Credit Card Fraud Detection

Problem Statement:

Fraudulent activities have increased severalfold, with around 52,304 cases of credit/debit card fraud reported in FY'19 alone. Due to this steep increase in banking frauds, it is the need of the hour to detect these fraudulent transactions in time in order to help consumers as well as banks, who are losing their credit worth each day.

Every fraudulent credit card transaction that occur is a direct financial loss to the bank as the bank is responsible for the fraud transactions as well it also affects the overall customer satisfaction adversely.

Goal:

The primary objective of this project is to build a machine-learning model to detect fraudulent credit card transactions. This will help credit card companies mitigate risks and protect their customers.

Dataset:

The dataset contains transactions made by European cardholders in September 2013. It covers transactions that occurred over two days, including 492 frauds out of 284,807 transactions. Given this, the dataset is highly unbalanced, with the positive class (frauds) accounting for only 0.172% of all transactions.

The dataset has the following structure:

• Rows: Each row represents a transaction.

• Columns: There are 31 columns:

- **o Time:** The time elapsed between this transaction and the first transaction in the dataset (in seconds).
- **o V1 to V28:** The result of a PCA (Principal Component Analysis) transformation applied to the original features (for confidentiality reasons).
- **o Amount:** The transaction amount.
- **o Class:** The class label, where 1 indicates a fraudulent transaction and 0 indicates a legitimate transaction.

Approach:

The approach to the problem can be divided into below parts:

1. Data Understanding, Data Preparation and EDA

First look at the data used here from the kaggle dataset suggests that it is highly imbalanced in nature. The positive class (frauds) account for only 0.172% of all transactions:

Class is the **target** variable which we have to predict where 0 is normal transaction and 1 is fraudulent transaction.

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 exampledependant cost-senstive learning.

Since the PCA transformed variable are already Gausian, there is **no need for normalisation.**

We can start with the **basic EDA like correlation, boxplots etc for outliers.**

Next, we can use **transformation to mitigate and check the skewness in the data.** (Boxcox, Log transformation, Yeo-Johnson etc)

Data Exploration and Preprocessing

Data Loading and Initial Exploration:

• The dataset is loaded using pandas, and initial exploration involves checking the first few rows, summary statistics, and data types of the columns.

Summary Statistics:

• Summary statistics provide insights into the distribution and scale of the data.

Checking for Missing Values:

• The dataset is checked for missing values, and it was found that there are no missing values in any of the columns. This ensures that the dataset is complete and ready for analysis without the need for imputation.

Distribution of Classes:

• The distribution of the classes (fraudulent vs. non-fraudulent transactions) is examined to understand the class imbalance, which is crucial for model training and evaluation.

Handling Data Imbalance:

As we see that the data is heavily imbalanced, several approaches are employed to handle this imbalance:

• Under sampling:

o For balancing the class distribution, the number of non-fraudulent transactions is reduced to match the count of fraudulent transactions (492).

• Oversampling:

o The number of fraudulent transactions is increased to match the count of non

Class Imbalances:

Class 0 1 Count 284315 492 The normal **Oversampling** method won’t be used here as it does not add any new information to the dataset and **Undersampling** will also not be used as it leads to the loss of information.

Next, we will try the below two class imbalance handling techniques:

- **SMOTE** is a process where you can **generate new data points**, which lie vectorially between two data points that belong to the minority class.

- **ADASYN** is similar to SMOTE, with a minor change i.e. the number of synthetic samples that it will add will have a **density distribution**. The aim here is to create synthetic data for minority examples that are harder to learn, rather than the easier ones.

2. Model Selection and Model Building:

We will start building the model with the **train-test split**. (At least 100 class 1 rows should be there in the test split), use the **stratified split** here. (80-20 ratio can be used) We need to find which ML model works good with the imbalance data and have better results on the test data.

- **Logistic regression** works best when the data **is linearly separable** and **needs to be interpretable.**

- **KNN** is also highly interpretable, but not preferred when we have a huge amount of data as **it will consume a lot of computation.**

- **The decision tree model** is the first choice when we want the output to be **intuitive**, but they tend to overfit if left unchecked.

- **KNN** is a simple, **supervised machine learning algorithm** used for both classification and regression tasks. The k value in KNN should be an **odd number** because you have to take the majority vote from the nearest neighbours by breaking the ties.

- **In Gradient Boosted machines/trees**. newly added trees are trained to **reduce the errors (loss function)** of earlier models.

- **XGBoost** is an extended version of **gradient boosting**, with additional features like regularization and parallel tree learning algorithm for finding the best split.

We will start with the Logistic regression model with the different value of regularisation and hyper param tuning, then go to the decision tree and so on etc. We will also try ensemble models and use **votingclassifier, Bagging, Boosting etc** on the best performing individual models to find the best model.

Hyperparameter Tuning:

- When the data is imbalanced or less, it is better to use **K-Fold Cross Validation** for evaluating the performance when the data set is randomly split into ‘k’ groups.

- **Stratified K-Fold Cross Validation** is an extension of K-Fold cross-validation, in which we rearrange the data to ensure that each fold is a good representative of all the strata of the data.

- When you have a small data set, the computation time will be manageable to test out different hyperparameter combinations. In this scenario, it is advised to use a grid search.

- But, with large data sets, it is advised to use a randomized search because the sampling will be random and not uniform.

In our case, we will use the **Stratified K-Fold Cross Validation** as the dataset is not really huge

Choosing the Best Model:

- Various models were built using the different balanced datasets (Undersampling, Oversampling, SMOTE, and ADASYN).

- While many models performed well, it's important to consider the best-performing model based on key factors:
    - o Under sampling techniques, despite good performance, result in loss of information and are less preferred.
    - o Models built with SMOTE and ADASYN generally performed well.
    - o The XGBOOST model with Random Oversampling and StratifiedKFold CV dataset showed excellent performance with an ROC score of 0.99 on the train set and on the test set.
    - o The XGBOOST model is chosen for its simplicity, ease of interpretation compared to more complex models.

3. Model Evaluation:

We will use **sklearn.metrics.roc_auc_score** for this as AOC and ROC metric in sklearn is used as the metric for highly imbalanced data-set, rest all fails.

ROC have better false negative than the false positives.

ROC-Curve = Plot between TPR and FPR

The threshold with highest value for TPR-FPR on the train set is usually the best cut-off.

We **should not use the confusion matrix** as the performance metrics as well as they have internally defined hard threshold of 0.5.

We also **can’t completely rely on the precision, recall and F1-score** for now as they also have their strings attached of some threshold value.

ROC curve takes into cognizance of all the possible threshold values.

The **ROC curve** is used to understand the strength of the model by evaluating the performance of the model at all **the classification thresholds.**

Because the ROC curve is measured at all thresholds, the best threshold would be one at which the **TPR is high and FPR is low, i.e., misclassifications are low.**

4. Cost-Benefit Analysis:

Depending on the use case, we have to account for what we need: high precision or high recall.

For banks with smaller average transaction value, we would want high precision because we only want to label relevant transactions as fraudulent. For every transaction that is flagged as fraudulent, you can add the human element to verify whether the transaction was done by calling the customer. However, when precision is low, such tasks are a burden because the human element has to be increased.

For banks having a larger transaction value, if the recall is low, i.e., it is unable to detect transactions that are labelled as non-fraudulent. So consider the losses if the missed transaction was a high-value fraudulent one, for e.g., a transaction of $10,000?

So here, to **save banks from high-value fraudulent transactions, we have to focus on a high recall in order to detect actual fraudulent transactions.**

We need to determine how much profit or dollar/rupee value we are saving with our best selected model

5. Results and Insights Summary of Findings:

- Key insights from the data exploration phase are summarized, including any significant patterns or anomalies detected.

Feature Importance:

- Important features impacting the model's decisions are identified, providing insights into which factors are most indicative of fraud.

Model Performance:

- The final model's performance is reported on the test set, comparing it against baseline models and discussing any improvements made through tuning.
- The chosen XGBOOST model with Random Oversampling and StratifiedKFold CV balancing technique demonstrated high recall and a strong ROC-AUC score, making it a suitable choice for deployment.

6. Conclusion:

- The XGBOOST model with Random Oversampling and StratifiedKFold CV is chosen for its simplicity, high recall, and cost-effectiveness.
- This model is easier to interpret and explain to the business, making it a practical choice for deployment