Maths for AI PBL
Tri- VIII

Topic: **Credit Card Fraud Detection using Smote Technique**

Soumyashri Singha - PB03
Dibyanshi Tripathy - PC09

I. INTRODUCTION

'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Necessary prevention measures can be taken to stop this abuse and the behaviour of such fraudulent practices can be studied to minimize it and protect against similar occurrences in the future.In other words, Credit Card Fraud can be defined as a case where a person uses someone else's credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used. Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behaviour, which consist of fraud, intrusion, and defaulting. This is a very relevant problem that demands the attention of communities such as machine learning and data science where the solution to this problem can be automated. This problem is particularly challenging from the perspective of learning, as it is characterized by various factors such as class imbalance. The number of valid transactions far outnumber fraudulent ones. Also, the transaction patterns often change their statistical properties over the course of time.

These are not the only challenges in the implementation of a real-world fraud detection system, however. In real world examples, the massive stream of payment requests is quickly scanned by automatic tools that determine which transactions to authorize. Machine learning algorithms are employed to analyse all the authorized transactions and report the suspicious ones. These reports are investigated by professionals who contact the cardholders to confirm if the transaction was genuine or fraudulent. The investigators provide a feedback to the automated system which is used to train and update the algorithm to eventually improve the fraud-detection performance over time.

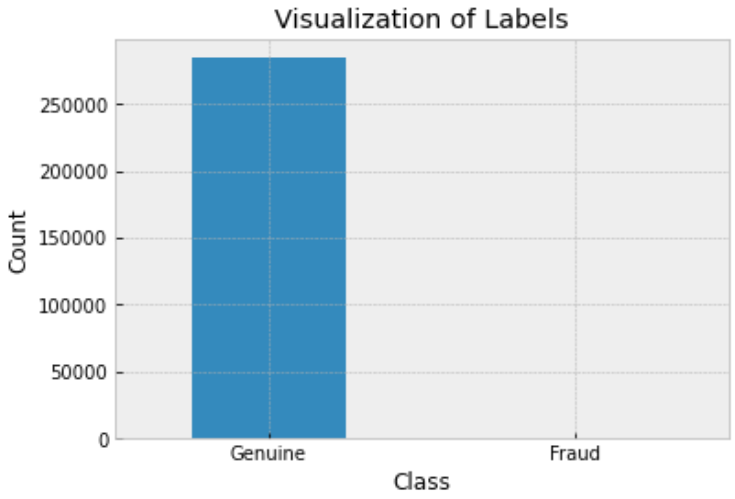
II. METHODOLOGY

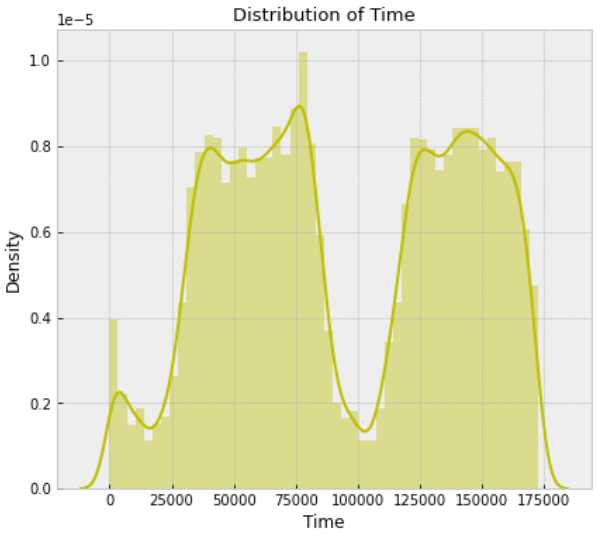
We use the following libraries and frameworks in credit card fraud detection projects.

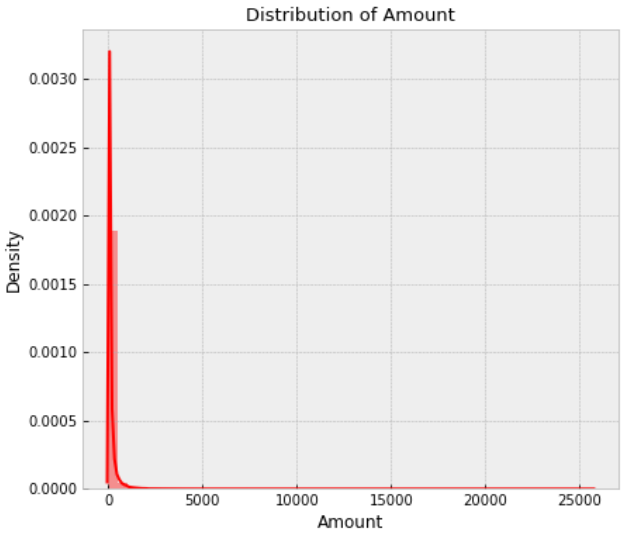
- Python – 3.x
- Numpy – 1.19.2
- Scikit-learn – 0.24.1
- Matplotlib – 3.3.4
- Imblearn – 0.8.0
- Collections, Itertools

Visualization of dataset:

We plot different graphs to check for inconsistencies in the dataset and to visually comprehend it:







III. ALGORITHM

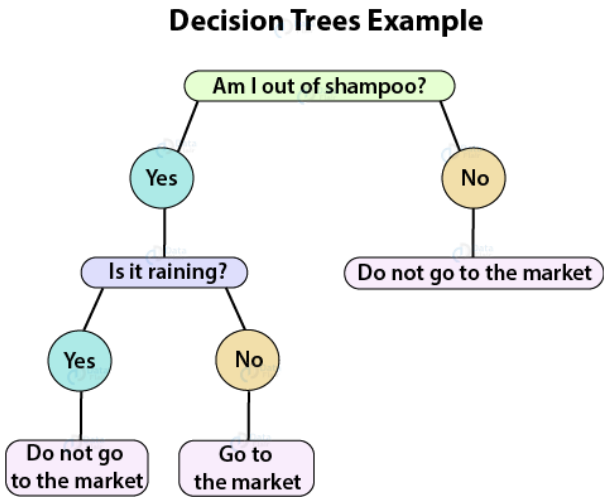
Let's train different models on our dataset and observe which algorithm works better for our problem. This is actually a binary classification problem as we have to predict only 1 of the 2 class labels. We can apply a variety of algorithms for this problem like Random Forest, Decision Tree, Support Vector Machine algorithms, etc.

In this machine learning project, we build Random Forest and Decision Tree classifiers and see which one works best. We address the "class imbalance" problem by picking the best-performed model.

But before we go into the code, let's understand what random forests and decision trees are.

1. Decision Tree Algorithm:

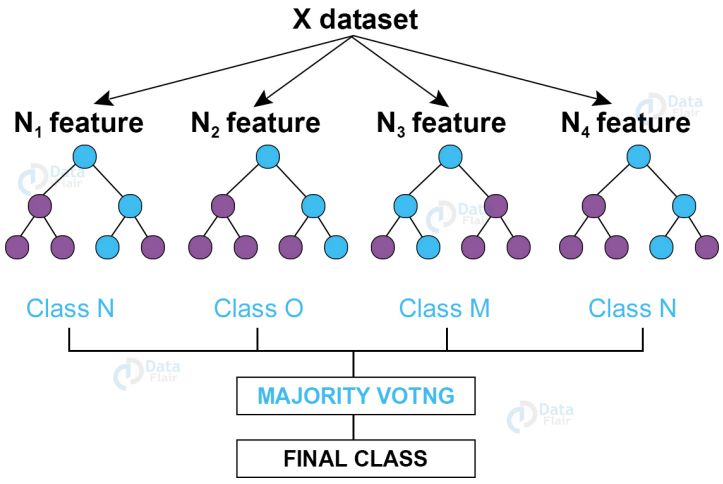
The Decision Tree algorithm is a supervised machine learning algorithm used for classification and regression tasks. The algorithm's aim is to build a training model that predicts the value of a target class variable by learning simple if-then-else decision rules inferred from the training data.



2. Random Forest Algorithm:

Random forest (one of the most popular algorithms) is a supervised machine learning algorithm. It creates a "forest" out of an ensemble of "decision trees", which are normally trained using the "bagging" technique. The bagging method's basic principle is that combining different learning models improves the outcome.

To get a more precise and reliable forecast, random forest creates several decision trees and merges them.



III. RESULTS

In our code we get the scores as:

```
In [11]: # Decision Tree Classifier
         decision_tree = DecisionTreeClassifier()
         decision_tree.fit(train_X, train_Y)

         predictions_dt = decision_tree.predict(test_X)
         decision_tree_score = decision_tree.score(test_X, test_Y) * 100
```

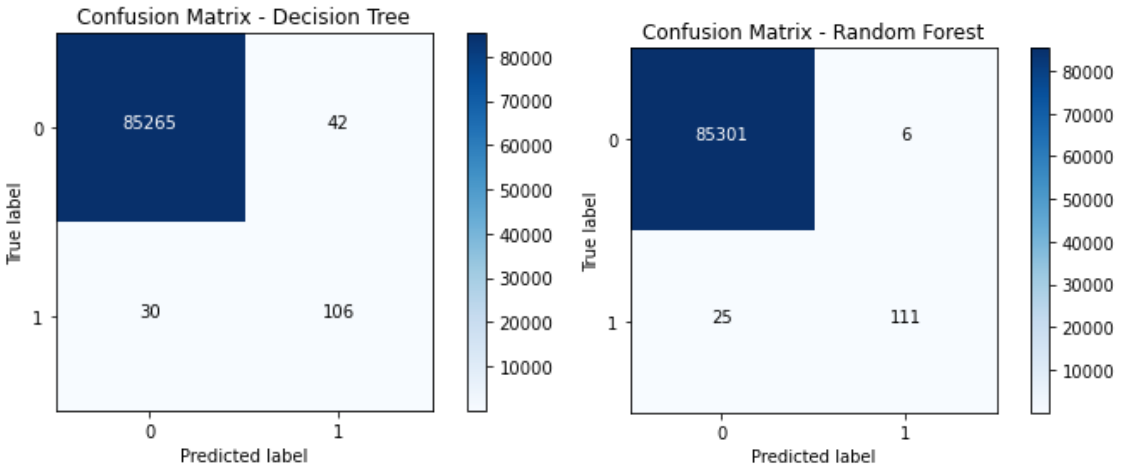
```
In [12]: # Random Forest
         random_forest = RandomForestClassifier(n_estimators= 100)
         random_forest.fit(train_X, train_Y)

         predictions_rf = random_forest.predict(test_X)
         random_forest_score = random_forest.score(test_X, test_Y) * 100
```

```
In [13]: # Print scores of our classifiers

         print("Random Forest Score: ", random_forest_score)
         print("Decision Tree Score: ", decision_tree_score)
```

```
Random Forest Score:  99.9637185023934
Decision Tree Score:  99.91573329588147
```



Comparing the confusion matrix calculations like accuracy, precision, F1 score, recall.

```
In [20]: print("Evaluation of Decision Tree Model")
         print()
         metrics(test_Y, predictions_dt.round())
```

```
Evaluation of Decision Tree Model

Accuracy: 0.99916
Precision: 0.71622
Recall: 0.77941
F1-score: 0.74648
```

```
In [21]: print("Evaluation of Random Forest Model")
         print()
         metrics(test_Y, predictions_rf.round())
```

```
Evaluation of Random Forest Model

Accuracy: 0.99964
Precision: 0.94872
Recall: 0.81618
F1-score: 0.87747
```

Comparing the above parameters, clearly Random Forest model works better than Decision Trees.

The Random Forest model works better than Decision Trees. But, if we observe our dataset suffers a serious problem of class imbalance. The genuine (not fraud) transactions are more than 99% with the credit card fraud transactions constituting 0.17%.

With such a distribution, if we train our model without taking care of the imbalance issues, it predicts the label with higher importance given to genuine transactions (as there is more data about them) and hence obtains more accuracy.

IV. CONCLUSION

The class imbalance problem can be solved by various techniques. Oversampling is one of them.

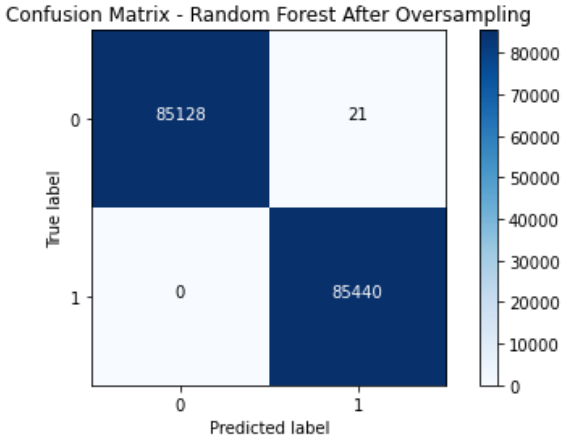
Oversample the minority class is one of the approaches to address the imbalanced datasets. The easiest solution entails doubling examples in the minority class, even though these examples contribute no new data to the model.

Instead, new examples may be generated by replicating existing ones. The Synthetic Minority Oversampling Technique, or SMOTE for short, is a method of data augmentation for the minority class.

The above SMOTE is present in the imblearn package. Let’s import that and resample our data.

In the following code below, we resampled our data and we split it using train_test_split() with a split of 70-30.

As the Random Forest algorithm performed better than the Decision Tree algorithm, we will apply the Random Forest algorithm to our resampled data.




```
In [28]: print("Evaluation of Random Forest Model")
         print()
         metrics(test_Y, predictions_resampled.round())
```

```
Evaluation of Random Forest Model

Accuracy: 0.99988
Precision: 0.99975
Recall: 1.00000
F1-score: 0.99988
```

Now it is evident that after addressing the class imbalance problem, our Random forest classifier with SMOTE performs far better than the Random forest classifier without SMOTE.

As seen from the final confusion matrix, the model was correctly able to classify 85128 records as valid and 85440 records as fraudulent. However, it incorrectly identified a valid transaction as fraudulent 21 times.