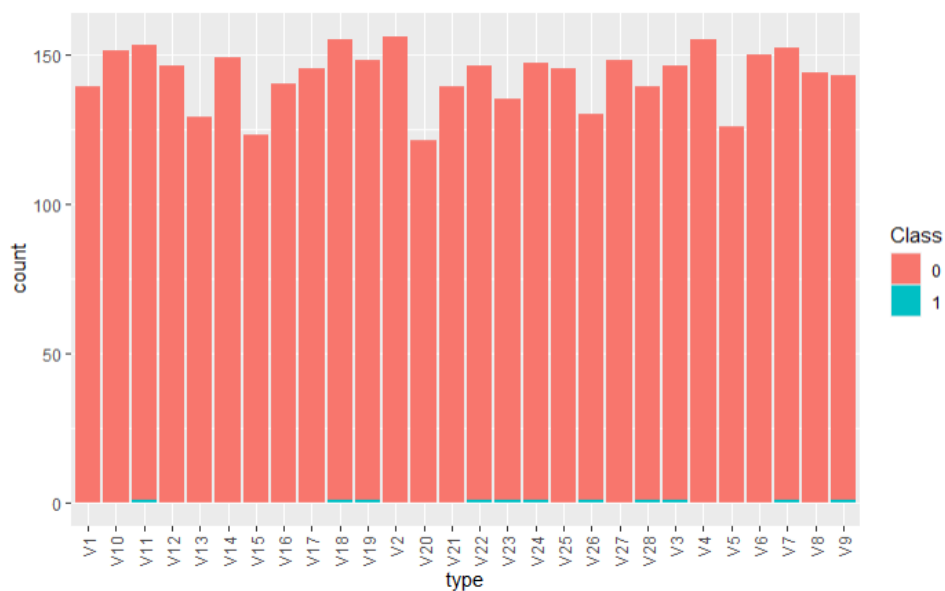


Figure 4 Plot highlights fraud cases in each feature

From the below plot (Figure 5) one can barely count the number of fraud transactions (marked in red colour).

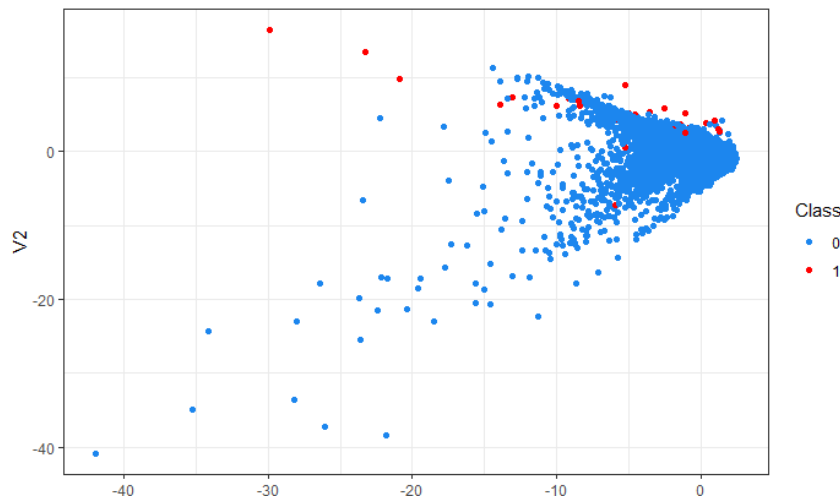


Figure 5 Comparing Legit and Fraud cases in V1 feature.

3.2 Data Cleaning (Pre-Processing):

Training any ML model on this dataset obviously will focus on legitimate (Higher number of samples). So, balancing the dataset is a huge concern.

This can be done using 4 methods, they are: *Random Over Sampling (ROS)*, *Random Under Sampling (RUS)*, *Both ROS and RUS*, *SMOTE (Synthetic Minority Over-Sampling Technique)*.

3.2.1 Random Over Sampling (ROS): ROS randomly increases the minority cases by duplicating them. Below is the comparison of V1 feature before and after applying ROS.

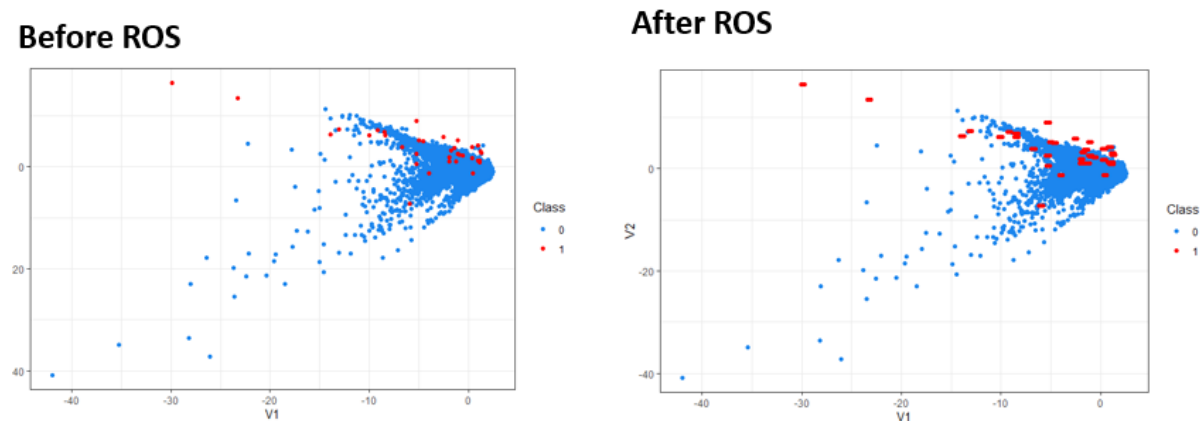


Figure 6 ROS comparison using V1 feature.

But the problem with ROS is that it allocates new data from the existing data. This produces *new points on top of the old ones*. **jitter** is used in ggplot to show the difference.

3.2.2 Random Under Sampling (RUS): RUS randomly decreases the majority cases by deleting them. Below is the comparison of V1 feature before and after applying RUS.

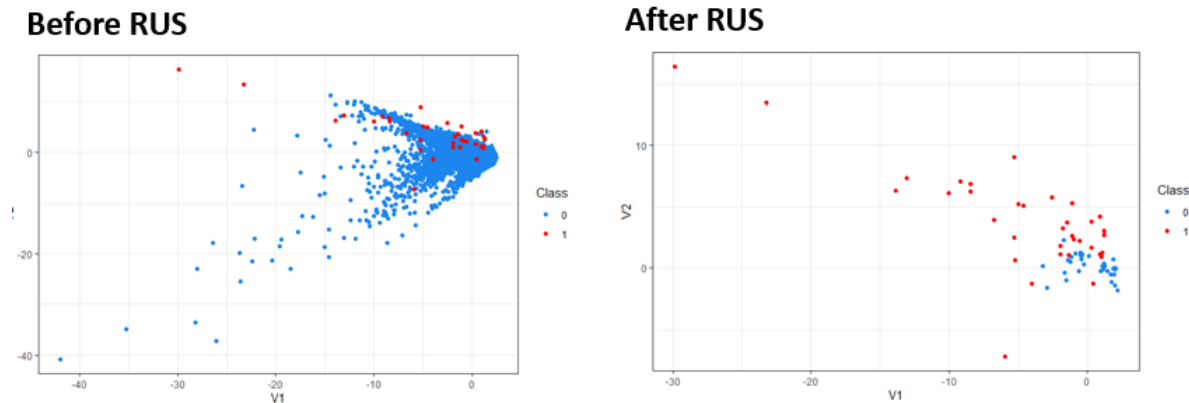


Figure 7 RUS comparison using V1 feature.

After applying RUS, majority class samples have decreased to 35. This affects the training process hugely.

3.2.3 Both (RUS & ROS): This follows performing both ROS and RUS. Below is the comparison of V1 feature before and after applying Both ROS & RUS.

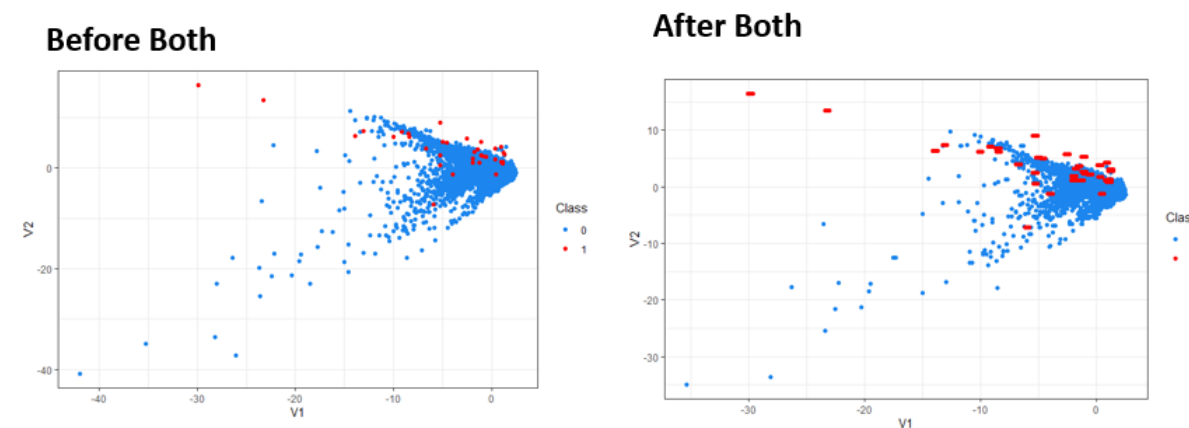


Figure 8 RUS&ROS comparison using V1 feature.

Though this decreases majority class samples, this also produces new points on top of the old ones. *jitter* is used in ggplot to show the difference.

3.2.4 SMOTE

Regarding synthetic data generation, synthetic minority oversampling technique (SMOTE) is a powerful and widely used method. SMOTE algorithm creates artificial data based on feature space (rather than data space) similarities from minority samples. It generates a random set of minority class observations to shift the classifier learning bias towards minority class.

To generate artificial data, it uses bootstrapping and k-nearest neighbours. Precisely, it works this way:

- Take the difference between the feature vector (sample) under consideration and its nearest neighbour.
- Multiply this difference by a random number between 0 and 1.
- Add it to the feature vector under consideration.
- This causes the selection of a random point along the line segment between two specific features.

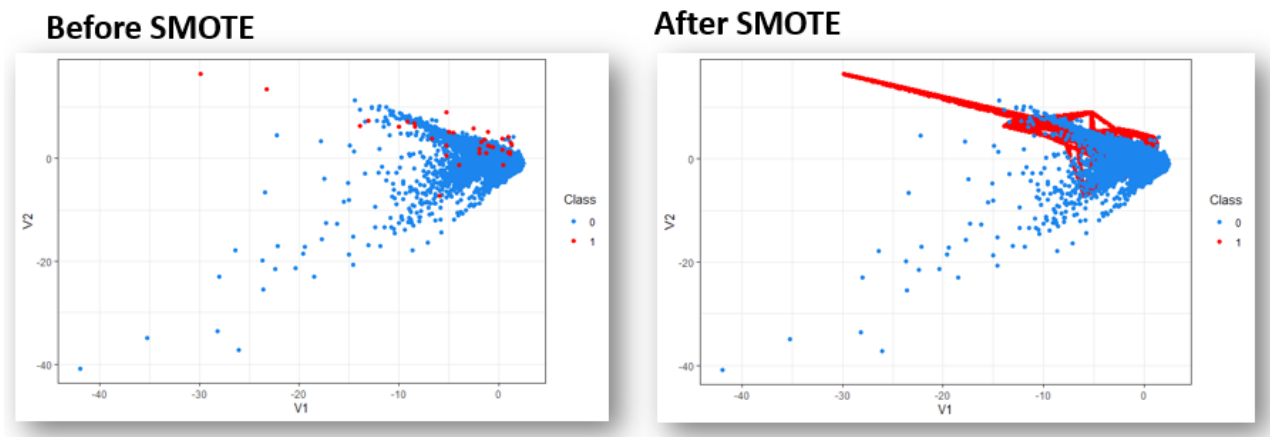


Figure 9 SMOTE comparison using V1 feature.

Unlike The above 3 balancing methods, SMOTE doesn't randomly populate minority samples. It used KNN to do so. This effectively overcame the case of unbalanced dataset.

It's time to use this data and apply different algorithms. The proposed work consists of multiple ML and their comparison with SMOTE data and Unbalanced data.

3.3 Model Building

In the proposed work, implementation of Decision tree and Random Forest models for the dataset have been completed, they are (Considered only 40% of data while building the model due to memory issues in R).

3.3.1 Train/ Test Split: 80% of the data for train set (91139) and the rest or 20% for test set (22784 rows) was partitioned from the dataset.

3.3.2 Building Decision Tree: Decision tree learning is a supervised machine learning technique for inducing a decision tree from training data. A decision tree (also referred to as a

classification tree or a reduction tree) is a predictive model which is a mapping from observations about an item to conclusions about its target value.

Trained on SMOTE data and tested on test data(non-smote):

		Reference	
Prediction		0	1
	0	22284	5
	1	460	35

Figure 10 Confusion Matrix for DT trained on SMOTE data and tested on test data.

Trained on Non-Smote data and tested on test data(non-smote):

		Reference	
Prediction		0	1
	0	22740	13
	1	4	27

Figure 11 Confusion Matrix for DT trained on Non-Smote data and tested on test data.

Trained on SMOTE data and tested on whole dataset:

		Reference	
Prediction		0	1
	0	111302	13
	1	2420	188

Figure 12 Confusion Matrix for DT trained on Smote data and tested on whole data.

Trained on Non-Smote data and tested on whole dataset:

		Reference	
Prediction		0	1
	0	113706	45
	1	16	156

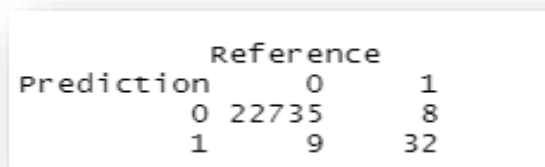
Figure 13 Confusion Matrix for DT trained on non-Smote data and tested on whole data.

By checking the Specificity scores, it is evident that applying SMOTE is helping the classifier.

3.3.3 Building Random Forest:

Random forest is a classification algorithm consisting of many decision trees and bagging, feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Trained with SMOTE data, tested with test data: Taking only 50% of the train data regarding memory issues and then built random forest model with 1000 trees on the smote dataset.



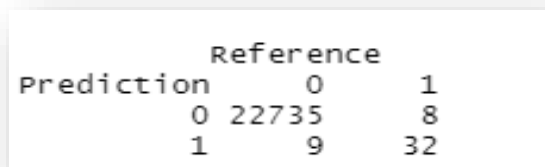
		Reference	
Prediction		0	1
0	22735	8	
1	9	32	

Figure 14 Confusion Matrix for RF trained on Smote data and tested on test data.

Out of bag error:

Out of bag error is observed while testing the RF model on multiple trees. In the proposed work testing is done on various number of trees (often referred as Hyperparameter tuning).

Test1: For this first test only 500 trees were taken to check for out of bag error. Then the model was able to predict 32 frauds out of 40 which was same with 1000 trees.



		Reference	
Prediction		0	1
0	22735	8	
1	9	32	

Figure 15 Confusion Matrix for RF trained on Smote data and tested on test data with 500 trees.

Test2: This time only proceeded with 200 trees; the performance was improved a bit.

		Reference	
Prediction		0	1
	0	22735	7
	1	9	33

Figure 16 Confusion Matrix for RF trained on Smote data and tested on test data with 200 trees.

Test3: Tuned with 400 trees.

		Reference	
Prediction		0	1
	0	22735	8
	1	9	32

Figure 17 Confusion Matrix for RF trained on Smote data and tested on test data with 400 trees.

It is observed that there is no improvement considering 400,500,1000 trees. So, it is suggestable to consider 200 trees for further training to obtain much better results.

Trained with Smote data, tested with whole data:

		Reference	
Prediction		0	1
	0	113694	8
	1	28	193

Figure 18 Confusion Matrix for RF trained on Smote data and tested on whole data.

Variable Importance plot:

The default method to compute variable importance is the *mean decrease in impurity* (or *gini importance*) mechanism: At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable and is accumulated over all the trees in the forest separately for each variable.

Observing top 10 important features from Random Forest. Below is the comparison of how the model will perform on these features alone.

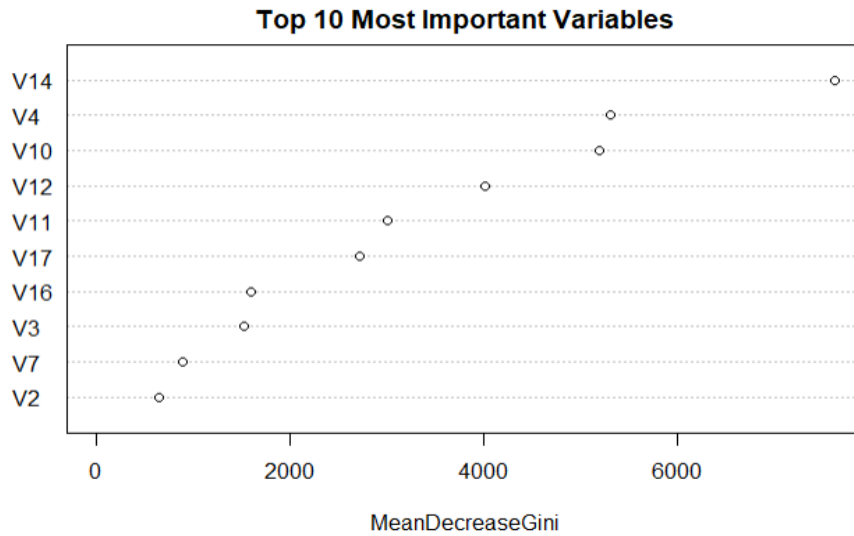


Figure 19 Variable importance plot for RF model

Trained with Smote data only with important features, tested with test data:

Prediction	Reference	
	0	1
0	22696	7
1	48	33

Figure 20 Confusion Matrix for RF trained on Smote data only important features and tested on test data.

Trained with Smote data only with important features, tested with whole data:

Prediction	Reference	
	0	1
0	113565	8
1	157	193

Figure 21 Confusion Matrix for RF trained on Smote data only important features and tested on whole data.

Inference:

This is the same result that was observed while building the model with all the features (200 trees). Though there is no difference in performance of two models (one with important features and other with all the features), there is a difference in build time (execution time) of the models. Model with important features was trained faster and is optimized.

Now we have extended our research to apply advanced algorithms such as **XGBoost**, **LightGBM**. We have normalized whole dataset to a common range [0-1].

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0	0.000000e+00	0.9351923	0.7664904	0.8813649	0.3130227	0.7634387	0.2676686	0.2668152	0.7864442	...	0.5829422	0.5611844	0.5229921
0	5.787305e-06	0.9352170	0.7531177	0.8681408	0.2687655	0.7623288	0.2811221	0.2701772	0.7880423	...	0.5858550	0.5654773	0.5460298
0	1.157461e-05	0.9510571	0.7773933	0.8571874	0.2444717	0.7685504	0.2627209	0.2682566	0.7881778	...	0.5811700	0.5581224	0.4839152
0	2.314922e-05	0.9791841	0.7687462	0.8381998	0.3052410	0.7670080	0.2657616	0.2653241	0.7862566	...	0.5779268	0.5587758	0.4974024
0	4.051113e-05	0.9431010	0.7702777	0.8354522	0.2398937	0.7836880	0.3004392	0.2676105	0.7945149	...	0.5808268	0.5602959	0.4975245
0	5.208574e-05	0.9525471	0.7790717	0.8555110	0.2420808	0.7690780	0.2605388	0.2693250	0.7861314	...	0.5824343	0.5574992	0.4804664

Figure 22 Normalized Dataset.

3.3.4 XGBoost: Applied XGBoost on whole smote data and achieved maximum precision & recall but it took about 2hrs to train the whole benchmark. While doing so, we have used 3-fold Cross validation. It helps the model to tune and achieve better accuracy.

```
xgb_train <- sparse.model.matrix(Class ~ . -1, data = train)
xgb_test  <- sparse.model.matrix(Class ~ . -1, data = test)

ctrl_xgb <- trainControl(method = "cv",
  number = 3, # 3-fold cross-validation
  summaryFunction=prSummary, # area under precision-recall curve
  classProbs=TRUE,
  allowParallel = TRUE)
```

```
tic()
set.seed(23)
xgb_train_benchmark <- train(x = xgb_train,
  y = train$Class,
  method = "xgbTree",
  metric = "AUC",
  trControl = ctrl_xgb)
toc()
```

Warning message in train.default(x = xgb_train, y = train\$Class, method = "xgbTree", :
"The training data could not be converted to a data frame for saving"

5433.579 sec elapsed

After using 3-fold cross validation on SMOTE data and applied XGBoost, we got area under Precision-Recall "0.99989".

```
paste("Area under the Precision-Recall curve:", round(xgb_trainauprc$auc.integral, 7))
```

'Area under the Precision-Recall curve: 0.9998952'

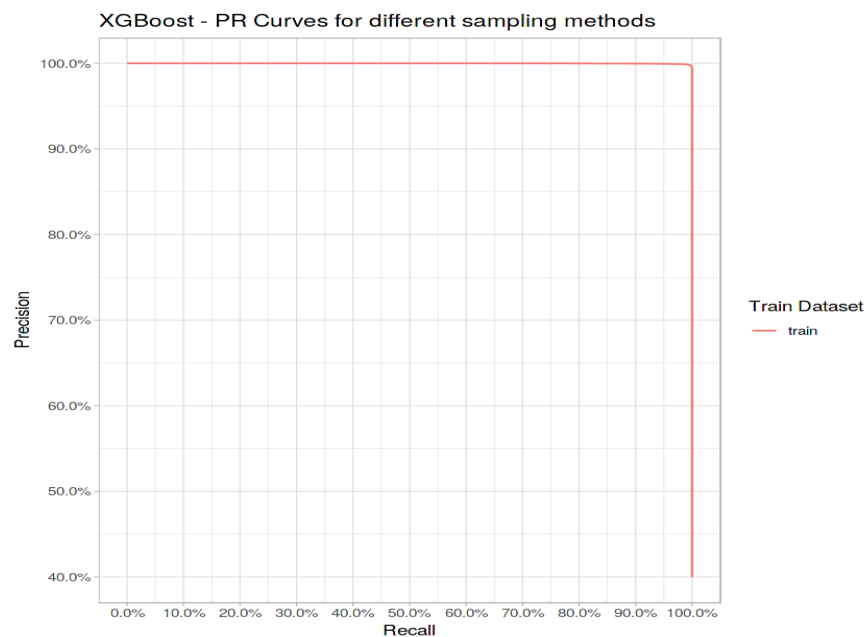


Figure 23 PR Curve for XGBoost with SMOTE data

3.3.5 LightGBM vs XGBoost: Applied both algorithms on unbalanced data and inferred that both are producing 0.97 ROC value. (lightGBM has 0.03 more). and after applying smote there isn't much difference in ROC values. But the training time differed. ***XGBoost is consuming more time than lightGBM and producing same result***. By this we can say that lightGBM is preferred over XGBoost in this case.

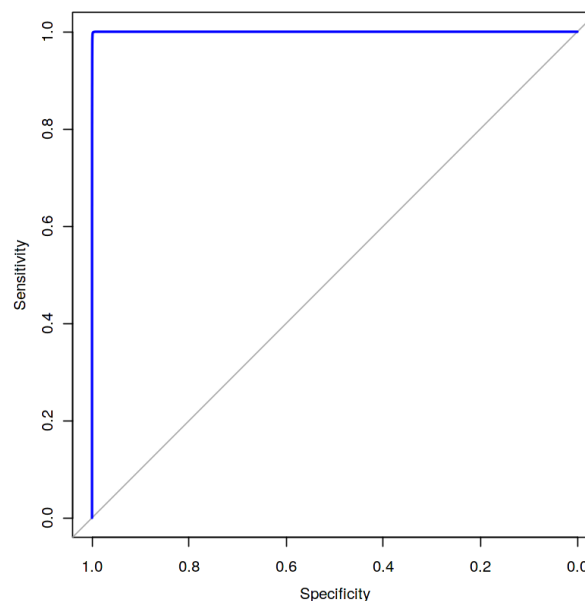


Figure 24 ROC for XGBoost

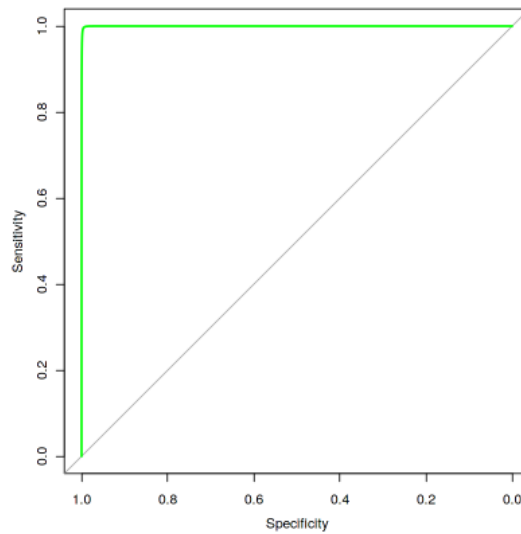
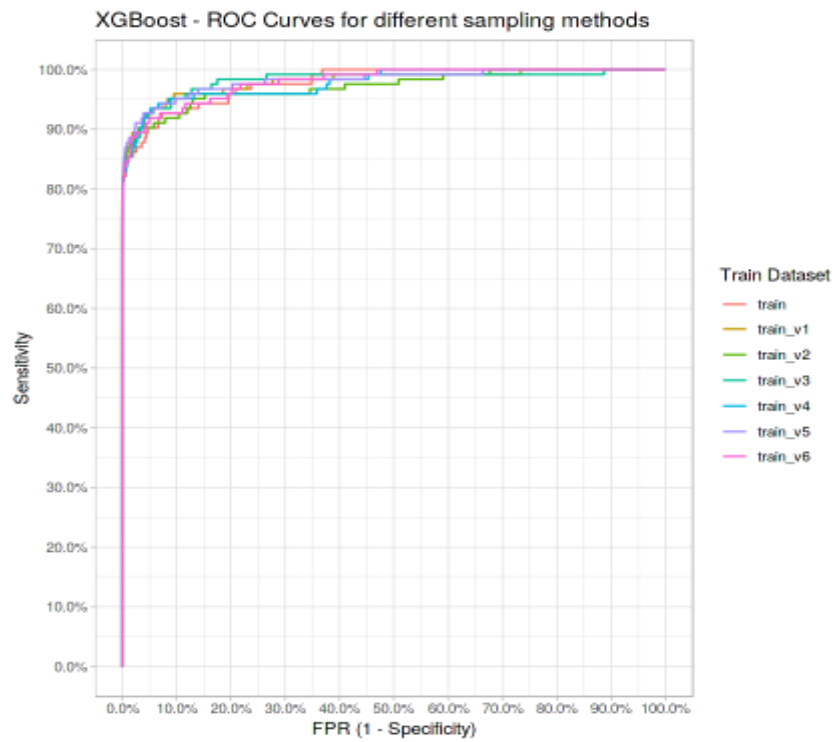


Figure 25 ROC for LightGBM

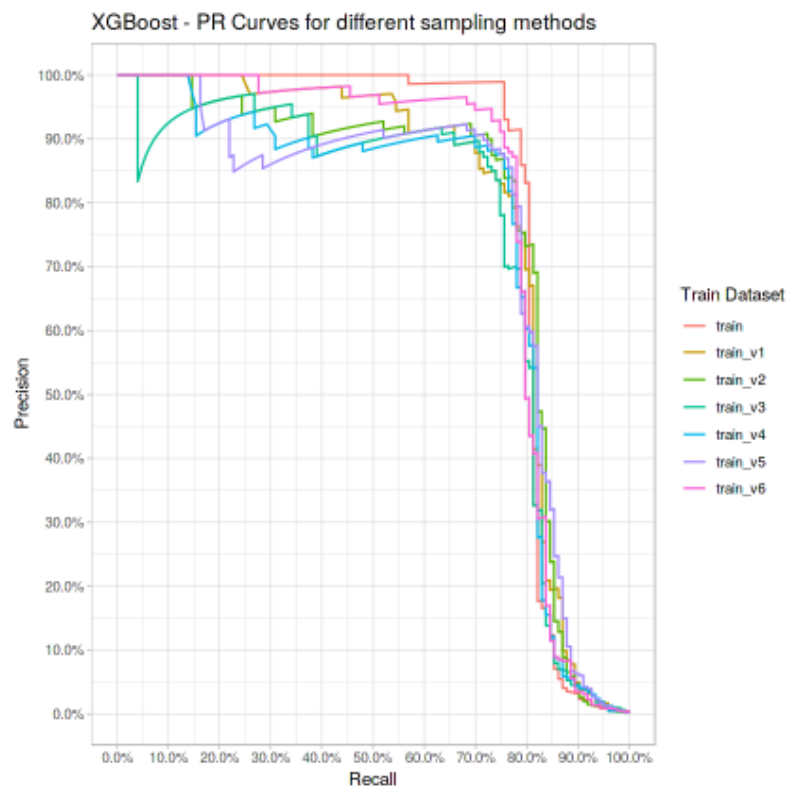
After this we specifically used XGBoost. We split the data into 6 parts and compared the PR & ROC curves. But it is clearly observed that we are not able to compare based only on ROC curve. This is the reason we have used PR curve.

name	num_obs	frauds	frauds_perc	weighting	smote_amt
<fct>	<dbl>	<dbl>	<dbl>	<fct>	<fct>
train	213606	369	0.00172748	original (very imbalanced)	none
train_v1	3690	1845	0.50000000	balanced	some
train_v2	22140	11070	0.50000000	balanced	lots
train_v3	2460	1845	0.75000000	mostly fraud	some
train_v4	14760	11070	0.75000000	mostly fraud	lots
train_v5	7380	1845	0.25000000	mostly non-fraud	some
train_v6	44280	11070	0.25000000	mostly non-fraud	lots

ROC Curve:



PR Curve:



By seeing the PR curve, we can say that **train_v6** has higher PR values. Though the graph shows **train** has higher value it is an unbalanced dataset. The balanced datasets and those that maintained the fraud minority seemed to perform better than those where the class imbalance was reversed (to a fraud majority)

```
In [89]: train_datasets_model_summary %>%
  group_by(weighting) %>%
  summarize(avg_AUPRC = mean(AUPRC)) %>%
  arrange(desc(avg_AUPRC))

`summarise()` ungrouping output (override with `.groups` argument)
```

A tibble: 4 × 2

weighting	avg_AUPRC
<fct>	<dbl>
original (very imbalanced)	0.8132780
balanced	0.7836056
mostly non-fraud	0.7790395
mostly fraud	0.7482153

```
In [90]: train_datasets_model_summary %>%
  group_by(smote_amt) %>%
  summarize(avg_AUPRC = mean(AUPRC)) %>%
  arrange(desc(avg_AUPRC))

`summarise()` ungrouping output (override with `.groups` argument)
```

A tibble: 3 × 2

smote_amt	avg_AUPRC
<fct>	<dbl>
none	0.8132780
lots	0.7747120
some	0.7658616

By the table we can say that balanced datasets (SMOTE) **train_v5** & **train_v6** are performing better than others. Though the results are accurate and efficient Both the boosting algorithms took huge time while training.

IV. MODEL EVALUATION

Below is the comparison of various metric measures for Random Forest and Decision tree models (Boosting algorithms are compared based on their ROC & PR Curves).

Model	Precision	Recall	Specificity	F-Score
Decision Tree – Smote Data, Test data	0.9997	0.9797	0.875	0.9896
Decision Tree – Non-Smote Data, Test data	0.9994	0.9998	0.675	0.9996
Decision Tree – Smote Data, Whole data	0.9998	0.97872	0.935	0.9891
Decision Tree – Non- Smote Data, Whole data	0.9996	0.9998	0.776	0.9997
Random Forest – SMOTE – 1000 trees – Test data	0.9996	0.9996	0.80	0.99962
Random Forest – SMOTE – 500 trees – Test data	0.9996	0.9996	0.80	0.99962
Random Forest – SMOTE – 200 trees – Test data	0.9997	0.9996	0.825	0.99964
Random Forest – SMOTE – 400 trees – Test data	0.9996	0.9996	0.80	0.99962
Random Forest – SMOTE – 200 trees – Whole data	0.9999	0.9997	0.96	0.9998
Random Forest – SMOTE – 200 trees – Imp Features - Test data	0.9997	0.9996	0.825	0.99964
Random Forest – SMOTE – 200 trees – Imp Features - Whole data	0.9999	0.9997	0.96	0.9998

Table 1 Metric comparison for ML models

V. CONCLUSION

From the comparison it is evident that Random Forest with important features is very efficient at detecting Fraudulent cases. This work was helpful in exploring many new techniques such as SMOTE, Boosting Algorithms and visualisations such as PR, ROC curves for new algorithms and comparing them based on metrics like precision, recall.

As *lightGBM* and *XGBoost* are gradient boosting algorithms, they took more time than others. And this was reflected in their efficiency. *Kaggle* was helpful for training these models as it provides free GPU. *The construction of multitude tree-based models has more significant accuracy than that of linear or polynomial ML techniques.*

Fraud detection systems have become essential for banks and financial institutions, to minimize their losses. Detecting fraudulent transactions in credit cards gives huge relief for banks as their trust among customers is reserved. So, these algorithms will help them in rectifying losses and by that they can retain customers' trust.

We conclude by saying that **Random Forest**, **XGBoost** with cross validation are performing well on SMOTE data and produced highest PR values (both 0.9989) followed by **lightGBM** (it took less time to train compared to XGBoost and produced good results)

REFERENCES

1. <https://www.rdocumentation.org/>
2. <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>
3. <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>
4. <https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html>