**Credit Card Fraud Detection using Machine Learning**

In this project, you will use Python, SMOTE Technique(to over-sample data), build a Logistic Regression Classifier, and apply it to detect if a transaction is fraudulent or not.

The real world datasets often might be with data of imbalanced classes. It is very important to feed a decent number of data samples of each class in a classification problem so that the classifier would detect the underlying hidden patterns for each class and prepare itself to reasonably classify the test data. Upon completing this project, you will understand the pragmatic application of various Pandas functions, with a clear picture of how to over-sample the dataset with imbalanced classes using the SMOTE technique and how to use the thus obtained data to train a classifier.

**Skills Used:**

1. Pandas
2. Python Programming
3. SMOTE
4. Scikit-Learn

**About project:**

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

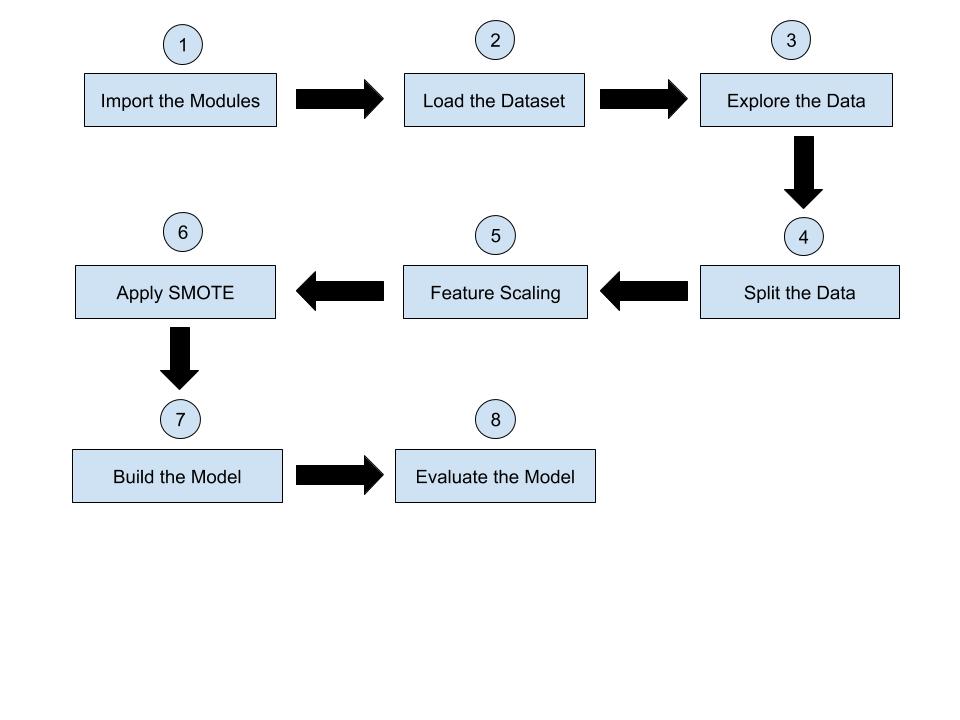
The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all 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-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group ([http://mlg.ulb.ac.be](http://mlg.ulb.ac.be/)) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. The dataset could be found in <https://www.kaggle.com/mlg-ulb/creditcardfraud>

More details on current and past projects on related topics are available on <https://www.researchgate.net/project/Fraud-detection-5> and the page of the DefeatFraud project

**Workflow of the project:**



**Understanding Class-Imbalance**

**Why don't we want class imbalance?**

* From our analysis, we observe there is a lot of imbalance in the classes, with most of the transactions were Non-Fraud (99.83%) of the time, while Fraud transactions occur (0.17%) of the time in the dataframe.
* Using this imbalanced data as such is not a good idea for training a model to classify if a transaction is fraudulent or not.
* This is because, if we use this imbalanced data is used to train a model, the algorithm does not have a decent amount of fraudulent-data to learn the patterns of fraudulent transactions. Thus, it most probably assumes that every transaction is non-fraudulent(the dominant class of the data).
* This would be a pity because the model naively assumes but doesn't learn/detect the patterns in order to classify.

To make the dataset balanced, we could either undersample or oversample it.

**Under-sampling:** In undersampling, we reduce the dataset such that the number of samples of one class is to that of the other class. But this method has a trade-off with the amount of information lost in the form of the samples removed.

**Over-sampling:** Next is the oversampling technique. We increase the number of total samples in the dataset by generating the synthetic samples for the minority class in order to achieve the balance between both the classes. The simplest approach involves duplicating examples in the minority class, although these examples don’t add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the Synthetic Minority Oversampling Technique, or **SMOTE** for short.

So to process this imbalance;

* We should do most pre-processing steps (splitting the data, normalization/standardization, etc) before under/over-sampling the data.
* This is because many sampling techniques require a simple model to be trained (e.g. SMOTE uses a k-NN algorithm to generate samples). These models have better performance on pre-processed datasets (e.g. both k-NN and k-means use euclidean distance, which requires the data to be normalized).
* So, in order for the sampling techniques to work best, we should previously perform any pre-processing steps we can. Then we shall proceed to use SMOTE technique to oversample the train data in order to use it to rain the classification algorithm.

### Get Confusion matrix and Recall

Let us predict the labels for train and test data, get the confusion matrix, and calculate the recall values.

**Note:**

* confusion\_matrix: computes confusion matrix to evaluate the accuracy of classification.
  + By definition, a confusion matrix C is such that Ci,j is equal to the number of observations known to be in the group i and predicted to be in group j.
  + Thus in binary classification, the count of true negatives is C00, false negatives is C10, true positives is C11 and false positives is C01.
* recall is calculated by (true positives)/(true positives + false negatives). Note that we are calculating recall value because we want to detect fraudulent credit card transactions. It might be tolerable to classify some valid transactions as fraudulent, but it is not tolerable to misclassify the fraudulent transactions as valid ones.

### ROC-AUC Curve

Let us now plot the ROC-AUC curve. The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR against FPR at various threshold values. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

**Note:**

* decision\_function predicts confidence scores for samples. The confidence score for a sample is the signed distance of that sample to the hyperplane. The advantage of Decision Function output is to set DECISION THRESHOLD and predict a new output for X\_test, such that we get the desired precision or recall value.
* roc\_curve computes ROC by taking true binary labels and confidence values, or non-thresholded measure of decisions as input arguments. It returns
  + increasing false-positive rates such that element i is the false positive rate of predictions with score >= thresholds[i] (fpr)
  + Increasing true positive rates such that element i is the true positive rate of predictions with score >= thresholds[i] (tpr)
  + Decreasing thresholds on the decision function used to compute fpr and tpr.

### Summary

* We have been given the Europe credit-card transaction data of 2 days. For privacy reasons, the personal details have been represented in the form of Principle Components. The Amount(the transaction Amount) and Time(the seconds elapsed between each transaction and the first transaction in the dataset) are also part of the columns other than the principal components. The transactions are of valid and fraudulent types. The goal is to build a classifier to detect fraudulent transactions.
* We have first loaded the data, explored it, and checked for any null values. While exploring, we found that the data is of high class-imbalance, with around 99.83% being valid transactions whereas about 0.17% are fraudulent.
* It is not a good idea to train a classifier with such highly imbalanced data as it leads to mere assumptions rather than learning by the algorithm. We could either undersample or oversample the data to achieve a balance between the class-wise data samples.
* We have split the data into train and test parts, in order to prevent any data leakage and to keep the test data untouched, before oversampling.
* We have scaled the Amount and Time features using StandardScaler.
* We then applied the SMOTE technique to oversample the train data and formed a new dataset with the thus obtained over-sampled datapoints.
* We used the GridSearch method with different parameter values, trained logistic regression classifiers with the different combinations of these parameters, and got the best logistic regression classifier which yields the least loss on the over-sampled data-set. All this mechanism is internally implemented by GridSearchCV of sklearn.
* We then used the best estimator thus obtained to evaluate its performance on the unseen test data. We calculated the recall, confusion-matrix and roc-auc scores.