Project Report On

Find Default (Prediction of Credit Card Fraud)

Abstract

Now a day's online transactions have become an important and necessary part of our lives. It is vital that credit card companies are able to identify fraudulent credit card transactions so that customers are not charged for items that they did not purchase. As frequency of transactions is increasing, number of fraudulent transactions are also increasing rapidly. Such problems can be tackled with Machine Learning with its algorithms. This project intends to illustrate the modelling of a data set using machine learning with Credit Card Fraud Detection.

The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the data of the ones that turned out to be fraud. This model is then used to recognize whether a new transaction is fraudulent or not. Our objective here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. Credit Card Fraud Detection is a typical sample of classification. In this process, we have focused on analyzing and pre-processing data sets as well as the deployment of multiple anomaly detection algorithms such as Local Outlier Factor and Isolation Forest algorithm on the PCA transformed Credit Card Transaction data.

Introduction

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.

Due to rise and acceleration of E- Commerce, there has been a tremendous use of credit cards for online shopping which led to High amount of frauds related to credit cards. In the era of digitalization, the need to identify credit card frauds is necessary. Fraud detection involves monitoring and analyzing the behaviour of various users in order to estimate detect or avoid undesirable behaviour. In order to identify credit card fraud detection effectively, we need to understand the various technologies, algorithms and types involved in detecting credit card frauds. Algorithm can differentiate transactions which are fraudulent or not. Find fraud, they need to passed dataset and knowledge of fraudulent transaction. They analyse the dataset and classify all transactions.

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. Machine learning algorithms are employed to analyses 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.

In this paper, we describe Random forest algorithm applicable on Find fraud detection. Random forest has two types. They describe in detail and their accuracy 91.96% and 96.77% respectively. This paper summaries second type is better than the first type.

Many supervised machine learning algorithms apply on 70% training and 30% testing dataset. Random forest, SVM, Decision tree and KNN algorithms compare each other.

In this we have describe flow chart of fraud detection process. i.e. data Acquisition, data pre-processing, Exploratory data analysis and methods or algorithms are in detail.

Methodology

The approach that this paper proposes, uses the latest machine learning algorithms to detect anomalous activities, called outliers. The basic rough architecture diagram can be represented with the following figure: When looked at in detail on a larger scale along with real life elements, the full architecture diagram can be represented as follows: First of all, we obtained our dataset provided from **upGrad**. Inside this dataset, there are 31 columns out of which 28 are named as v1-v28 to protect sensitive data. The other columns represent Time, Amount and Class. Time shows the time gap between the first transaction and the following one. Amount is the amount of money transacted. Class 0 represents a valid transaction and 1 represents a fraudulent one.

We plot different graphs to check for inconsistencies in the dataset and to visually comprehend it: This graph shows that the number of fraudulent transactions is much lower than the legitimate ones. This graph shows the times at which transactions were done within two days.

It can be seen that the least number of transactions were made during night time and highest during the days. This graph represents the amount that was transacted. A majority of transactions are relatively small and only a handful of them come close to the maximum transacted amount. After checking this dataset, we plot a histogram for every column.

This is done to get a graphical representation of the dataset which can be used to verify that there are no missing any values in the dataset. This is done to ensure that we don’t require any missing value imputation and the machine learning algorithms can process the dataset smoothly.

After this analysis, we plot a heatmap to get a colored representation of the data and to study the correlation between out predicting variables and the class variable. This heatmap is shown below: The dataset is now formatted and processed. The time and amount column are standardized and the Class column is removed to ensure fairness of evaluation. The data is processed by a set of algorithms from modules.

The following module diagram explains how these algorithms work together: This data is fit into a model and the following outlier detection modules are applied on it:

- Local Outlier Factor
- Random Forest Algorithm

These algorithms are a part of sklearn. The ensemble module in the sklearn package includes ensemble-based methods and functions for the classification, regression and outlier detection.

This free and open-source Python library is built using NumPy, SciPy and matplotlib modules which provides a lot of simple and efficient tools which can be used for data analysis and machine learning.

It features various classification, clustering and regression algorithms and is designed to interoperate with the numerical and scientific libraries. Wave used Jupyter Notebook platform to make a program in Python to demonstrate the approach that this paper suggests. This program can also be executed on the cloud using Google Collab platform which supports all python notebook files. Detailed explanations about the modules with pseudocodes for their algorithms and output graphs are given as follows:

1. Local Outlier Factor

It is an Unsupervised Outlier Detection algorithm. 'Local Outlier Factor' refers to the anomaly score of each sample. It measures the local deviation of the sample data with respect to its neighbors.

More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local data. The pseudocode for this algorithm is written as: On plotting the results of Local Outlier Factor algorithm, we get the following figure: By comparing the local values of a sample to that of its neighbors, one can identify samples that are substantially lower than their neighbors.

These values are quite amanous and they are considered as outliers. As the dataset is very large, we used only a fraction of it in out tests to reduce processing times. The final result with the complete dataset processed is also determined and is given in the results section of this paper.

Implementation

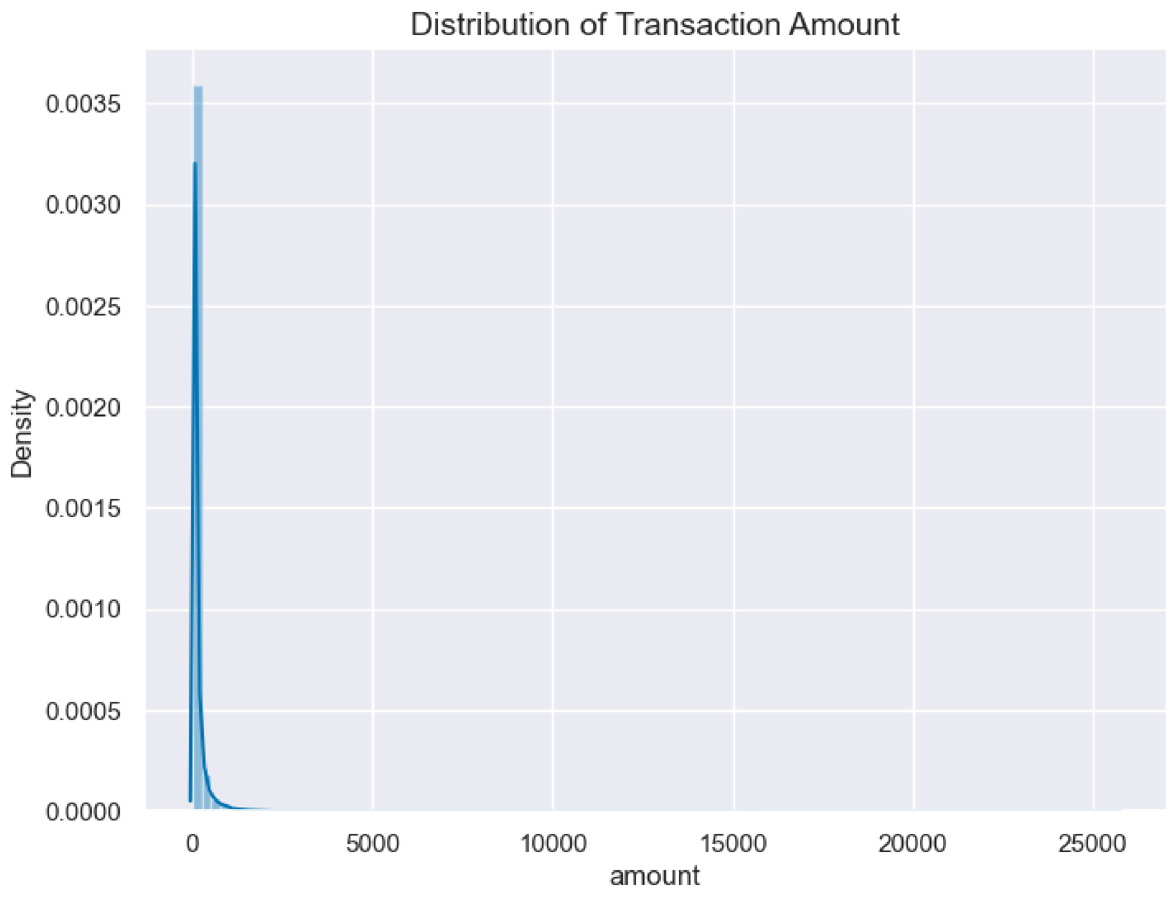
About dataset the datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Algorithm

- Step 1: Import dataset
- Step 2: Convert the data into data frames format
- Step3:Do random oversampling using SMOTE package
- Step4: Decide the amount of data for training data and testing data
- Step5: Give 80% data for training and remaining data for testing.
- Step6: Assign train dataset to the models
- Step7: Choose the algorithm among 4 different algorithms and create the model
- Step8: Make predictions for test dataset for each algorithm
- Step9: Calculate accuracy for each algorithm
- Step10: Apply confusion matrix for each variable
- Step11: Compare the algorithms for all the variables and find out the best algorithm.

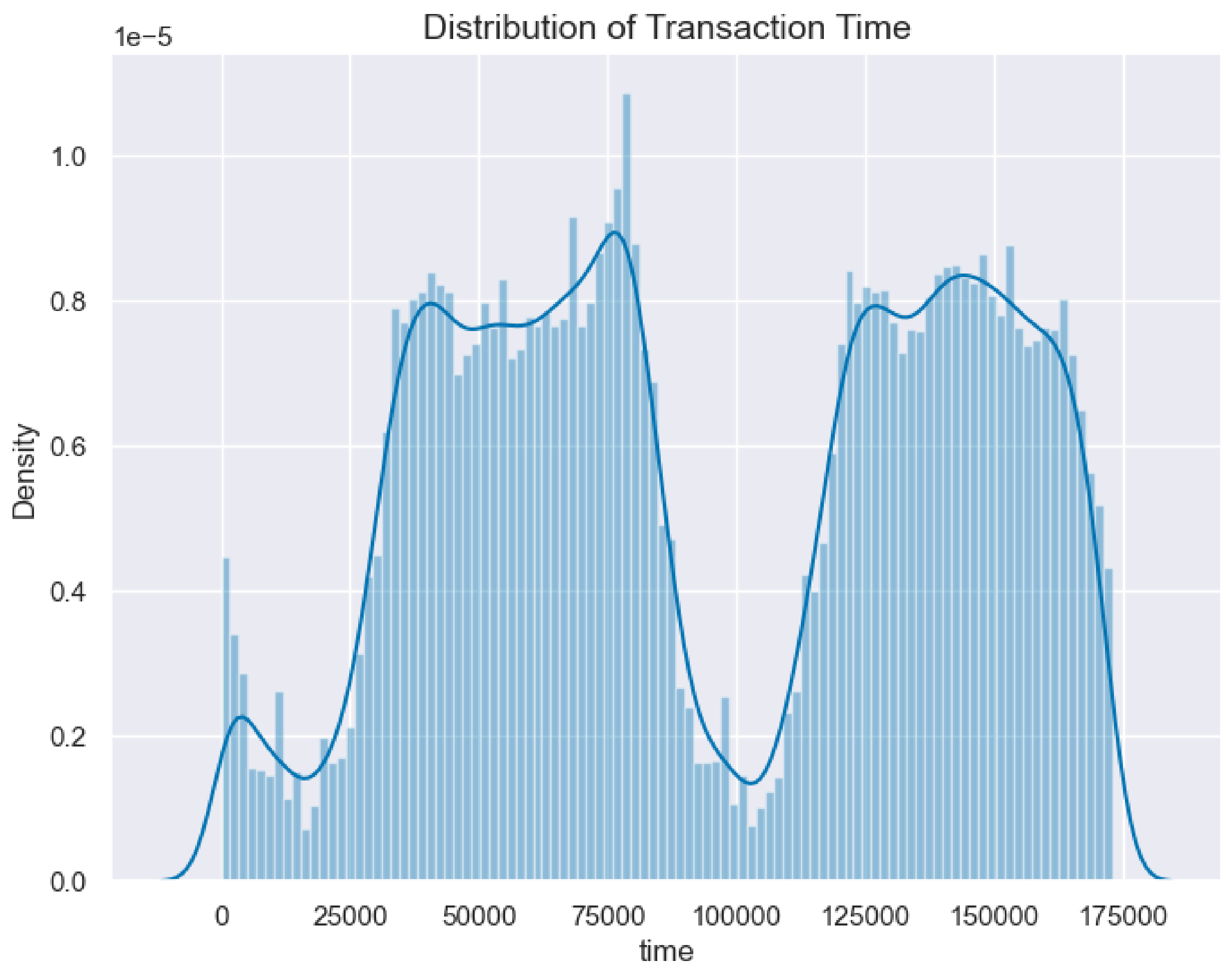
Exploratory Data Analysis (EDA)

Since nearly all predictors have been anonymised, I decided to focus on the non-anonymised predictor's time and amount of the transaction during my EDA. The data set contains 284,807 transactions. The mean value of all transactions is $88.35 while the largest transaction recorded in this data set amounts to $25,691.16. However, as you might be guessing right now based on the mean and maximum, the distribution of the monetary value of all transactions is heavily right-skewed. The vast majority of transactions are relatively small and only a tiny fraction of transactions comes even close to the maximum.

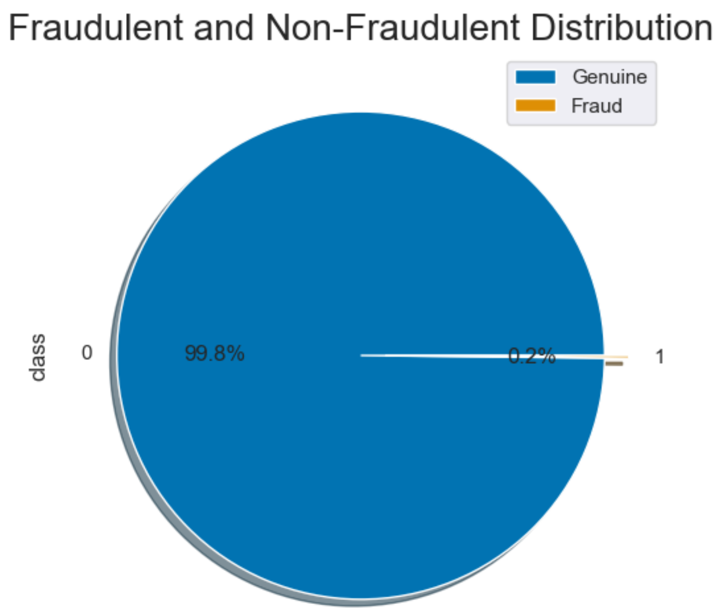


The time is recorded in the number of seconds since the first transaction in the data set. Therefore, we can conclude that this data set includes all transactions recorded over the course of two days. As opposed to the distribution of the monetary value of the transactions, it is bimodal.

This indicates that approximately 28 hours after the first transaction there was a significant drop in the volume of transactions. While the time of the first transaction is not provided, it would be reasonable to assume that the drop-in volume occurred during the night.



What about the class distributions? How many transactions are fraudulent and how many are not? Well, as can be expected, most transactions are no fraudulent. In fact, 99.83% of the transactions in this data set were not fraudulent while only 0.17% were fraudulent. The following visualization underlines this significant contrast.



Confusion Matrix

```
cnf_matrix = metrics.confusion_matrix(y_test,y_pred)
cnf_matrix
```

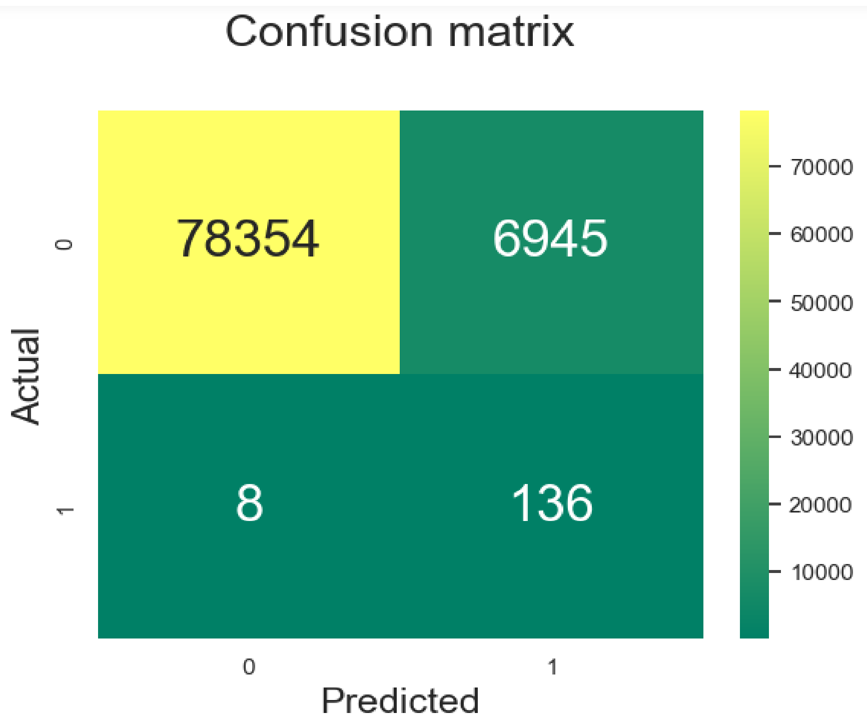
```
array([[85286,    13],
       [   56,    88]], dtype=int64)
```

```
# Heatmap for Confusion Matrix
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, annot_kws={"size": 25}, cmap="summer" ,fmt='g')

plt.title('Confusion matrix', y=1.1, fontsize = 22)
plt.ylabel('Actual',fontsize = 18)
plt.xlabel('Predicted',fontsize = 18)

# ax.xaxis.set_ticklabels(['Genuine', 'Fraud']);
# ax.yaxis.set_ticklabels(['Genuine', 'Fraud']);

plt.show()
```



There are 88 transaction recognised as True Postive, means they are originally fraud transactions and our model predicted them as fraud.

True Negative - 85286 (truly saying negative - genuine transaction correctly identified as genuine)

True Postive - 88 (truly saying positive - fraud transaction correctly identified as fraud)

False Negative - 56 (falsely saying negative - fraud transaction incorrectly identified as genuine)

False Positive - 13 (falsely saying positive - genuine transaction incorrectly identified as fraud)

Feature Engineering

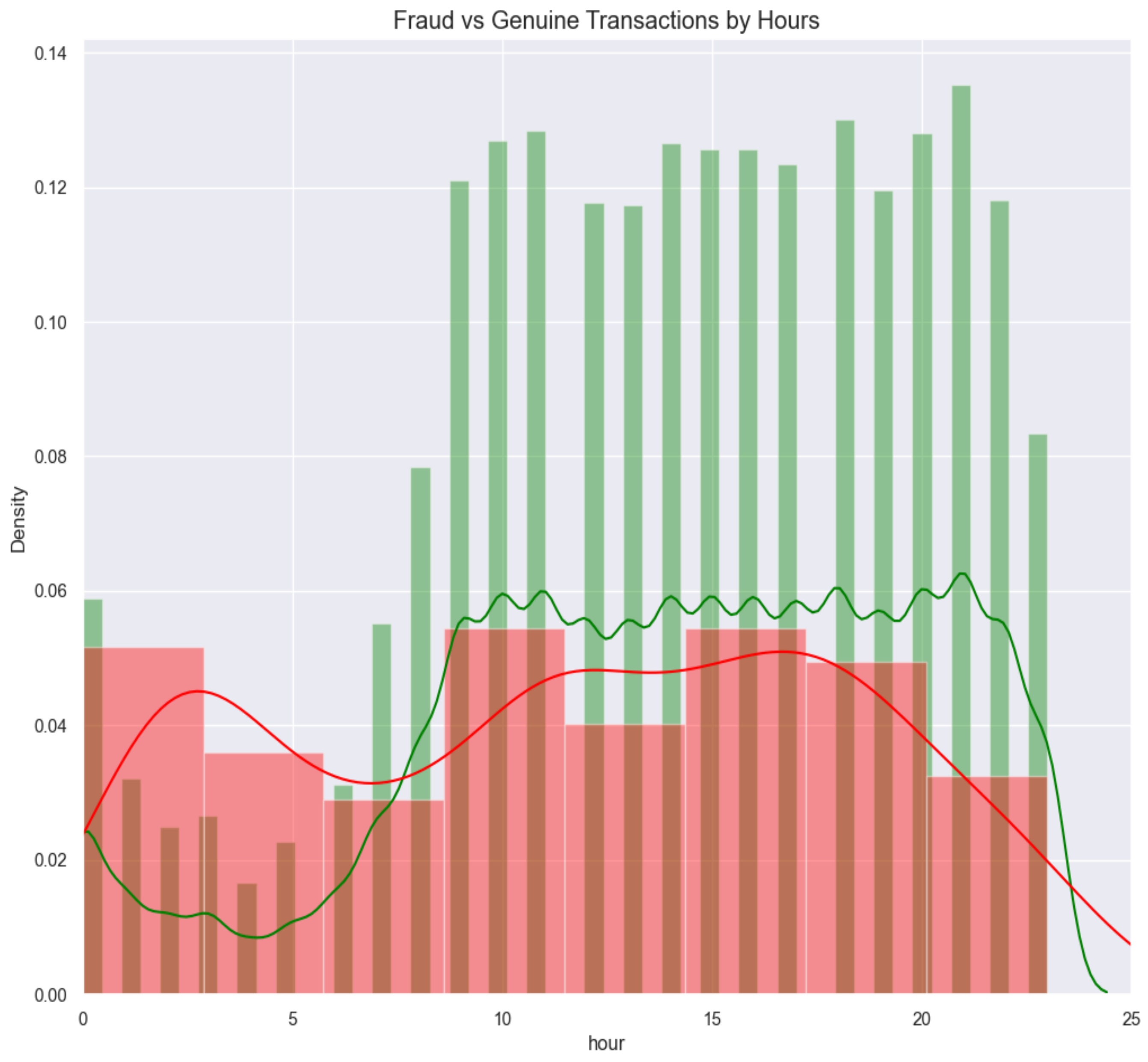
Feature engineering on Time

Converting time from second to hour.

Total number of transaction on Day 1 was 144787, out of which 281 was a fraud and 144506 was genuine. Fraud transaction was 0.19% of the total transaction on day 1.

Total number of transaction on Day 2 was 140020, out of which 211 was a fraud and 139809 was genuine. Fraud transaction was 0.15% of the total transaction on day 2.

Most of the transaction including the fraud transaction happened on day 1.



Model Selection

1. Classification Models

- Logistic Regression
- Decision Trees
- Random Forest
- Naive Bayes Classifier

2. Class Imbalance Solutions

- Under Sampling
- Over Sampling
- SMOTE
- ADASYN

3. Metrics

- Accuracy Score
- Confusion Matrix
- Precision Score
- Recall Score
- ROC_AUC
- F1 Score

1. Logistic Regression

Use logistic regression to detect credit card fraud. Logistic regression is the classical and the best bicategorical algorithm which is preferred when dealing with classification problems, especially bicategorical ones. The choice of algorithm is based on the principle of simplicity before complexity. Logistic regression is also an excellent choice because it is a recognised statistical method used to predict the outcome of a binomial or polynomial. A multinomial logistic regression algorithm can regenerate the model. It will be a better classification algorithm when the target field or data is a set field with two or more possible values.

The main methods of logistic regression method:

Objective: It is to look for some risk factor, then in this project, they want to find a particular transaction factor or reasons that are suspected of being fraudulent.

Prediction: Predicting the probability of fraud under other independent variables, based on different algorithmic models.

Judgment: It is somewhat similar to prediction. It is also based on different models to see how likelyit is that a transaction is a risk factor in a situation where fraud falls into a specific category

- Finding the h-function (i.e., the prediction function)

Constructing the predictive function h(x), the logistic function, or also known as the sigmoid function,we generally the first step is to build the predictive process, where the training data for the vector, as well as the best parameters. The basic form of the function shown in figure 1

$$g(z) = \frac{1}{1+e^{-z}}$$

- Constructing the J-function (loss function)

The second step is that we need to construct the loss function-j. In general, there will be m samples, each with n characteristics. The Cost and J functions are as follows, and they are derived based on maximum likelihood estimation (Sahin and Duman 2011).

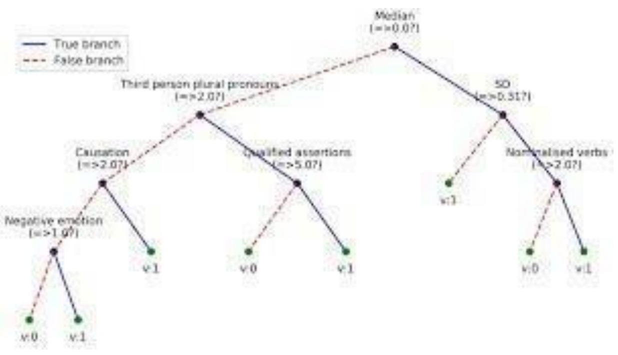
$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & if\ y = 1 \\ -\log(1 - h_\theta(x)) & if\ y = 0 \end{cases}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x_i), y_i) = -\frac{1}{m}\left[\sum_{i=1}^{m}(y_i log h_\theta(x_i) + (1 - y_i)\log(1 - h_\theta(x_i)))\right]$$

The Cost and J functions

2. Decision Tree

A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered. The model isa form of supervised learning, meaning that the model is trained and tested on a set ofdata that contains the desired categorization.



True branch
False branch
Median (=>0.07)
Third person plural pronouns (=>2.07)
SD (=>0.317)
Causation (=>2.07)
Qualified assertions (=>5.07)
Nominalised verbs (=>2.07)
Negative emotion (=>1.07)
v:1
v:0
v:1
v:0
v:1
v:0
v:1
v:0
v:1

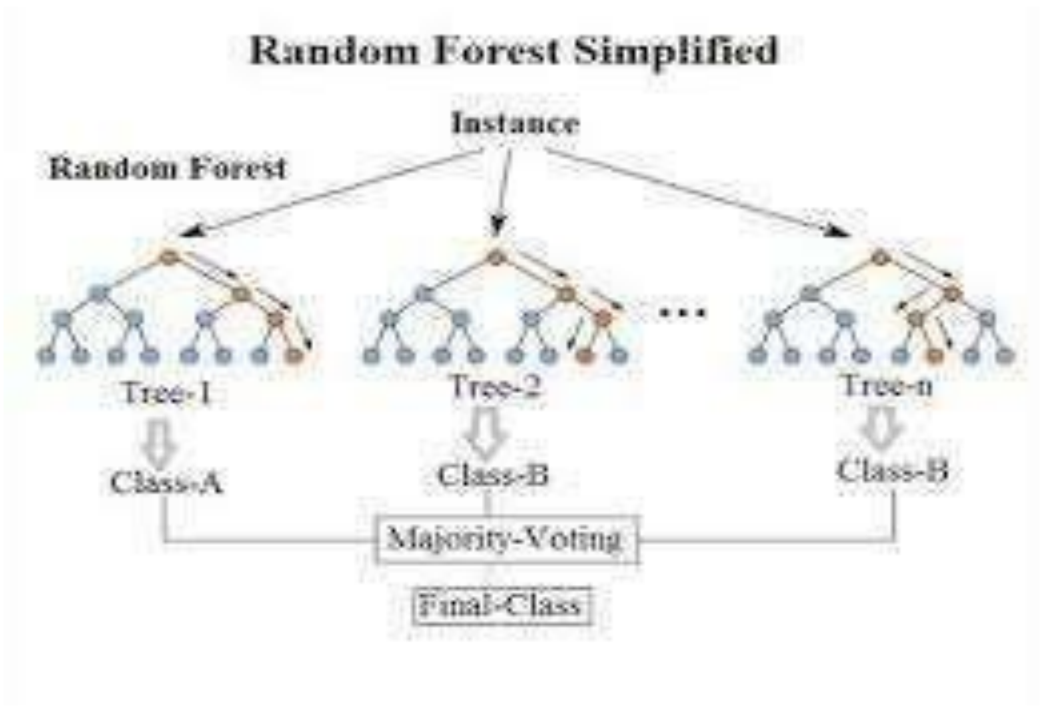
Decision tree Algorithm

Advantages

- Support vector machine works comparably well when there is an understand able margin of dissociation between classes.
- SVM is effective in instances where the number of dimensions is largerthan the number of specimens.
- Simple to understand and to interpret.
- Requires little data preparation.
- The cost of using the tree (i.e., predicting data) is logarithmic in the numberof data points used to train the tree.
- Able to handle both numerical and categorical data.
- Random forest classifier can be used to solve for regression or classification problems.
- The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a trainingset with replacement, called the bootstrap sample.

3. Random Forest Classifier

Features are cheekbone to jaw width, width to upper facial height ratio, perimeter to area ratio, eye size, lower face to face height ratio, face width to lower face height ratio and mean of eyebrow height. The extracted features are normalized and finally subjected to support regression.



Simplified Random Forest algorithm

4. k-nearest neighbour (KNN)

Initially proposed by Cover and Hart in 1968, Knn is a theoretically mature method that is one of the simplest of the data mining classification techniques. The term K nearest neighbours mean K nearest neighbours which says that its closest K neighbouring values can represent each sample. The nearest neighbour algorithm is a method of classifying every record in a data set.

The implementation principle of KNN nearest neighbour classification algorithm is: to determine the Category of unknown samples by taking all the examples of known types as a reference and at the same time calculate the distance between the new models and all the available pieces, from which thenearest K has known examples are selected, according to the rule of majority-majority-voting, the unknown samples(Bahnsen et al. 2014) and the K nearest models belong to a category with more categories (Duman et al. 2013).

$$d((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

The formula for calculating the distance between two points

The K value of the KNN algorithm in 'scikit-learn' is adjusted by the n_neighbors parameter, and the default value is 5.

Model Training

Splitting data into Training and Testing samples

We don't use the full data for creating the model. Some data is randomly selected and kept aside for checking how good the model is. This is known as Testing Data and the remaining data is called Training data on which the model is built. Typically 70% of data is used as training data and the rest 30% is used as testing data.

```
# Load the library for splitting the data
from sklearn.model_selection import train_test_split
```

```
# Split the data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, shuffle=True, random_state=101)
```

```
# Quick sanity check with the shapes of Training and testing datasets
print("X_train - ",X_train.shape)
print("y_train - ",y_train.shape)
print("X_test - ",X_test.shape)
print("y_test - ",y_test.shape)
```

```
X_train -  (199364, 29)
y_train -  (199364,)
X_test -  (85443, 29)
y_test -  (85443,)
```

Model Validation

Under Sampling and Over Sampling

Oversampling and undersampling in data analysis are techniques used to adjust the class distribution of a data set.

- Random oversampling duplicates examples from the minority class in the training dataset and can result in overfitting for some models.
- Random undersampling deletes examples from the majority class and can result in losing information invaluable to a model.

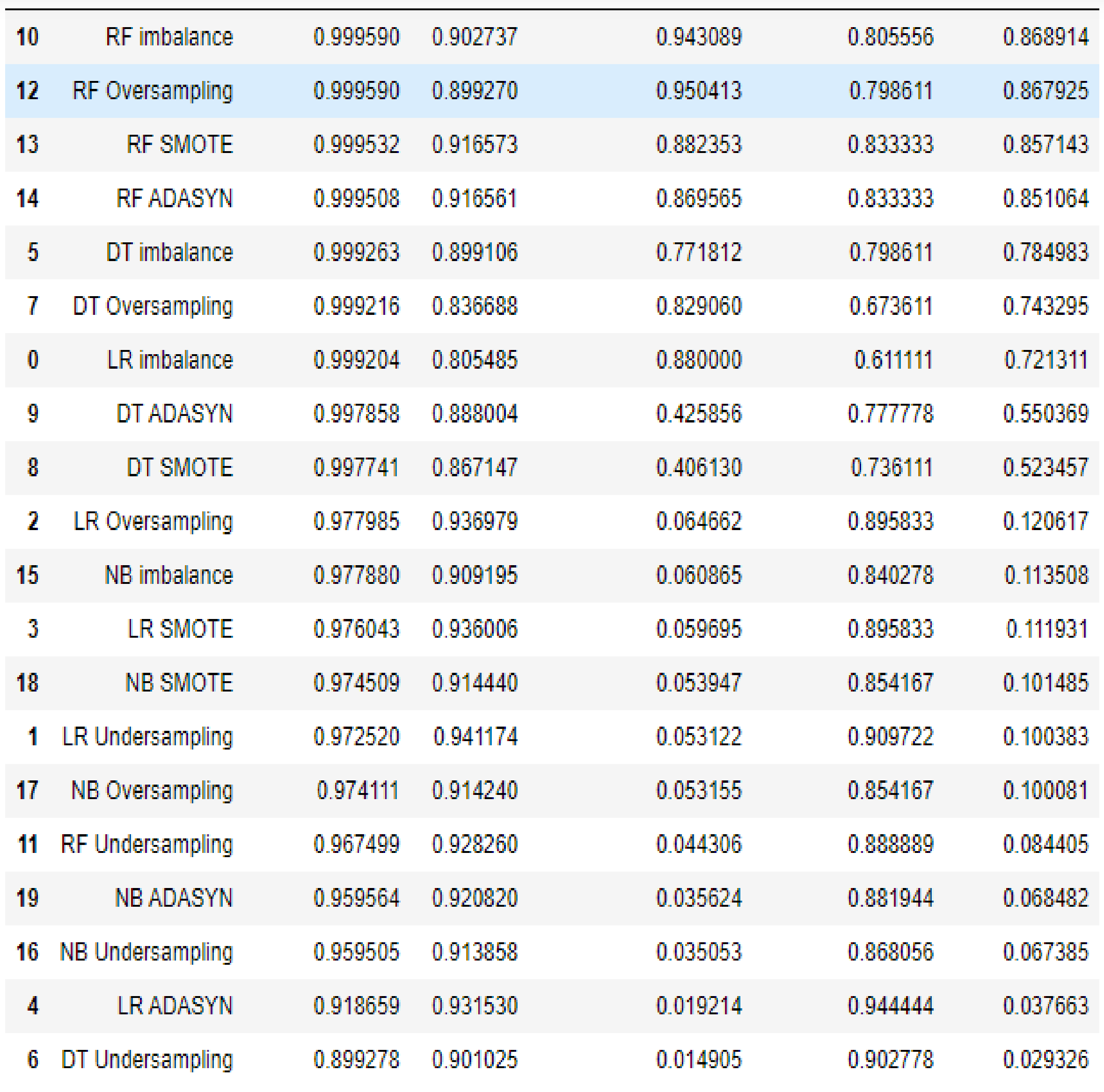
Synthetic Minority OverSampling Technique (SMOTE)

In this technique, instead of simply duplicating data from the minority class, we synthesize new data from the minority class. This is a type of data augmentation for tabular data can be very effective. This approach to synthesizing new data is called the Synthetic Minority Oversampling Technique, or SMOTE for short.

Adaptive Synthetic Sampling Method for Imbalanced Data (ADASYN)

ADASYN (Adaptive Synthetic) is an algorithm that generates synthetic data, and its greatest advantages are not copying the same minority data, and generating more data for “harder to learn” examples.

Performance measures of various classifiers



10	RF imbalance	0.999590	0.902737	0.943089	0.805556	0.868914
12	RF Oversampling	0.999590	0.899270	0.950413	0.798611	0.867925
13	RF SMOTE	0.999532	0.916573	0.882353	0.833333	0.857143
14	RF ADASYN	0.999508	0.916561	0.869565	0.833333	0.851064
5	DT imbalance	0.999263	0.899106	0.771812	0.798611	0.784983
7	DT Oversampling	0.999216	0.836688	0.829060	0.673611	0.743295
0	LR imbalance	0.999204	0.805485	0.880000	0.611111	0.721311
9	DT ADASYN	0.997858	0.888004	0.425856	0.777778	0.550369
8	DT SMOTE	0.997741	0.867147	0.406130	0.736111	0.523457
2	LR Oversampling	0.977985	0.936979	0.064662	0.895833	0.120617
15	NB imbalance	0.977880	0.909195	0.060865	0.840278	0.113508
3	LR SMOTE	0.976043	0.936006	0.059695	0.895833	0.111931
18	NB SMOTE	0.974509	0.914440	0.053947	0.854167	0.101485
1	LR Undersampling	0.972520	0.941174	0.053122	0.909722	0.100383
17	NB Oversampling	0.974111	0.914240	0.053155	0.854167	0.100081
11	RF Undersampling	0.967499	0.928260	0.044306	0.888889	0.084405
19	NB ADASYN	0.959564	0.920820	0.035624	0.881944	0.068482
16	NB Undersampling	0.959505	0.913858	0.035053	0.868056	0.067385
4	LR ADASYN	0.918659	0.931530	0.019214	0.944444	0.037663
6	DT Undersampling	0.899278	0.901025	0.014905	0.902778	0.029326

Highlights

After training each of the models, these are the final results. All of the scores for Random Forest with Oversampling technique and the Random Forest with SMOTE technique models are very promising for our dataset! Each model has a high true positive rate and a low false-positive rate, which is exactly what we're looking for.

In the ROC graph above, the AUC scores for Random Forest with Oversampling technique is pretty high, which is what we'd like to see. As we move further right along the curve, we both capture more True Positives but also incur more False Positives. This means we capture more fraudulent transactions, but also flag even more normal transactions as fraudulent. So Random Forest with Oversampling technique is our final model, as this gives highest F1 score of 86.47% on test datasets.

Data Preparation

Before continuing with our analysis, it is important not to forget that while the anonymised features have been scaled and seem to be centred around zero, our time and amount features have not. Not scaling them as well would result in certain machine learning algorithms that give weights to features (logistic regression) or rely on a distance measure (KNN) performing much worse. To avoid this issue,

I standardized both the time and amount column. Luckily, there are no missing values and we, therefore, do not need to worry about missing value imputation.

Creating a Training Set for a Heavily Imbalanced Data Set Now comes the challenging part: Creating a training data set that will allow our algorithms to pick up the specific characteristics that make a transaction more or less likely to be fraudulent. Using the original data set would not prove to be a good idea for a very simple reason:

Since over 99% of our transactions are no fraudulent, an algorithm that always predicts that the transaction is nonfraudulent would achieve an accuracy higher than 99%. Nevertheless, that is the opposite of what we want. We do not want a 99% accuracy that is achieved by never labelling a transaction as fraudulent, we want to detect fraudulent transactions and label them as such.

There are two key points to focus on to help us solve this. First, we are going to utilize random under-sampling to create a training dataset with a balanced class distribution that will force the algorithms to detect fraudulent transactions as such to achieve high performance. Speaking of performance, we are not going to rely on accuracy.

Instead, we are going to make use of the Receiver Operating Characteristics-Area under the Curve or ROC-AUC performance measure (I have linked further reading below this article).

Essentially, the ROC-AUC outputs a value between zero and one, whereby one is a perfect score and zero the worst. If an algorithm has a ROC-AUC score of above 0.5, it is achieving a higher performance than random guessing.

To create our balanced training data set, I took all of the fraudulent transactions in our data set and counted them. Then, I randomly selected the same number of non-fraudulent transactions and concatenated the two. After shuffling this newly created data set, I decided to output the class distributions once more to visualize the difference.

Outlier Detection & Removal

Outlier detection is a complex topic. The trade-off between reducing the number of transactions and thus volume of information available to my algorithms and having extreme outliers skew the results of your predictions is not easily solvable and highly depends on your data and goals.

In my case, I decided to focus exclusively on features with a correlation of 0.5 or higher with the class variable for outlier removal. Before getting into the actual outlier removal, let’s take a look at visualizations of those features

Grid Search

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.

A model hyper parameter is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyper parameter has to be set before the learning process begins. For example, c in Support Vector Machines, k in k-Nearest Neighbors, the number of hidden layers in Neural Networks.
In contrast, a parameter is an internal characteristic of the model and its value can be estimated from data. Example, beta coefficients of linear/logistic regression or support vectors in Support Vector Machines.

Future Enhancement

Detection, we did end up creating a system that can, with enough time and data, get very close to that goal. As with any such project, there is some room for improvementhere. The very nature of this project allows for multiple algorithms to be integrated together as modules and their results can be combined to increase the accuracy of the final result. This model can further be improved with the addition of more algorithmsinto it. However, the output of these algorithms needs to be in the same format as theothers. Once that condition is satisfied, the modules are easy to add as done in the code.

This provides a great degree of modularity and versatility to the project. More room for improvement can be found in the dataset. As demonstrated before, the precision of the algorithms increases when the size of dataset is increased. Hence, more data will surely make the model more accurate in detecting frauds and reduce the number of false positives. However, this requires official support from the banks themselves.

Result

Identify fraudulent credit card transactions. Given the class imbalance ratio, we recommend measuring the accuracy using the Area under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification. The code prints out the number of false positives it detected and compares it with the actual values. This is used to calculate the accuracy score and precision of the algorithms. The fraction of data we used for faster testing is 10% of the entire dataset. The complete dataset is also used at the end and both the results are printed. These results along with the classification report for each algorithm is given in the output as follows, where class 0 means the transaction was determined to be valid and 1 means it was determined as a fraud transaction.

Conclusion

Fraud detection is a complex issue that requires a substantial amount of planning before throwing machine learning algorithms at it. Nonetheless, it is also an application of data science and machine learning for the good, which makes sure that the customer’s money is safe and not easily tampered with. Future work will include a comprehensive tuning of the Random Forest algorithm I talked about earlier. Having a data set with non-anonymised features would make this particularly interesting as outputting the feature importance would enable one to see what specific factors are most important for detecting fraudulent transactions. As always, if you have any questions or found mistakes, please do not hesitate to reach out to me. A link to the notebook with my code is provided at the beginning of this article.