Analysis and enhancement of Credit Card Fraud detection using Pycaret and Random Forest Classifier

Table of Content

List of Figures

List of Tables

1 Introduction

1.1 Overview

The Credit Card Fraud detection problem comprises sporting past credit card transactions with the knowledge of the ones that turned out to be a fraud. The Credit Card Fraud detection problem comprises sporting past credit card transactions with the knowledge of the ones that turned out to be a fraud. Our purpose here is to detect extensively fraudulent transactions as possible while minimizing the incorrect fraud classifications. Machine Learning is a technique to detect criminal patterns in transactions. It makes no doubt that the integration of machine learning techniques in the payment of card fraud detection systems has improved their ability efficiently to detect frauds. The model is a parametric function that allows predicting the probability of a transaction to be a scam or not. Many machine-learning algorithms predict the scale of the input. The values of time and amount are highly differing, scaling is done to bring all features to the same level of magnitudes. Machine learning qualifies for developing algorithms that process big datasets with multiple variables and support finding invisible

1.4 Research questions

What are the various feature selection methods and would that effect the precision of the model when applied to the present dataset?

What are advantages of pycaret and can that be used in the present study to gain the aim?

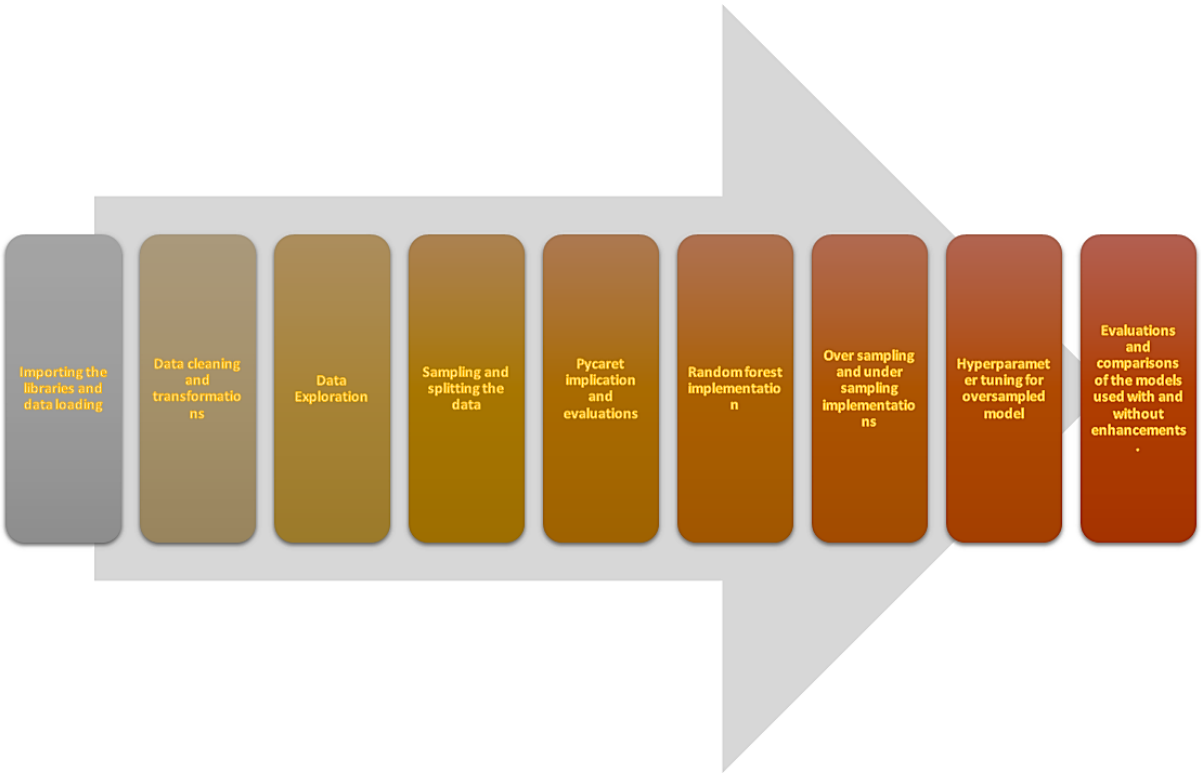
2 Literature review

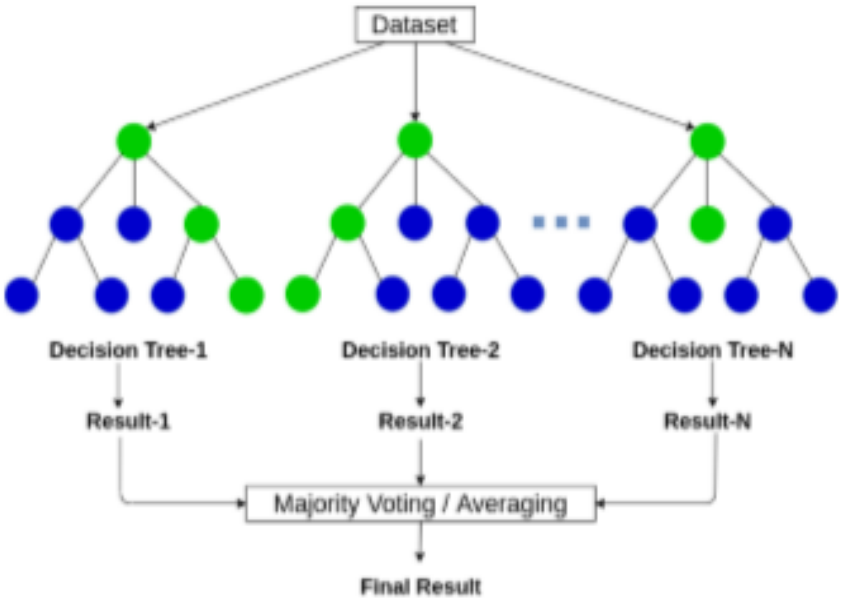
2.1 Overview

In this chapter, Section 2.2 provides a brief introduction to the PyCaret library, Section 2.3 discusses the different hyperparameter tuning techniques that are used in machine learning (ML). Section 2.4 is based on recent studies collected and authors point of view, in Section 2.5 a brief introduction is provided related to the K-fold and Random Search techniques. In Section 2.6 discuss about the related studies on detecting the frauds for the credit card transactions by using ML techniques, some of the contributions of works are discussed. and in Section 2.7 a brief summary related to this chapter is provided.

2.2 Introduction to Pycaret library

In python Pycaret is a low-code ML library that helps in automating the ML workflows and is of high level. This is an open-source library where time taken for coding is less and time taken for the analysis is more says Mulpuru and Mishra (2021). Python library helps to relate easily, train, estimate, tune, and to use ML models with fewer code lines. It is a tool for managing the model that increase the experiment cycle exponentially (Raschka, 2015). And thus makes the experiments that are performed very much fast and also efficient. It is assumed to be a big wrapper for other libraries and the frameworks in data science like Spacy, SHAP, Optuna, Yellowbrick and Scikit-learn, XGBoost, Hyperopt, Ray, CatBoost, LightGBM and much. Hence, there are many advantages of the library.


Importing the libraries and data loading
Data cleaning and transformations
Data Exploration
Sampling and splitting the data
Pycaret implication and evaluations
Random forest implementation
Over sampling and under sampling implementations
Hyperparameter tuning for oversampled model
Evaluations and comparisons of the models used with and without enhancements.


Dataset
Decision Tree-1
Decision Tree-2
Decision Tree-N
Result-1
Result-2
Result-N
Majority Voting / Averaging
Final Result

3.4 Features and advantages of Random Forest Classifier

Machine learning techniques random forest as it utilizes the bagging technique and is the efficient version of the decision tree is considered to be the highest accurate model when working with big data like this one (Rokach, 2010).

This type of algorithms ais considered for both classification as well as regression tasks and it can be considered efficient because it is based on the conditions and root note and also the combination of the decision trees in the model is also based on the bagging technique.

Type of algorithm works better and provides efficient results when a large amount of data is provided. This is because the random forest consists of a parallel setup of different trees which are based on the voting mechanism. When such a hierarchy of an algorithm is present inside an algorithm it is very important that it is filled with a huge amount of data. Basically, it can also be provided with a medium amount of data (Rokach, 2010).

When the type of algorithm consists of decision trees it is normal to find a lot of Trees and branches together which can be constituted and called as a forest. This forest bases its classification prediction output based on a working mechanism which means the tree with the highest working mechanism or the highest prediction is said to be the output.

It works better when its parameters are tuned. However, in this study the Random Forest model is recognized as best model by the pycaret library. For building individual decision trees, a random forest approach can focus on both observations and variables from training data, and gain full voting for categorization and the overall average for regression issues, respectively. It also employs a bagging strategy, which selects all columns incapable of expressing relevant

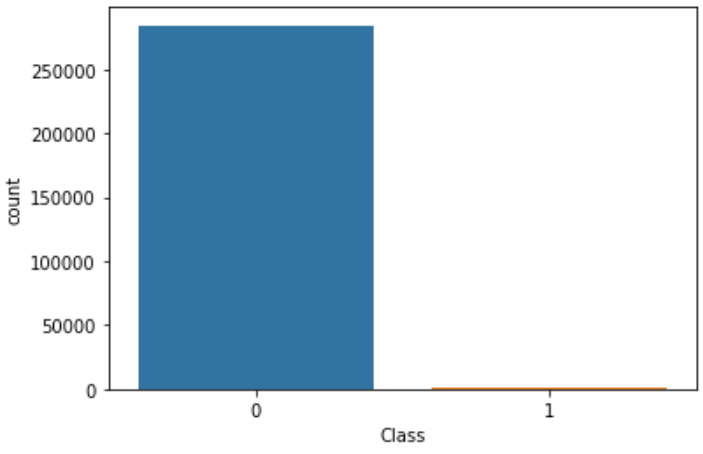
variables at the root of all decision trees based on a random sample of data. Financial assessment necessitates a significant amount of time and effort because to the great potential for profit or loss. Among the most often used studies in the banking industry is customer analysis. Random forest may be a good solution for problems like predicting a customer's loan default or identifying any fraudulent activity (Rokach, 2010).

4 Findings

4.1 Dataset and features

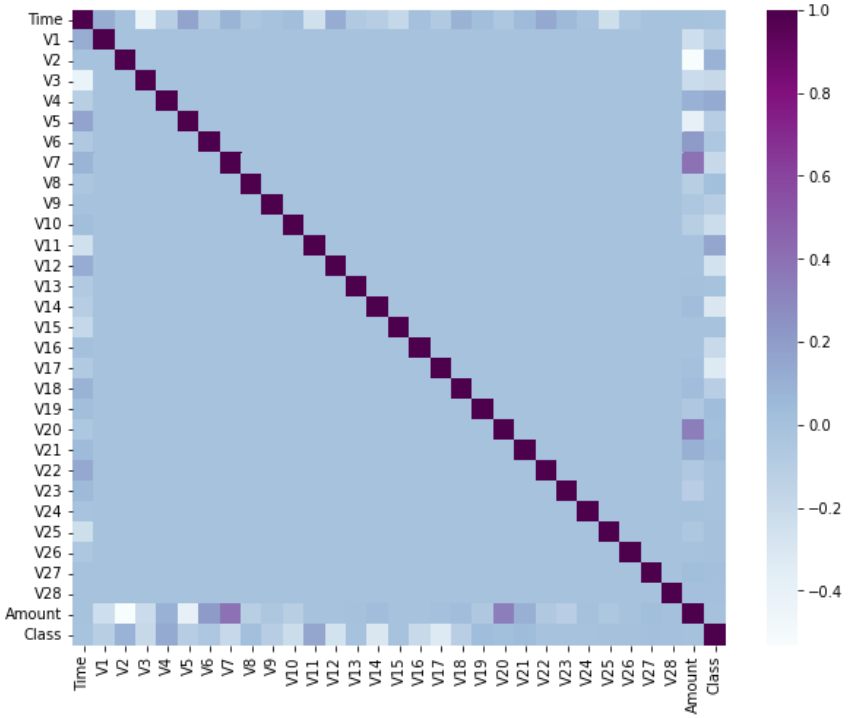
As this is the study based on the machine learning implementations, dataset is required as a fuel to the study. Here in this study, dataset in the format of Comma separated value (CSV) is used and implemented. These types of formats are very easy to be used as these are plain text files with commas. These are efficient when considering to organize the data like this in the study. The dataset used here consist of 284807 instances and 31 columns including the dependent variable which is the “Class” column as seen in the Fig. 4.1.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```



```
#Importing the required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
!pip install pycaret
import time

from collections import Counter
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import RandomizedSearchCV,train_test_split,l
earning_curve
from imblearn.under_sampling import ClusterCentroids
from imblearn.combine import SMOTETomek
from sklearn.metrics import confusion_matrix,classification_report,plot_c
onfusion_matrix,accuracy_score
from sklearn.metrics import roc_curve,auc,roc_auc_score
```

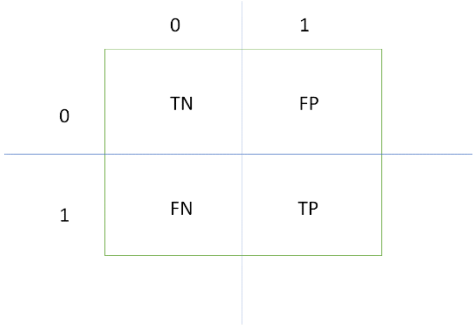

Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

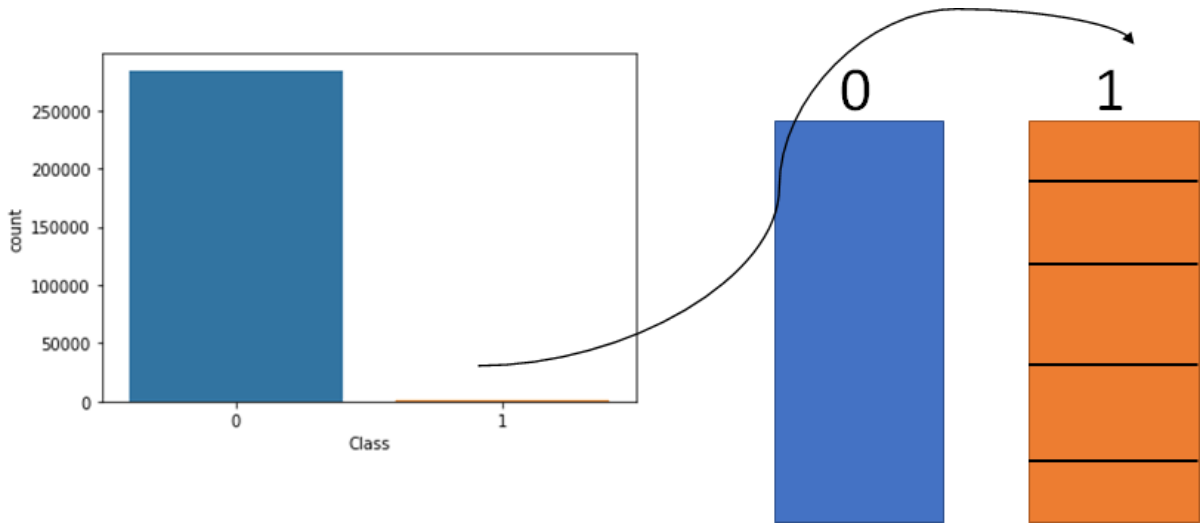
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9992	0.9316	0.6667	0.9250	0.7514	0.7511	0.7728	4.213
ada	Ada Boost Classifier	0.9992	0.9777	0.6833	0.8800	0.7503	0.7499	0.7653	3.230
lda	Linear Discriminant Analysis	0.9992	0.8957	0.6417	0.9250	0.7248	0.7244	0.7520	0.122
et	Extra Trees Classifier	0.9991	0.9476	0.6167	0.9250	0.7105	0.7101	0.7386	0.976
lr	Logistic Regression	0.9989	0.9855	0.6417	0.8333	0.6762	0.6757	0.7047	0.810
ridge	Ridge Classifier	0.9989	0.0000	0.5083	0.9250	0.6081	0.6077	0.6574	0.040
qda	Quadratic Discriminant Analysis	0.9989	0.8227	0.6000	0.7217	0.6377	0.6373	0.6483	0.062
gbc	Gradient Boosting Classifier	0.9986	0.9384	0.6917	0.6533	0.6544	0.6537	0.6626	15.902
dt	Decision Tree Classifier	0.9983	0.7829	0.5667	0.6055	0.5353	0.5346	0.5584	0.403
knn	K Neighbors Classifier	0.9981	0.5968	0.0000	0.0000	0.0000	0.0000	0.0000	0.243
svm	SVM - Linear Kernel	0.9981	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.212
dummy	Dummy Classifier	0.9981	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.031
lightgbm	Light Gradient Boosting Machine	0.9968	0.7573	0.5083	0.4833	0.4396	0.4384	0.4635	0.660
nb	Naive Bayes	0.9944	0.9863	0.6417	0.2157	0.3050	0.3030	0.3579	0.041

```
[[198999     28]
 [    77    261]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    199027
           1       0.90      0.77      0.83       338

    accuracy                           1.00    199365
   macro avg       0.95      0.89      0.92    199365
weighted avg       1.00      1.00      1.00    199365
```




250000
200000
150000
100000
50000
0
count
0
1
Class
0
1

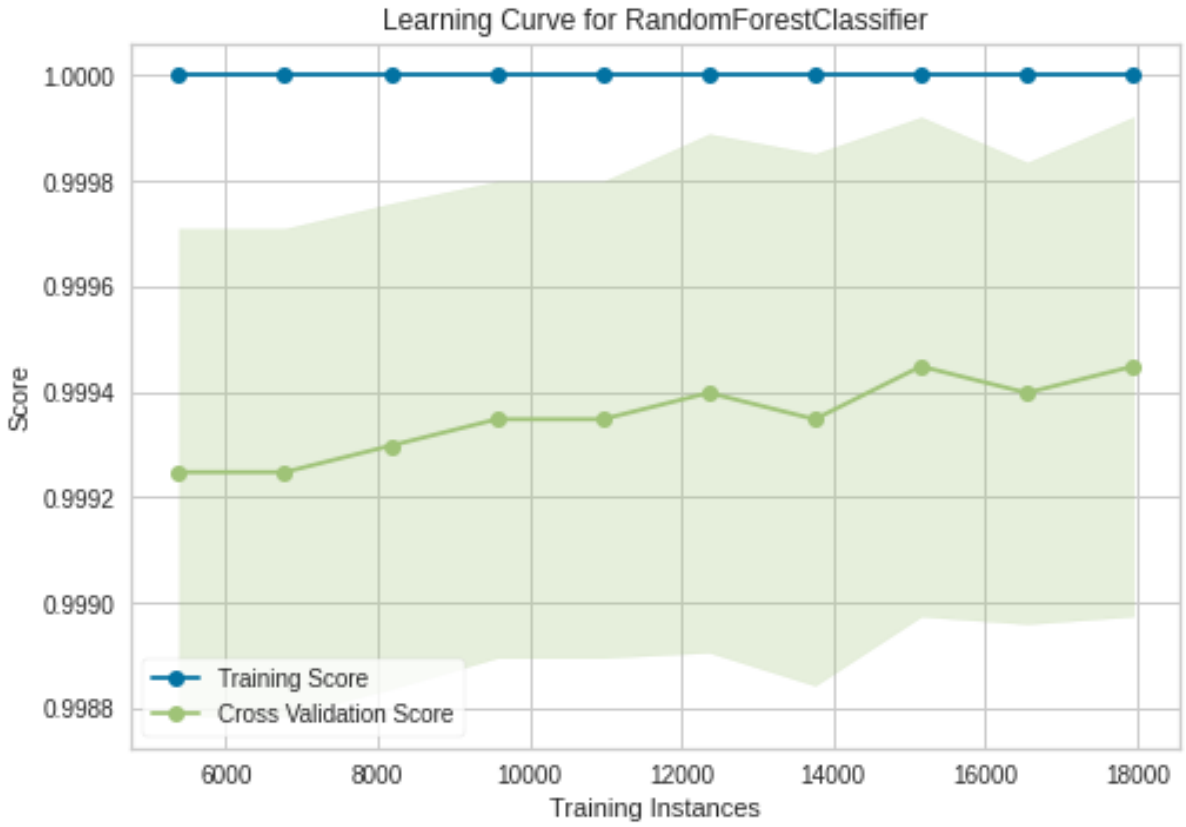
4.11 Implementation of Hyperparameter tuning in the study

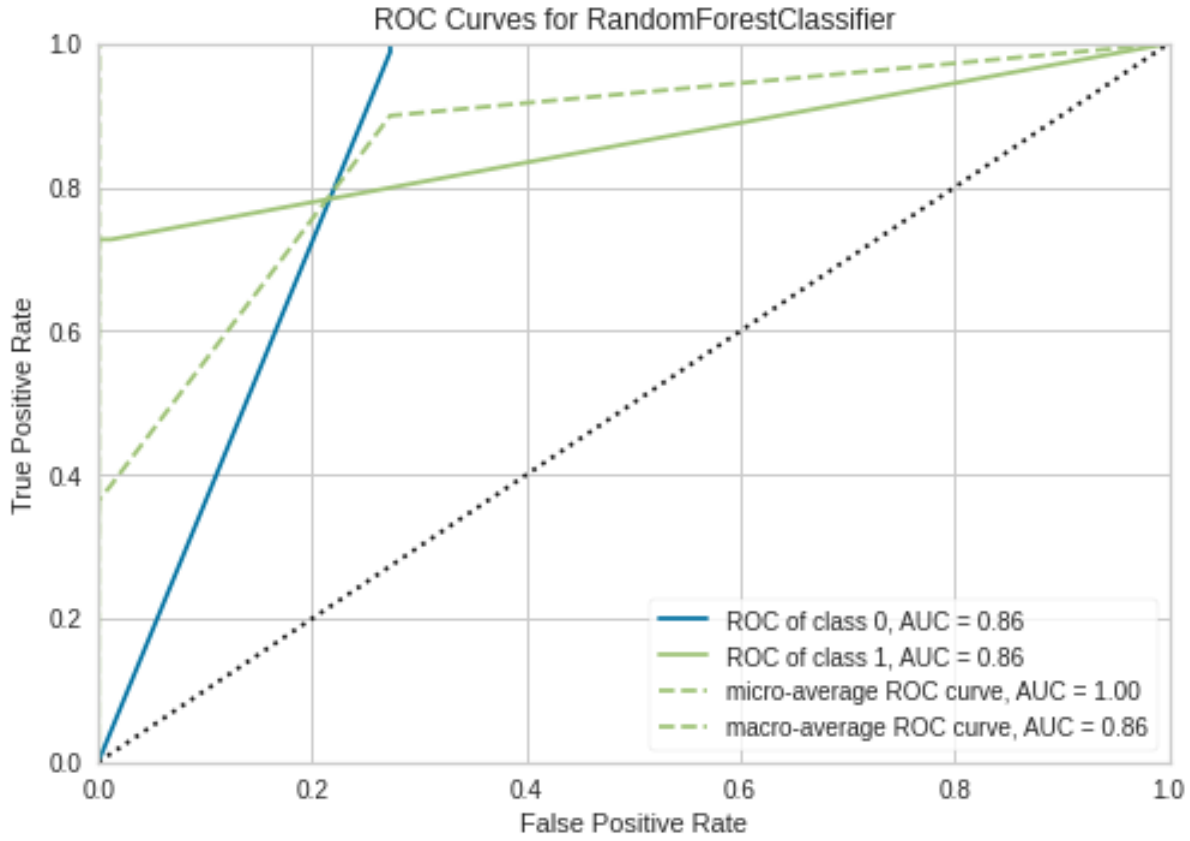
However, after it was found that over sampling technique was better and employed with the model providing with an accuracy of 99%. However, hyperparameter tuning was applied to the to improve the precision of the model as it contributes towards finding the best prediction possible with the model. Here in this study the aim was considered, enhanced fraud detection model is considered which is only possible when the parameters like precision and recall are higher for the values when predicted as fraud. To gain this hyperparameter tuning is considered. Traditional hyperparameter tuning seems to be hefty and needs a lot of coding as already here pycaret is employed, the tuning library from the pycaret module can be employed. The analysis on the findings is discussed below.

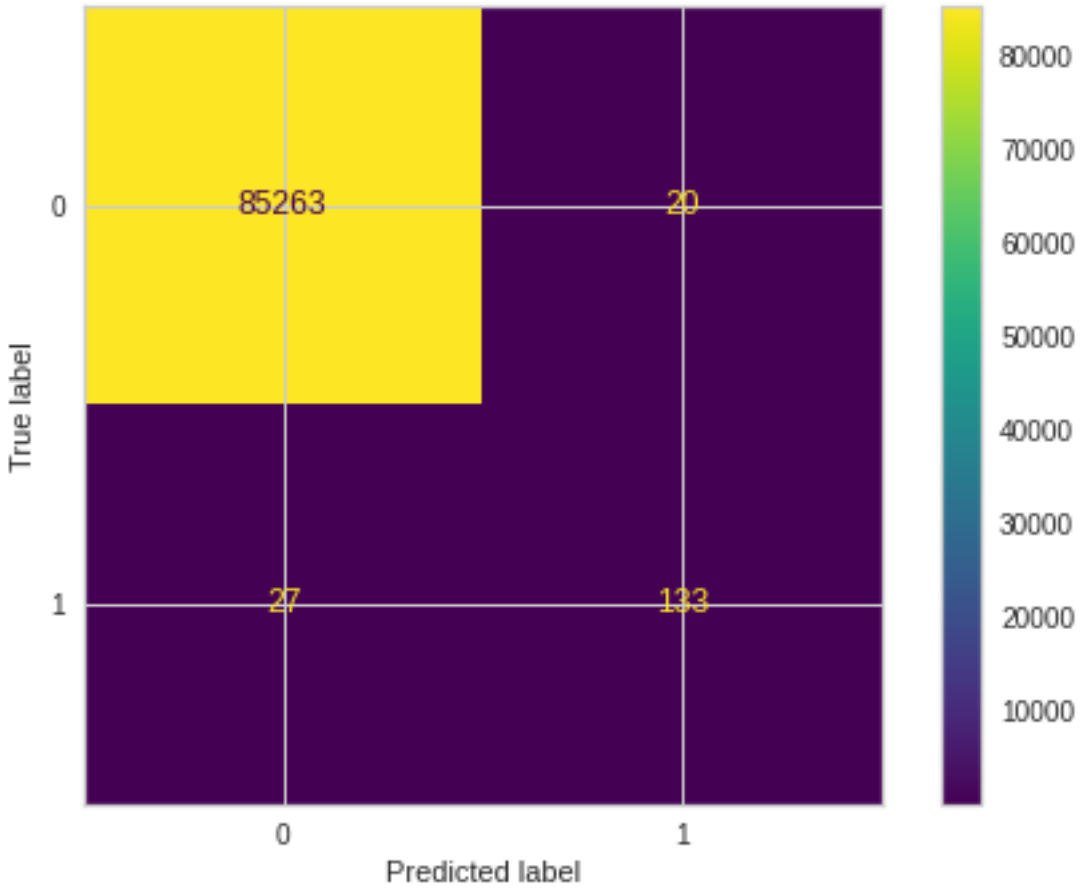
5 Analysis and Discussions

5.1 Learning curve and ROC curve evaluations for the model

Linear curves are used to learn about the variance and bias in the model and in the dataset. the Fig. 5.2 shows the learning curve graph between the scores for both training and validation. the x-axis consists of training instances wearing from 600 to 1800 and the y-axis consists of scores which are varying from 0.99 to 1. It can be seen that the training score always remains at 100% throughout the training instances. While cross-validation was occurring for the instances varying from 600 to 12000 number of scores remained to be 99.9 4% and however, a hike is seen after the training instances increases to 15000 (Fig. 5.1). As the curve indicates the validation




ROC Curves for RandomForestClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.86
ROC of class 1, AUC = 0.86
micro-average ROC curve, AUC = 1.00
macro-average ROC curve, AUC = 0.86


85263
20
27
133
0
1
True label
0
1
Predicted label
80000
70000
60000
50000
40000
30000
20000
10000

hence, to solve these two methods are proposed which	Hence, this methos is rejected	This method best suits and is utilised, and considered for hyperparameter.

is either under or Over sampling		

Coming to the under sampling, it is seen that sampling technique here has improved the classification testing accuracy. The class which has higher examples which is legitimate here in the dataset, cannot be sampled and quantified with to fill the minority class as even such methos is not applied the nature of random forest model shows better accuracy. The model is better off with the under-sampling method as it already is biased and performs well on the classifying the legitimate values. Overall accuracy of the random forest model was recorded to be 100% without adding any sampling technique. However, it is seen that there is a high problematic situation with the data set with imbalanced data classes. In the case of the nature of the data, it can be seen that if a small population of the data set has information related to be legitimate and not fraud it might be a chance that this kind of imbalance might affect then this kind of algorithm is applied in an overall population. When in a real time situation where only 10% of the data consists of Fraud based data then there might be chances that the model might predict to be more legitimate than fraud.

5.3 Compare the over sampled model before hyperparameter tunning and after hyperparameter tunning based on Classification report

The overall models were implemented. These models were evaluated using the classification report. This type of report is used for the performance evaluation of a classification model. Here the classification model used is random forest classifier. However, the enhancement of the data server and the problem related to the data set was solved using sampling technique. there were two types of sampling techniques applied under and oversampling. Among these over-sampling was selected as the conclusion of these techniques and applied to the random forest model. The overall classification report for three of these methods are seen in the Fig. 5.4.

Normal Random forest model

```
[[85274     9]
 [   35   125]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85283
           1       0.93      0.78      0.85       160

    accuracy                           1.00     85443
   macro avg       0.97      0.89      0.93     85443
weighted avg       1.00      1.00      1.00     85443
```

Under sampling Random forest model

```
[[66375 18908]
 [    6   154]]
              precision    recall  f1-score   support

           0       1.00      0.78      0.88     85283
           1       0.01      0.96      0.02       160

    accuracy                           0.78     85443
   macro avg       0.50      0.87      0.45     85443
weighted avg       1.00      0.78      0.87     85443
```

Over sampling Random forest model

```
[[85263    20]
 [   27   133]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85283
           1       0.87      0.83      0.85       160

    accuracy                           1.00     85443
   macro avg       0.93      0.92      0.92     85443
weighted avg       1.00      1.00      1.00     85443
```

Tuned model

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9995	1.0000	1.0000	0.7500	0.8571	0.8569	0.8658
1	0.9995	0.9997	1.0000	0.7500	0.8571	0.8569	0.8658
2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	0.9985	0.8729	0.5000	0.6667	0.5714	0.5707	0.5766
4	0.9995	0.9999	0.7500	1.0000	0.8571	0.8569	0.8658
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	0.9990	1.0000	0.3333	1.0000	0.5000	0.4996	0.5771
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	0.9990	0.9995	1.0000	0.6000	0.7500	0.7495	0.7742
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9995	0.9872	0.8583	0.8767	0.8393	0.8391	0.8525
Std	0.0005	0.0381	0.2358	0.1562	0.1729	0.1732	0.1567

Normal model

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9995	1.0000	1.0000	0.7500	0.8571	0.8569	0.8658
1	0.9995	0.9998	0.6667	1.0000	0.8000	0.7998	0.8163
2	0.9995	1.0000	0.6667	1.0000	0.8000	0.7998	0.8163
3	0.9985	0.8737	0.5000	0.6667	0.5714	0.5707	0.5766
4	0.9995	0.9997	0.7500	1.0000	0.8571	0.8569	0.8658
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	0.9990	1.0000	0.3333	1.0000	0.5000	0.4996	0.5771
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	0.9990	1.0000	1.0000	0.6000	0.7500	0.7495	0.7742
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9994	0.9873	0.7917	0.9017	0.8136	0.8133	0.8292
Std	0.0005	0.0379	0.2335	0.1539	0.1641	0.1643	0.1486

Table 2 Comparison of the execution time

Model methodology used	Execution time in seconds
Random Forest model	180.04 seconds
Random Forest model with under sampling technique	95 seconds
Random Forest model with over sampling technique	240.79 seconds

Hence from table 2 , it can be seen that when a normal random forest model was applied during Pycaret the combination of both starting time and time showed there was 180 seconds required for executive the model. But when under sampling technique was employed, the model showed a better execution time of only 95 seconds in contrast when over sampling technique was applied it basically showed up to 40 seconds which was higher than the normal random forest

model. This was quite expected because the sampling techniques increasing the labels in the rear and which somehow increases the size of the data server through which the execution time also increases this execution time is somewhat related to the training time of the model as well.

When the model was tuned the parameters set by the pycaret library was, Zero Alpha value, a balanced class weight with an entropy type of criterion with 10 as a maximum depth and log two to be selected as maximum number of features. Another hyper parameter set by the model was that there were no maximum number of samples selected with the maximum number of leaf nodes. A very low value that is 0.00032 as an impurity was implemented with nothing to be split with. The number of estimators with nine numbers of minimum sample splits made with 4 sample leaves per split with no jobs selected and a random state of 372 with zero verbose and no warm start.

6 Conclusions

Mastercard transactions are extremely common across the world. However, there's a third-party one that keeps track of our activity and gets folks in trouble. This was happening not only today, but also approximately twenty years ago, and with the help of the latest technological algorithms, fraud activity in certain areas like online shopping, marketing, and so on has expanded significantly. So, in order to detect such fraudulent activity, our research focuses on determining the proportion of fake data in a particular data collection. Over sampling technique is the best model because it’s giving better and similar performance with normal model. It can be seen from the Fig. 6.1, The overall conclusions gathered from the models and methods implemented.

	Accuarcy	precision for fraud lables	execution time	F1-score
normal model	100	93	180	85
under sampling	78	0	95	0
over sampling	100	87	240	85

-

  was much better than the model precision when under sampling was used. As a result, it can be concluded that the model exhibits improved precision in both scenarios when over sampling was done.
- The F-1 score for the normal model was recorded as 85 percent for the fraud labels in the model employed here, which is then preserved similarly to how the training set was sampled and the testing set was left unaltered.

Despite the fact that there are several fraud detection approaches, we cannot claim that our algorithm totally identifies fraud. The accuracy and recall values reported for the under-sampling approach, on the other hand, are of less value and are unworthy of being used to

forecast fraud levels. As a result, it can be concluded that using the random forest with an over sampling strategy resulted in significant improvements in the model's false positive rate.

7 References

Ajit, S., 2021. *Machine Learning Framework for End to End Implementation of Incident Duration Prediction* (Doctoral dissertation, Iowa State University).

Alharbi, A., Alshammari, M., Okon, O.D., Alabrah, A., Rauf, H.T., Alyami, H. and Meraj, T., 2022. A Novel text2IMG Mechanism of Credit Card Fraud Detection: A Deep Learning Approach. Electronics, 11(5), p.756.

Anelli, V.W., Di Noia, T., Di Sciascio, E., Pomo, C. and Ragone, A., 2019, September. On the discriminative power of hyper-parameters in cross-validation and how to choose them. In Proceedings of the 13th ACM Conference on Recommender Systems (pp. 447-451).

Bentejac, C., Csorgo, A. and Martinez-Munoz, G., 2021. A comparative analysis of gradient boosting algorithms. Artificial Intelligence Review, 54(3), pp.1937-1967.

Vuttipittayamongkol, P., Elyan, E., Petrovski, A. and Jayne, C., 2018, November. Overlap-based Varmedja, D., Karanovic, M., Sladojevic, S., Arsenovic, M. and Anderla, A., 2019, March. Credit card fraud detection-machine learning methods. In 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH) (pp. 1-5). IEEE.

Yang, L. and Shami, A., 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, pp.295-316.

Yazici, Y., 2020. Approaches to Fraud detection on credit card transactions using artificial intelligence methods. arXiv preprint arXiv:2007.14622.

Yee, O.S., Sagadevan, S. and Malim, N.H.A.H., 2018. Credit card fraud detection using machine learning as data mining technique. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 10(1-4), pp.23-27.