APRIL 28, 2021

Data Science Project

Final Report

Credit Card Fraud Detection using Machine Learning

Aswath Rangachari| Janani Balaji

Table of Contents

1.BACKGROUND

1.1 What is Credit Card Fraud Detection?

"Fraud detection is a set of activities that are taken to prevent money or property from being obtained through false pretenses."

Credit card frauds can be committed in different ways and in many industries. Most detection methods combine a variety of fraud detection datasets to form a connected overview of both valid and non-valid payment data to decide. The different types of data points used to detect frauds can be the IP address, geolocation, device identification, "BIN" data, global latitude/longitude, historic transaction patterns, and the actual transaction information. In practice, this means that merchants and issuers deploy analytically based responses that use internal and external data to apply a set of business rules or analytical algorithms to detect fraud.

1.2 Proposed Solution

It is important that credit card companies can recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. To identify and

classify fraudulent transactions in real time, machine learning models that learn from prior data and estimate the probability of a fraudulent credit card transaction can be used.

Once the Fraud Detection model is ready, the model can be integrated into real time systems so that it starts tracking the transactions. Whenever a transaction happens, the model can be used to predict whether it is a fraudulent transaction or not. Based on the predicted fraud probability for a transaction, the following outcomes are possible:

- If the probability of fraud is less than a certain threshold, for example 10%, then the transaction can be processed automatically.
- If the probability of fraud is between a certain range (between 10% and 80%), one level of an additional authentication mechanism like One Time Password (OTP) should be applied.
- If the probability of fraud is high (more than 80%), then the transaction should be frozen and processed manually to validate and approve the transaction.

2.DATASET DESCRIPTION

The ***Credit Card Fraud Detection*** dataset from Kaggle has been used for this project.

The data represents credit card transactions that occurred over two days in September 2013 by European cardholders. The dataset is credited to the Machine Learning Group at the Free University of Brussels (Université Libre de Bruxelles) and a suite of publications by Andrea Dal Pozzolo, et al.

All details of the cardholders have been anonymized via a principal component analysis (PCA) transform. Unfortunately, due to confidentiality issues, details regarding the original features and more background information about the data is not available. a total of 28 principal components of these anonymized features is provided. In addition, the time in seconds between transactions is provided, as is the purchase amount (presumably in Euros). Each record is classified as normal (class "0") or fraudulent (class "1") and the transactions are heavily skewed towards normal. Specifically, there are 492 fraudulent credit card transactions out of a total of 284,807 transactions, which is a total of about 0.172% of all transactions.

3. EXPLORATORY DATA ANALYSIS

In this step, the dataset was analyzed to perform the following:

- Removing duplicate records
- Handling Missing value
- Normalizing and Scaling (Numerical Variables)

3.1 Removing duplicate rows

Rows that have identical data are probably useless, if not dangerously misleading during model evaluation. Here, a duplicate row is a row where each value in each column for that row appears in identically the same order (same column values) in another row.

From an algorithm evaluation perspective, duplicate rows will result in misleading performance. For example, if you are using a train/test split or k-fold cross-validation, then it is possible for a duplicate row or rows to appear in both train and test datasets and any evaluation of the model on these rows will be (or should be) correct. This will result in an optimistically biased estimate of performance on unseen data.

The dataset was checked for duplicate rows and a total of 1081 records were found to be duplicated. These records were removed from the dataset.

```
column_names = credit_card_df.columns.to_list()

print("Number of duplicates in dataset: "+str(credit_card_df.duplicated(column_names).sum()))
```

```
Number of duplicates in dataset: 1081
```

```
# dropping ALL duplicte values
credit_card_df.drop_duplicates(subset = column_names,
                               keep = 'first', inplace = True)
print("Total records after removing duplicates: "+str(len(credit_card_df)))
```

```
Total records after removing duplicates: 283726
```

Fig 1. Code snippet - Duplicate records removal

3.2 Handling missing values

There is a possibility of having few missing values in one or more columns of the dataset. Before modelling, these missing values can be handled in the following ways:

- **Drop the missing values:** In this case, we drop the missing values from those variables. In case there are very few missing values you can drop those values.
- **Impute with mean value:** For the numerical column, you can replace the missing values with mean values. Before replacing with mean value, it is advisable to check that the variable should not have extreme values i.e., outliers.
- **Impute with median value:** For the numerical column, you can also replace the missing values with median values.
- **Impute with mode value:** For the categorical column, you can replace the missing values with mode values i.e. the frequent ones.

In our case, the dataset did not have any missing values. Hence, missing value imputation was not required.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 283726 entries, 0 to 283725
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    283726 non-null  float64
 1   V1      283726 non-null  float64
 2   V2      283726 non-null  float64
 3   V3      283726 non-null  float64
 4   V4      283726 non-null  float64
 5   V5      283726 non-null  float64
 6   V6      283726 non-null  float64
 7   V7      283726 non-null  float64
 8   V8      283726 non-null  float64
 9   V9      283726 non-null  float64
 10  V10     283726 non-null  float64
 11  V11     283726 non-null  float64
 12  V12     283726 non-null  float64
 13  V13     283726 non-null  float64
 14  V14     283726 non-null  float64
 15  V15     283726 non-null  float64
 16  V16     283726 non-null  float64
 17  V17     283726 non-null  float64
 18  V18     283726 non-null  float64
 19  V19     283726 non-null  float64
 20  V20     283726 non-null  float64
 21  V21     283726 non-null  float64
 22  V22     283726 non-null  float64
 23  V23     283726 non-null  float64
 24  V24     283726 non-null  float64
 25  V25     283726 non-null  float64
 26  V26     283726 non-null  float64
 27  V27     283726 non-null  float64
 28  V28     283726 non-null  float64
 29  Amount  283726 non-null  float64
 30  Class   283726 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.1 MB
```

Fig 2. Code output - Missing value details across all columns

3.3 Normalizing and scaling numerical variables

Machine learning algorithms like linear regression, logistic regression etc. that use gradient descent as an optimization technique require data to be scaled. The difference in ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here is the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature, respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0.
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1.
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1.

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Here is the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

μ is the mean of the feature values and σ is the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

In the dataset, the variables V1 to V28 consists of values within a very small range - between -120 to 120. However, the variables Time & Amount seem to have a bigger range of values (0 to 25000). Hence, we can scale both the variables before modelling. The ***StandardScalar package from the sklearn library*** was used to standardise the values.

4. MODEL DEVELOPMENT

4.1 Dependent variable distribution

The dependent variable 'Class' is highly imbalanced since most of the transactions in the dataset represent non-fraud transactions and very few samples represent the fraudulent transactions. This is understandable from the business aspect as well since fraudulent transactions are very infrequent.

To represent in percentages, 99.83% of the data represent non-fraud transactions and only 0.17% of the data represent fraud transactions.

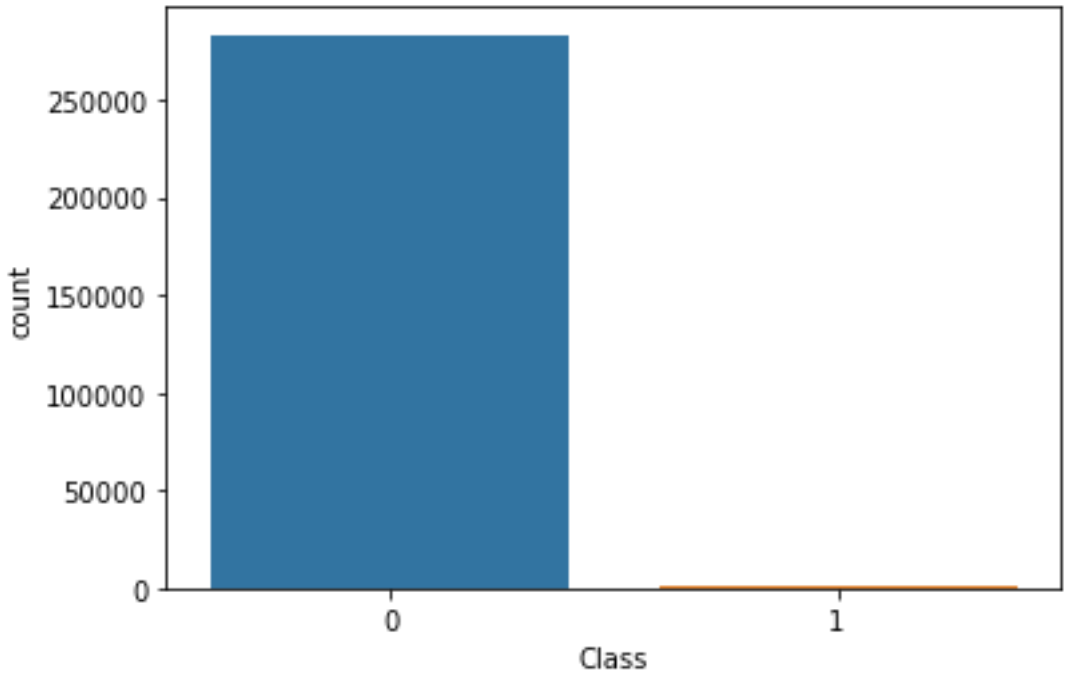

Fig 3. Dependent variable 'Class' distribution

4.1.1 Imbalanced classification

Imbalanced classifications pose a challenge for predictive modelling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. This is a problem because typically, the minority class is more important and therefore the problem is more sensitive to classification errors for the minority class than the majority class.

4.1.2 Handling Imbalance in dataset

Dealing with imbalanced datasets entails strategies such as improving classification algorithms or balancing classes in the training data (data pre-processing) before providing the

data as input to the machine learning algorithm. The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class. This is done to obtain approximately the same number of instances for both the classes. Let us look at a few resampling techniques:

- **Random Under-Sampling** - Random Under Sampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

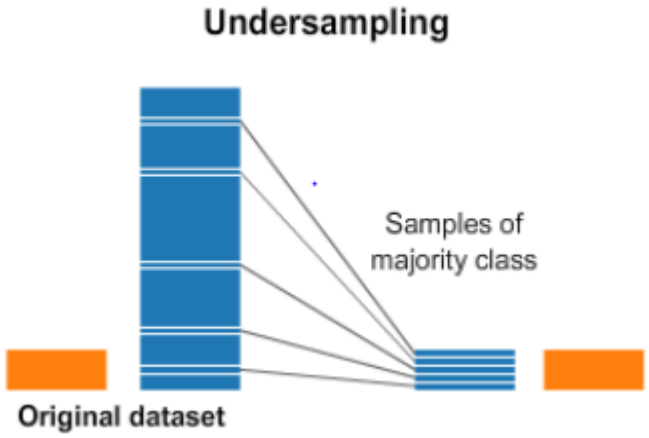

Fig 4. Undersampling technique

- **Random Over-Sampling -** Over-Sampling increases the number of instances in the minority class by randomly replicating them to present a higher representation of the minority class in the sample.

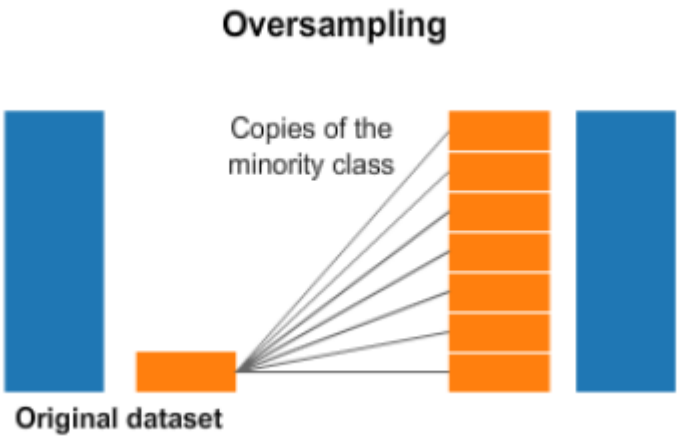

Fig 4. Oversampling technique

- **Cluster-Based Over Sampling -** In this case, the K-means clustering algorithm is independently applied to minority and majority class instances. This is to identify clusters in the dataset. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size.

- **Synthetic Minority Over-sampling Technique (SMOTE)** - This technique is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models.

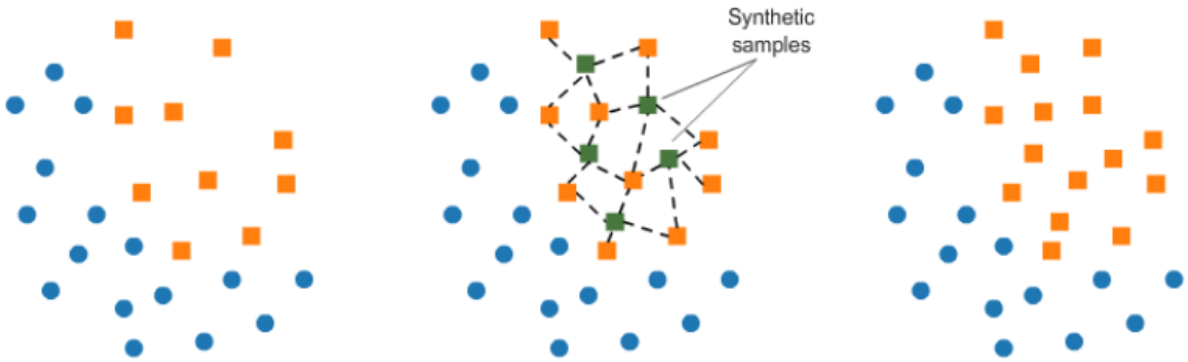

Fig 6. SMOTE technique

Another way to handle imbalance in the data is by using class weights. Most machine learning algorithms are not very useful with biased class data. This can be modified by making the current training algorithm to consider the skewed distribution of the classes. This can be achieved by giving different weights to both the majority and minority classes. The difference in weights will influence the classification of the classes during the training phase. The whole purpose is to penalize the misclassification made by the minority class by setting a higher-class weight and at the same time reducing weight for the majority class.

Most of the sklearn classifier modelling libraries have an in-built parameter "class_weight" which helps us optimize the scoring for the minority class just the way we have learned so far. By default, the value of class_weight=None, i.e., both the classes have been given equal weights. Other than that, we can either give it as 'balanced' or we can pass a dictionary that contains manual weights for both the classes. When the class_weights = 'balanced', the model automatically assigns the class weights inversely proportional to their respective frequencies.

In our case, since the minority class (Fraud class) has very less samples, applying oversampling is very challenging since we will not be able to generate enough samples to

match the majority class. Hence, we use a combination of under sampling and class weights to train the machine learning models. The simplest resampling method is using the random under sampling technique which randomly selecting examples from the majority class and deleting them from the training dataset.

For resampling methods, we can use the imblearn package which comes in-built with various resampling strategies. When we use any sampling technique, we divide the data first into train & test sets and then apply sampling on the training data only. Once the model is trained, we evaluate the model on the test set that contains only the original samples.

4.1.3 Evaluation metrics for imbalanced dataset

There are standard metrics that are widely used for evaluating classification predictive models, such as classification accuracy or classification error.

Standard metrics work well on most problems, which is why they are widely adopted. But all metrics make assumptions about the problem or about what is important in the problem. Therefore, an evaluation metric must be chosen that best captures what is important about the model or predictions, which makes choosing model evaluation metrics challenging. This challenge is made even more difficult when there is a skew in the class distribution. The reason for this is that many of the standard metrics become unreliable or even misleading when classes are imbalanced, or severely imbalanced, such as 1:100 or 1:1000 ratio between a minority and majority class.

Normally, the following classification metrics are used to evaluate a machine learning model:

- **Accuracy:** the proportion of the total number of predictions that were correct
- **Positive Predictive Value or Precision:** the proportion of positive cases that were correctly identified.
- **Negative Predictive Value:** the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall:** the proportion of actual positive cases which are correctly identified.
- **Specificity:** the proportion of actual negative cases which are correctly identified.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

Fig 7. Classification metrics

The above metrics may not be good enough for imbalanced classification problems. In case of imbalanced data, we can use the ROC Curve or ROC Analysis and the recall score to

evaluate our models. These two metrics help us to evaluate the models **based on their ability to discriminate both the classes and identify the fraudulent transactions effectively.**

ROC is an acronym that means Receiver Operating Characteristic and summarizes a field of study for analysing binary classifiers based on their ability to discriminate classes. A ROC curve is a diagnostic plot for summarizing the behaviour of a model by calculating the false positive rate and true positive rate for a set of predictions by the model under different thresholds.

- The true positive rate is the recall or sensitivity.

  TruePositiveRate = TruePositive / (TruePositive + FalseNegative)

- The false positive rate is calculated as:

  FalsePositiveRate = FalsePositive / (FalsePositive + TrueNegative)

Each threshold is a point on the plot and the points are connected to form a curve. A classifier that has no skill (e.g., predicts the majority class under all thresholds) will be represented by a diagonal line from the bottom left to the top right.

Any points below this line have worse than no skill. A perfect model will be a point in the top right of the plot.

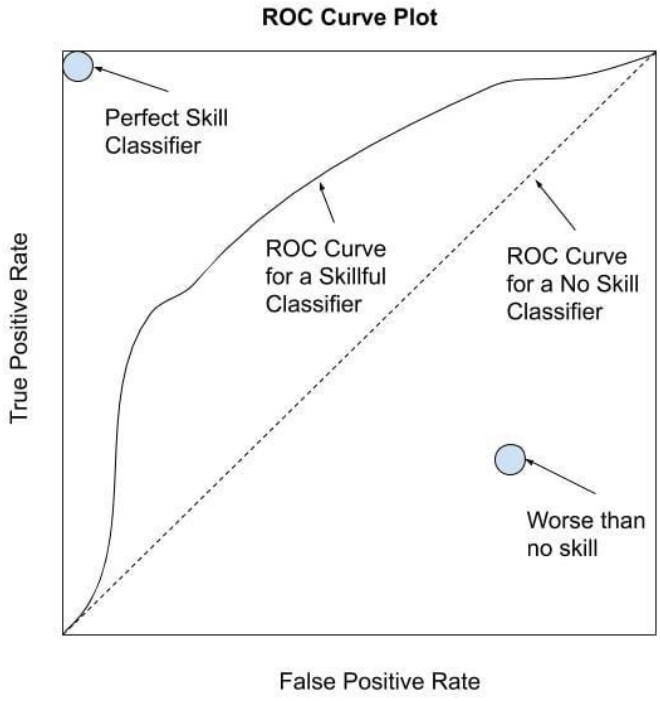

Fig 8. ROC Curve explanation

The **recall score** is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.

4.2 Train-Test split

The data set for developing the classification models was split using a stratified split with 75% of the observations composing the training test and 25% of the observations composing the test set.

4.3 Hyperparameter Tuning Methodology using Cross Validation

The following methodology was used when defining the best set of hyperparameters for each model:

1. Decide which hyperparameters must be tuned for each model, considering the ones that may have more influence in the model behaviour, and considering that a high number of parameters would require a lot of computational time.
2. Define a grid of possible values and performed a Randomized Search using 3-Fold Cross Validation (with 50 iterations).
3. Finally, using the best hyperparameters, perform a Grid Search using 3-Fold Cross Validation centred in those values to exhaustively search in the hyperparameter space for the best performing combination. Once the range is narrowed down using randomized search, perform a concentrated our search and explicitly specify every combination of settings to try.
4. The reason behind choosing K = 3 as the number of folds and 50 iterations in the randomized search comes from the trade-off between shorter execution time or testing a high number of combinations.

Note: When choosing the best model in the process, accuracy is chosen as the evaluation metric. The ROC AUC score and recall score will be used finally to compare the model performance on the test set.

4.3.1 Random Forest

Random forests or random decision forests are an ensemble learning method (bagging) that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (in the case of classification). Random decision forests correct for decision trees' habit of overfitting to their training set.

Hyperparameter	Brief Description
n_estimators	Number of trees in the forest
max_features	Maximum number of features considered for splitting a node
max_depth	Maximum number of levels in each decision tree
min_samples_split	Minimum number of data points placed in a node before the node is split.
min_samples_leaf	Minimum number of data points allowed in a leaf node
bootstrap	Method for sampling data points

Table 1. Hyperparameter tuning – Random Forest

4.3.2 Support Vector Machine

Support Vector Machine is a supervised machine learning algorithm which is mostly used in classification problems. The classification is performed by finding the hyperplane that best differentiates the classes.

Hyperparameter	Brief Description
C	Penalty parameter C of the error term
kernel	Specifies the kernel type to be used in the algorithm
gamma	Kernel coefficient
degree	Degree of the polynomial kernel function

Table 2. Hyperparameter tuning – Support Vector Machine

4.3.3 Logistic Regression

Logistic regression is a class of regression where the independent variable is used to predict the dependent variable. When the dependent variable has two categories, then it is a binary logistic regression. When the dependent variable has more than two categories, then it is a multinomial logistic regression.

Hyperparameter	Brief Description
C	Regularization parameter
solver	Algorithm to use in optimization problem
class_weights	Weights associated with the classes
penalty	Used to specify the norm used in penalization

Table 3. Hyperparameter tuning – Logistic Regression

4.3.4 Gradient Boosting

Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant $t - 1$. The outcomes predicted correctly are given a lower weight and the ones misclassified are weighted higher. In the case of gradient boosting, the weak learners are normally decision trees.

Note: Tree-Specific Parameters are the same ones as in the Random Forest.

Hyperparameter	Brief Description
learning_rate	Learning rate shrinks the contribution of each tree by learning rate
subsample	The fraction of samples to be used for fitting the individual base learners

Table 4. Hyperparameter tuning – Gradient Boosting

4.4 Model selection

Apart from the four models discussed in the hyperparameter tuning section, we used a voting classifier to combine the different machine learning models.

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on majority. The idea is instead of creating separate dedicated models and finding the accuracy for each of them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Voting Classifier supports two types:

- **Hard Voting:** In hard voting, the predicted output class is a class with the highest majority i.e. the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class (A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.
- **Soft Voting:** In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So, the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.

Note: We will use Soft Voting since we need the predicted probabilities to get the ROC AUC scores.

Before using hyperparameter tuning, the baseline accuracies of the models were evaluated to compare the performances before and after hyperparameter tuning. A Baseline model is the one where the default values are used for the model hyperparameters. The baseline model accuracies of the various models are as follows:

Model	Training accuracy
Logistic Regression (LRC)	0.96
Support Vector Classifier (SVC)	0.95
Random Forest Classifier (RFC)	0.95
Gradient Boosting Classifier (GBC)	0.96

Table 5. Baseline model accuracies

Model	Training Set Accuracy	Test Set Accuracy	ROC AUC Score – Test set	Recall score – Test set
Gradient Boosting Classifier (GBC)	1	0.99421	0.98392	0.86992
Support Vector Classifier (SVC)	0.9574	0.99466	0.98338	0.86179
Voting Classifier 1 (RFC, GBC & SVC)	1	0.99334	0.983	0.86992
Voting Classifier 2 (RFC, SVC & LRC)	0.97134	0.99132	0.98247	0.86992
Voting Classifier 3 (SVC, LRC & GBC)	0.97521	0.99258	0.98127	0.86992
Voting Classifier 4 (RFC, LRC & GBC)	1	0.99274	0.98009	0.86992
Logistic Regression (LRC)	0.96747	0.98653	0.97803	0.88618
Random Forest Classifier (RFC)	1	0.99421	0.97736	0.86992

Table 6. Model metrics - Sorted by ROC AUC score and Recall score.

The table above shows the performance of the various models on the training and testing dataset. The table is sorted based on the ROC AUC score and Recall score on the test set.

Overall, all the models yield good accuracy, ROC AUC and recall scores when evaluated on the test data set. The Gradient Boosting Classifier yields the highest ROC AUC & recall score and hence, is selected as the best model.

The classification report of the Gradient Boosting model on the test set is as follows:

Classification report

	precision	recall	f1-score	support
Non-fraud	1.00	0.99	1.00	71079
Fraud	0.21	0.87	0.34	123
accuracy			0.99	71202
macro avg	0.61	0.93	0.67	71202
weighted avg	1.00	0.99	1.00	71202

Fig 9. Classification report – Gradient Boosting Classifier

The confusion matrix for the Gradient Boosting model on the test set is as follows:

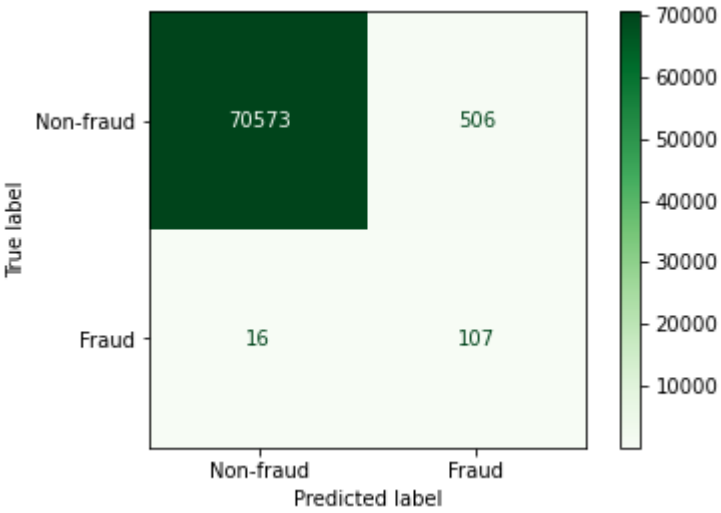

Fig 10. Confusion matrix – Gradient Boosting Classifier

The ROC curve for the Gradient Boosting model on the test set is as follows:

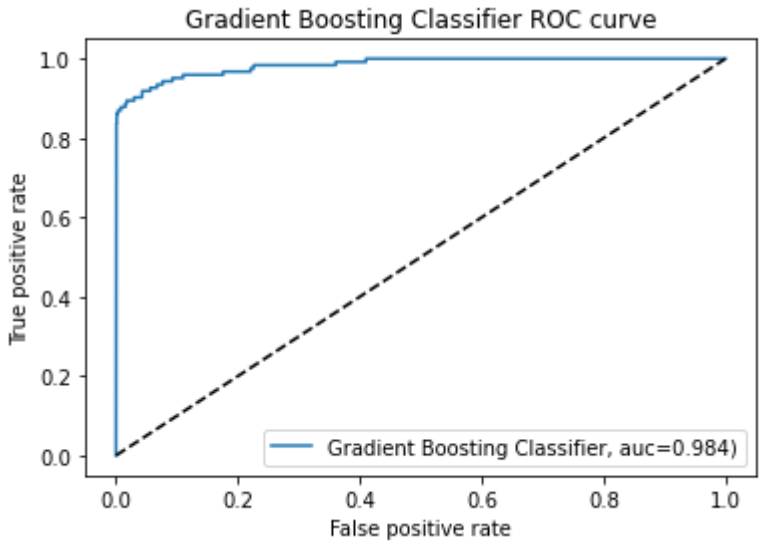

Fig 11. ROC Curve – Gradient Boosting Classifier

Note: The trained models were saved as pickle files so that they can be used in the future for predicting on new data.

5. CONCLUSION

To sum up,

- The CRISP-DM approach was used to solve the given problem in a systematic way. CRISP-DM breaks the process into six major phases:
    - Business Understanding
    - Data Understanding
    - Data Preparation
    - Modelling
    - Evaluation
    - Deployment (Not needed for this project)

- Different machine learning models were explored to develop a robust model for credit card fraud detection. All the models have produced good results in both training and testing with very minute differences in performance.

- The imbalance in the dataset was handled accordingly by using sampling and class weights during model training.

- During the model training phase, by using Randomized Search and Grid Search Cross Validation for Hyperparameter Tuning, the training accuracy of the models improved considerably in comparison with the baseline models (without hyperparameter tuning).

- Given that the dataset is imbalanced, getting a good accuracy is not the goal. However, the high AUC scores and Recall scores indicate that the models are quite robust in classifying fraudulent transactions and non-fraudulent transactions.

6. REFERENCES

https://spd.group/machine-learning/credit-card-fraud-detection/#What_is_Credit_Card_Fraud_Detection

https://machinelearningmastery.com/imbalanced-classification-with-the-fraudulent-credit-card-transactions-dataset/

https://www.analyticsvidhya.com/blog/2020/08/exploratory-data-analysiseda-from-scratch-in-python/

https://machinelearningmastery.com/basic-data-cleaning-for-machine-learning/

https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/

https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html

https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/