**Credit Card Fraud Detection**

Introduction

Welcome to the Capstone Project on Credit Card Fraud Detection.

Although digital transactions in India registered a 51% growth in 2018-2019, their safety remains a concern. 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. Machine learning can play a vital role in detecting fraudulent transactions.

the data set includes credit card transactions made by European cardholders over a period of two days in September 2013. Out of a total of 2,84,807 transactions, 492 were fraudulent. This data set is highly unbalanced, with the positive class (frauds) accounting for just 0.172% of the total transactions. The data set has also been modified with Principal Component Analysis (PCA) to maintain confidentiality. Apart from ‘time’ and ‘amount’, all the other features (V1, V2, V3, up to V28) are the principal components obtained using PCA. The feature 'time' contains the seconds elapsed between the first transaction in the data set and the subsequent transactions. The feature 'amount' is the transaction amount. The feature 'class' represents class labelling, and it takes the value 1 in cases of fraud and 0 in others.

The distribution plots of the variables were Gaussian, which might indicate the effects of transformations that had already occurred on the data set.

When we are trying to predict the fraud there are a variety of variable which needs to be taken into account. To combat this problem PCA can be used to reduce the number of variables necessary to take into account the when analyzing the data. It created new variable or “Principal components” from linear transformation of the original variables.

Further, the data is highly imbalanced. Over 2,00,000 cases are mapped to 0, but hardly 500 cases are mapped to 1. Any machine learning algorithm would work well when there is equal representation of each of the classes. However, in this case, no matter which model is built, the underlying algorithm will learn more about the non-fraudulent cases rather than the fraudulent ones. Therefore, the loss function optimisation will be heavily biased to the former type of data. This is known as the ‘minority class problem’.

Now, we can use certain methods to mitigate this problem. They are as follows:

Under sampling: In this method, we have the choice of selecting fewer data points from the majority class for the model-building process. In case we have only 500 data points in the minority class, we will also have to take 500 data points from the majority class; this will make the classes somewhat balanced. However, in practice, this method is not effective because we will lose over 99% of the original data.

Oversampling: Using this method, we can assign weights to randomly chosen data points from the minority class. This way, the occurrence of each data point will be multiplied by the assigned weight, and the machine learning algorithm will now be able to focus on this class while optimising the loss function. However, this method does not add any new information and may even exaggerate the existing information quite a bit.

Synthetic Minority Over-Sampling Technique (SMOTE): In this process, you can generate new data points, which lie vectorially between two data points that belong to the minority class. These data points are randomly chosen and then assigned to the minority class. This method uses K-nearest neighbours to create random synthetic samples. The steps in this process are as follows:

Randomly selecting a minority point A

The k nearest neighbours for that data point belonging to the same are found and then a random point, B form the k\_neighbours is selected.

Specifying a random value in the range [0, 1] as

λ

.

Generating and placing a synthetic sample between the two points A and B on the vector located at

λ

% from the original point A.

ADAptive SYNthetic (ADASYN): This is similar to SMOTE, with a minor change in the generation of synthetic sample points for minority data points. For a particular data point, the number of synthetic samples that it will add will have a density distribution, whereas, for SMOTE, the distribution will be uniform. The aim here is to create synthetic data for minority examples that are harder to learn, rather than the easier ones. To sum it up, ADASYN offers the following advantages: It lowers the bias introduced by the class imbalance.

It adaptively shifts the classification decision boundary towards difficult examples.

KNN (K nearest neighbors) predictive algorithm is to be used

KNN: K-nearest neighbour is a simple, supervised machine learning algorithm used for both classification and regression tasks. It performs these tasks by identifying the neighbours that are nearest to a data point. For classification tasks, it takes the majority vote and for regression tasks, it takes the average value from the neighbours.

The k in KNN specifies the number of neighbours that the algorithm should focus on. For example, if k = 3, then, for a particular test data, the algorithm observes the three nearest neighbours and takes the majority vote from them. Depending on the majority of the classes from the three nearby points, the algorithm classifies the test data.