BATCH-D AUGUST 2021

REGEX-TEAM 3 PROJECT REPORT

CREDIT CARD FRAUD DETECTION

Github link of the project

Regex Team-3:

- **Nikhil Shrestha: AIRSS1129**
- **Maneesh Arava: AIRSS1142**
- **Y. Manohar Reddy: AIRSS1139**
- **Furzana J: AIRSS1132**
- **Afzal Vali: AIRSS1138**

02

ABSTRACT

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 challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase.

CONSTRAINTS

Main challenges involved in credit card fraud detection are:

- Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
- Imbalanced Data i.e. most of the transactions (99.8%) are not fraudulent which makes it really hard for detecting the fraudulent ones
- Data availability as the data is mostly private.
- Miss-classified Data can be another major issue, as not every fraudulent transaction is caught and reported.
- Adaptive techniques used against the model by the scammers.
- How to tackle these challenges?
- The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.
- Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph
- For protecting the privacy of the user, the dimensionality of the data can be reduced.
- A more trustworthy source must be taken which double-checks the data, at least for training the model.
- We can make the model simple and interpretable so that when the scammer adapts to it with just some tweaks, we can have a new model up and running to deploy.

03

TOOLS USED


INITIAL APPROACH

- **Data Acquisition:** Collected data from research (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles).
- **Exploratory Data Analysis (EDA):**
  - Missing Value and Duplicates
  - Variable Distribution
  - Outlier Treatment
  - Correlation
  - Impact of Imbalanced data by training and comparing 3 models (Isolation Forest, Local Outlier Factor and SVM)
- **Preprocessing:**
  - Standardized Amount and Time variable
  - Using SMOTETomek to take care of unbalanced data.
- Used train_test_split with stratify parameter was to maintain the imbalanced proportion of data.
- Hyper-parameter tuning using **Optuna** - **Random Forest, XGBoost and MLPClassifier (NN)**. Evaluation used roc_auc_score
- Build Model using parameters obtained from Optuna hyper-parameter tuning.
- **Final model used: Voting Classifier with soft voting mode**, which considers the probabilities thrown by each ML model, these probabilities will be weighted and averaged, consequently the winning class will be the one with the highest weighted and averaged probability.

PROJECT ARCHITECTURE

DATA COLLECTION

The dataset has been collected from a research (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles).

EXPLORATORY DATA ANALYSIS

- Missing values
- Duplicates
- Variable distributions
- Correlation
- Skewness
- Outliers detection & treatment

IMBALANCED DATA

- Explained the impact of imbalanced data.
- We used SMOTETomek

CROSS VALIDATION

- Created 5 Folds using StratifiedKfold to maintain the proportion of dependent variable in all folds.

HYPER-TUNING MODELS

- Used Optuna which is a software framework for automating the optimization process of hyperparameters. Used Random Forest, XGBoost, MLPClassifier

MODELLING

- Trained respective models
- Final model was Voting classifier which is combination of 3 models created

EVALUATION

- Metrics used roc_auc_score during hyper-parameter tuning and training phase.
- Confusion matrix and classification report on test data during prediction phase.

CONCLUSION

- Summarization of project, insights and approach taken to get the best result.

04

CROSS VALIDATION

Cross-validation

- Cross-validation is a statistical method used to estimate the skill of machine learning models.
- It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.
- In this tutorial, you will discover a gentle introduction to the k-fold cross-validation procedure for estimating the skill of machine learning models.
- After completing this tutorial, you will know:
- That k-fold cross validation is a procedure used to estimate the skill of the model on new data.
- There are common tactics that you can use to select the value of k for your dataset.
- There are commonly used variations on cross-validation such as stratified and repeated that are available in scikit-learn.

k-Fold Cross-Validation

- k-Fold Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.
- The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.
- Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.
- It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

05

VARIATIONS ON CROSS-VALIDATION

- There are a number of variations on the k-fold cross validation procedure.
- Three commonly used variations are as follows:
  - **Train/Test Split:** Taken to one extreme, k may be set to 2 (not 1) such that a single train/test split is created to evaluate the model.
  - **LOOCV:** Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short.
  - **Stratified: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.**
  - **Repeated:** This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.
  - **Nested:** This is where k-fold cross-validation is performed within each fold of cross-validation, often to perform hyperparameter tuning during model evaluation. This is called nested cross-validation or double cross-validation.

06

MISSING VALUES

- Real-world data often has missing values.
- Data can have missing values for a number of reasons such as observations that were not recorded and data corruption.
- Handling missing data is important as many machine learning algorithms do not support data with missing values.
- **Missing Data can occur when no information is provided** for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenario. Missing Data can also refer to as NA (Not Available) values in pandas. In Data Frame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.
- **None**:None is a Python singleton object that is often used for missing data in Python code.
- **NaN**: NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation.
- Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas Data Frame.

07

OUTLIERS

- We will generally define **outliers as samples that are exceptionally far from the mainstream of the data.**
- Outliers are observations in a dataset that don't fit in some way.
- Perhaps the most common or familiar type of outlier is the observations that are far from the rest of the observations or the centre of mass of observations.
- This is easy to understand when we have one or two variables and we can visualize the data as a histogram or scatter plot, although it becomes very challenging when we have many input variables defining a high-dimensional input feature space.
- In this case, simple statistical methods for identifying outliers can break down, such as methods that use standard deviations or the interquartile range.
- It can be important to identify and remove outliers from data when training machine learning algorithms for predictive modelling.
- Outliers can skew statistical measures and data distributions, providing a misleading representation of the underlying data and relationships. Removing outliers from training data prior to modelling can result in a better fit of the data and, in turn, more skillful predictions.
- Thankfully, there are a variety of automatic model-based methods for identifying outliers in input data. Importantly, each method approaches the definition of an outlier is slightly different ways, providing alternate approaches to preparing a training dataset that can be evaluated and compared, just like any other data preparation step in a modelling pipeline.
- Before we dive into automatic outlier detection methods, let's first select a standard machine learning dataset that we can use as the basis for our investigation.

08

OUTLIERS . . .

Standard Deviation Method

- If we know that the distribution of values in the sample is Gaussian or Gaussian-like, we can use the standard deviation of the sample as a cut-off for identifying outliers.
- The Gaussian distribution has the property that the standard deviation from the mean can be used to reliably summarize the percentage of values in the sample.
- Given mu and sigma, a simple way to identify outliers is to compute a z-score for every xi, which is defined as the number of standard deviations away xi is from the mean. Data values that have a z-score sigma greater than a threshold.

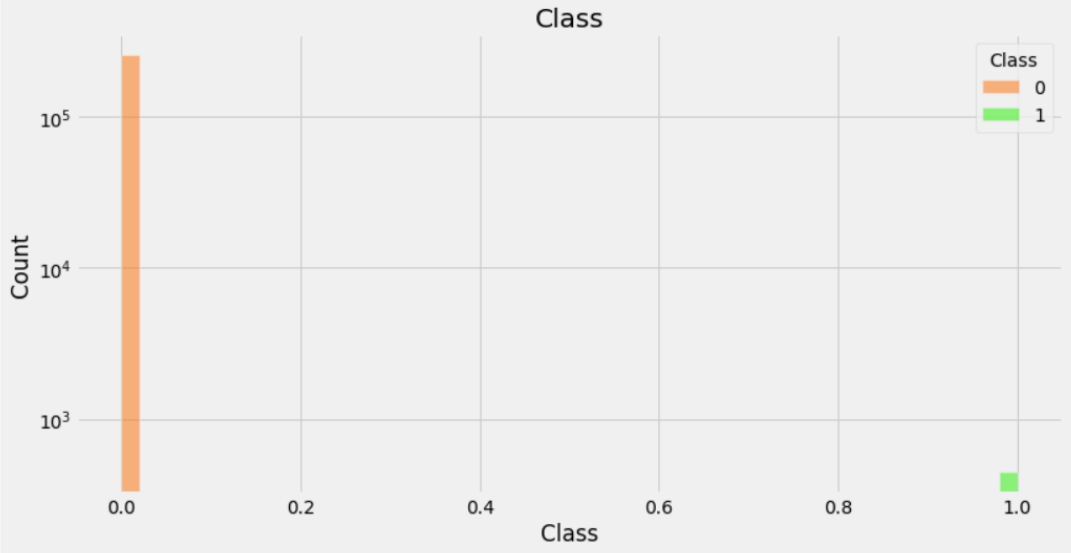
Interquartile Range Method

- Not all data is normal or normal enough to treat it as being drawn from a Gaussian distribution.
- A good statistic for summarizing a non-Gaussian distribution sample of data is the Interquartile Range, or IQR for short.
- The IQR is calculated as the difference between the 75th and the 25th percentiles of the data and defines the box in a box and whisker plot.
- The IQR defines the middle 50% of the data, or the body of the data.
- Statistics-based outlier detection techniques assume that the normal data points would appear in high probability regions of a stochastic model, while outliers would occur in the low probability regions of a stochastic model.
- On a box and whisker plot, these limits are drawn as fences on the whiskers (or the lines) that are drawn from the box. Values that fall outside of these values are drawn as dots.

09

IMBALANCED DATA

- The imbalanced dataset in real-world problems is not so rare. In layman terms, an imbalanced dataset is a dataset where classes are distributed unequally. An imbalanced data can create problems in the classification task. Before delving into the handling of imbalanced data, we should know the issues that an imbalanced dataset can create.
- The total legit transactions are 284315 out of 284807, which is 99.83%. The fraud transactions are only 492 in the whole dataset (0.17%).



Problems an imbalance dataset can create

- If we are using accuracy as a performance metric, it can create a huge problem. Let's say our model predicts each transaction as legal (dumb model). Using an accuracy metric on the credit card dataset will give 99.83% accuracy, which is excellent. **IS IT A GOOD RESULT? NO**
- For an imbalanced dataset, other performance metrics should be used, such as the Precision-Recall AUC score, F1 score, etc.. Moreover, the model will be biased towards the majority class. Since most machine learning techniques are designed to work well with a balanced dataset, we must create balanced data out of an imbalanced dataset.

10

IMPACT OF IMBALANCED DATA

- We checked the impact of imbalanced data using Three (3) different models.
  - Isolation Forest
  - One-Class SVM
  - Local Outlier Factor

- **Isolation Forest**, or iForest for short, is a tree-based anomaly detection algorithm. It is based on modelling the normal data in such a way as to isolate anomalies that are both few in number and different in the feature space.
  - This method takes advantage of two anomalies' quantitative properties:
  - They are the minority consisting of fewer instances and
  - They have attribute-values that are very different from those of normal instances.

- **ONE-CLASS SVM:** The support vector machine, or SVM, algorithm developed initially for binary classification can be used for one-class classification.
  - When modelling one class, the algorithm captures the density of the majority class and classifies examples on the extremes of the density function as outliers. This modification of SVM is referred to as One-Class SVM.
  - An algorithm that computes a binary function that is supposed to capture regions in input space where the probability density lives (its support), that is, a function such that most of the data will live in the region where the function is nonzero.

11

IMPACT OF IMBALANCED DATA. . .

- **LOCAL OUTLIER FACTOR:** A simple approach to identifying outliers is to locate those examples that are far from the other examples in the feature space.
  - This can work well for feature spaces with low dimensionality (few features), although it can become less reliable as the number of features is increased, referred to as the curse of dimensionality.
  - In machine learning, an approach to tackling the problem of outlier detection is one-class classification.
  - One-Class Classification, or OCC for short, involves fitting a model on the “normal” data and predicting whether new data is normal or an outlier/anomaly.
  - A one-class classifier aims at capturing characteristics of training instances, in order to be able to distinguish between them and potential outliers to appear.
  - A one-class classifier is fit on a training dataset that only has examples from the normal class. Once prepared, the model is used to classify new examples as either normal or not-normal, i.e. outliers or anomalies.
  - A simple approach to identifying outliers is to locate those examples that are far from the other examples in the feature space.
  - This can work well for feature spaces with low dimensionality (few features), although it can become less reliable as the number of features is increased, referred to as the curse of dimensionality.
  - The local outlier factor, or LOF for short, is a technique that attempts to harness the idea of nearest neighbors for outlier detection. Each example is assigned a scoring of how isolated or how likely it is to be outliers based on the size of its local neighborhood. Those examples with the largest score are more likely to be outliers.
  - We introduce a local outlier (LOF) for each object in the dataset, indicating its degree of outlier-ness.

12

IMPACT OF IMBALANCED DATA. . .

```
Isolation Forest: 41
Accuracy Score :
0.9983396776544909
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     24659
           1       0.42      0.43      0.42        35

    accuracy                           1.00     24694
   macro avg       0.71      0.71      0.71     24694
weighted avg       1.00      1.00      1.00     24694

Local Outlier Factor: 69
Accuracy Score :
0.9972057989795092
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     24659
           1       0.03      0.03      0.03        35

    accuracy                           1.00     24694
   macro avg       0.51      0.51      0.51     24694
weighted avg       1.00      1.00      1.00     24694

Support Vector Machine: 8657
Accuracy Score :
0.6494290110958127
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.65      0.79     24659
           1       0.00      0.31      0.00        35

    accuracy                           0.65     24694
   macro avg       0.50      0.48      0.39     24694
weighted avg       1.00      0.65      0.79     24694
```

- Isolation Forest detected 73 errors versus Local Outlier Factor detecting 97 errors vs. SVM detecting 20495 errors
- Isolation Forest has a 99.73% more accurate than LOF of 99.61% and SVM of 20.00
- When comparing error precision & recall for 3 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 27 % versus LOF detection rate of just 2 % and SVM of 0%.
- So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.
- We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

13

HANDLING IMBALANCED DATA

OVERSAMPLING AND UNDERSAMPLING:

- The main two methods that are used to tackle the class imbalance is upsampling/oversampling and downsampling/undersampling.
- The sampling process is applied only to the training set and no changes are made to the validation and testing data. Imblearn library in python comes in handy to achieve the data resampling.
- **Oversampling** is a procedure where synthetically generated data points (corresponding to minority class) are injected into the dataset. After this process, the counts of both labels are almost the same. This equalization procedure prevents the model from inclining towards the majority class.
- Furthermore, the interaction (boundary line)between the target classes remains unaltered. And also, the upsampling mechanism introduces bias into the system because of the additional information.

SMOTE(SyntheticMinorityOversamplingTechnique) — upsampling:-

- It works based on the KNearestNeighbours algorithm, synthetically generating data points that fall in the proximity of the already existing outnumbered group. The input records should not contain any null values when applying this approach.

```
#import imblearn library
from imblearn.over_sampling import SMOTENC
oversample = SMOTENC(categorical_features=[0,1,2,3,4,9,10],
random_state = 100)
X, y = oversample.fit_resample(X, y)
```

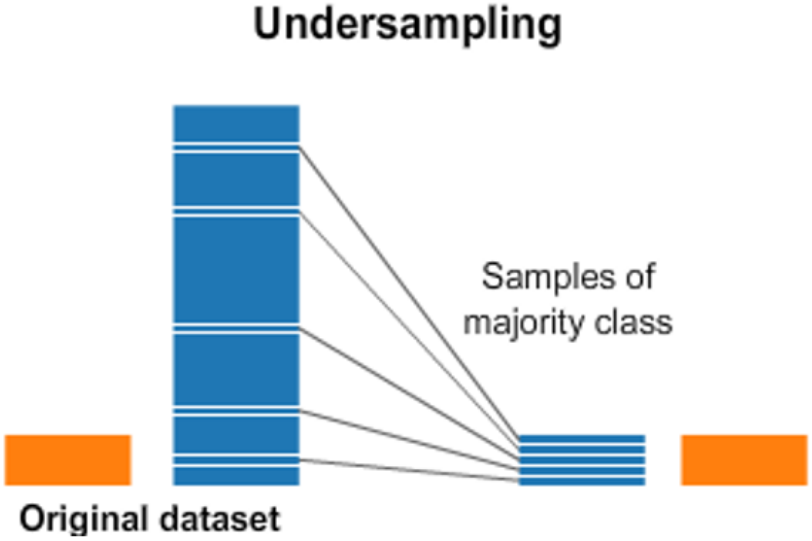
14

HANDLING IMBALANCED DATA. . .

DataDuplication – upsampling:- In this approach, the existing data points corresponding to the outvoted labels are randomly selected and duplicated.

```
from sklearn.utils import resample
maxcount = 332
train_nonnull_resampled = train_nonnull[0:0]
for grp in train_nonnull['Loan_Status'].unique():
   GrpDF = train_nonnull[train_nonnull['Loan_Status'] == grp]
   resampled = resample(GrpDF, replace=True, n_samples=int(maxcount),
random_state=123)
   train_nonnull_resampled = train_nonnull_resampled.append(resampled)
```

Undersampling is a mechanism that reduces the count of training samples falling under the majority class. As it helps to even up the counts of target categories. By removing the collected data, we tend to lose so much valuable information.

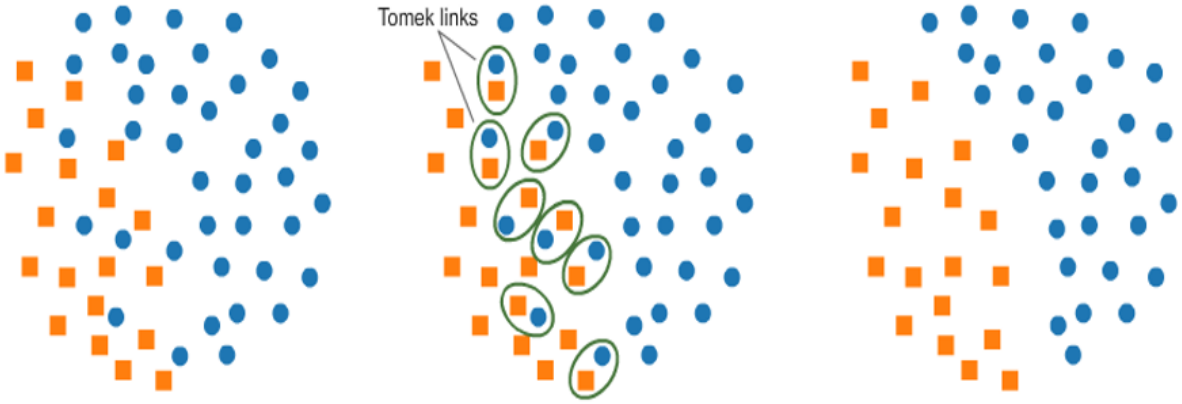


15

HANDLING IMBALANCED DATA. . .

Tomek(T-Links):-

- T-Link is basically a pair of data points from different classes(nearest-neighbors). The objective is to drop the sample that corresponds to the
- Majority and thereby minimalizing the count of the dominating label. This also increases the borderspace between the two labels and thus improving the performance accuracy.




```
from imblearn.under_sampling import TomekLinks
undersample = TomekLinks()
X, y = undersample.fit_resample(X, y)
```

Centroid Based:-

- The algorithm tries to find the homogenous clusters in the majority class and retains only the centroid. This would reduce the lion's share of the majority label. It leverages the logic used in the KMeans clustering. But a lot of useful information is wasted.

16

HANDLING IMBALANCED DATA. . .

SMOTETomek

- SMOTETomek is somewhere upsampling and downsampling.
- SMOTETomek is a hybrid method which is a mixture of the above two methods, it uses an under-sampling method (Tomek) with an oversampling method (SMOTE). This is present within imblearn.combine module.

```
from imblearn.combine import SMOTETomek
# Implementing Oversampling for Handling Imbalanced
smk = SMOTETomek(random_state=42)
X_res,y_res=smk.fit_sample(X,Y)
```

```
from collections import Counter
print(f'Original dataset shape {Counter(Y)}')
print(f'Resampled dataset shape {Counter(y_res)}')
```

```
Original dataset shape Counter({0: 246539, 1: 396})
Resampled dataset shape Counter({0: 245622, 1: 245622})
```

17

HYPER-PARAMETER TUNING

OPTUNA is an automatic hyperparameter optimization software framework, particularly designed for machine learning. It features an imperative, define-by-run style user API. Thanks to our define-by-run API, the code written with Optuna enjoys high modularity, and the user of Optuna can dynamically construct the search spaces for the hyperparameters.

- The different samplers available in optuna are as follows:
  - **Grid Search:** The search space of each hyper-parameter is discretized. The optimizer launches learning for each of the hyper-parameter configurations and selects the best at the end.
  - **Random:** Randomly samples the search space and continues until the stopping criteria are met.
  - **Bayesian:** Probabilistic model-based approach for finding the optimal hyperparameters
  - **Evolutionary algorithms:** Meta-heuristic approaches that employ the value of the fitness function to find the optimal hyperparameters.

Optuna has modern functionalities as follows:

- Lightweight, versatile, and platform agnostic architecture
- Handle a wide variety of tasks with a simple installation that has few requirements.
- Pythonic search spaces
- Define search spaces using familiar Python syntax including conditionals and loops.
- Efficient optimization algorithms
- Adopt state-of-the-art algorithms for sampling hyperparameters and efficiently pruning gloomy trials.
- Easy parallelization
- Scale studies to tens or hundreds or workers with little or no changes to the code.
- Quick visualization
- Inspect optimization histories from a variety of plotting functions.

18

HYPER-PARAMETER TUNING

```
def objective(trial):
  for fold, (train_indicies, valid_indicies) in enumerate(skf.split(X_train,y_train )): # creating Stratify-5-Folds and
    xtrain, ytrain = X_train.iloc[train_indicies], y_train.iloc[train_indicies]         # assigning xtrain, ytrain
    xvalid, yvalid = X_train.iloc[valid_indicies], y_train.iloc[valid_indicies]         # assigning xvalid, yvalid

    xtrain = xtrain[useful_features]                                                    # removed Class variable
    xvalid = xvalid[useful_features]                                                    # removed Class variable


    # Modelling
    params =   {'n_estimators' : trial.suggest_int('n_estimators', 10, 20)
    ,'max_depth' : trial.suggest_int('max_depth', 3, 10)
    }
    model = RandomForestClassifier(**params,criterion = 'entropy',random_state= 5)# Initialization of RandomForestClassifier Class
    model.fit(xtrain,ytrain)                                                        # Training the Model on training set

    # Predictions and Evaluation
    preds_valid = model.predict_proba(xvalid)[:, 1]                                 # Validating the model on Validation data.
    roc_auc_score = metrics.roc_auc_score(yvalid, preds_valid)                      # Evaluating the model using roc_aucc_curve
    print(fold, roc_auc_score)

  return roc_auc_score
study_rf = optuna.create_study(direction='maximize')                                # direction = "maximize", optuna will try to maximize the roc_auc_score
study_rf.optimize(objective, n_trials=15)                                           # Running our objective function for 15 trials.
```

Hypertuning using Optuna (Random Forest).

- We split the data using stratifykfold.
- Next we setup the parameters for RandomForest
- Finally prediction was done on validation set and roc_auc_score was the metrics used.
- Study was created to maximize roc_auc_score and trials was set to 15.
- To summarize we had 15 trials and each trial had 5 folds therefore we trained 75 models to obtain best combination of parameters also considering the folds we will be using to train the final model.
- Similarly multiple models were trained and best parameters were obtained respectively.

19

MODEL-BUILDING

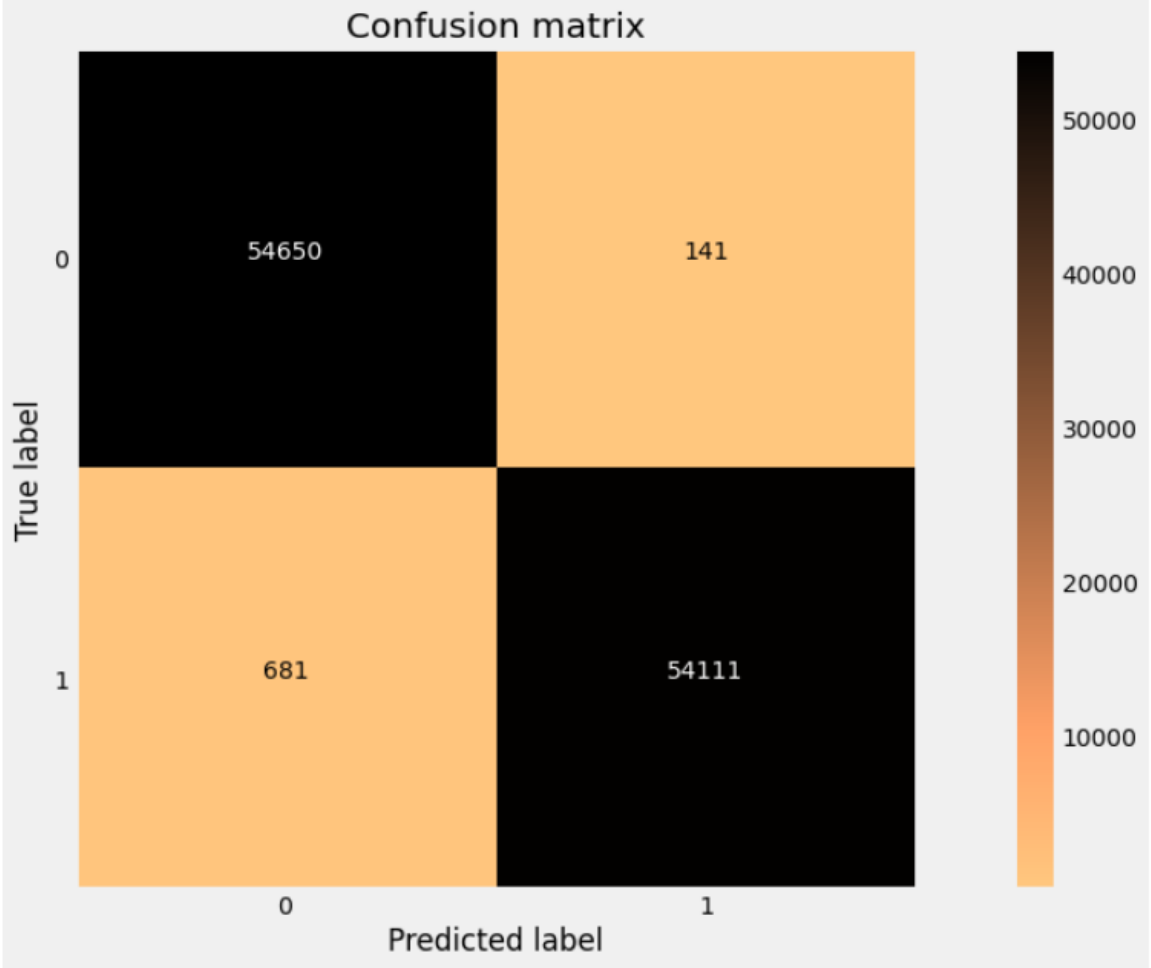
```
useful_features = [c for c in df_train.columns if c not in ("Class")]

final_test_predictions_rf = [] #to store final test predictions
final_valid_predictions_rf = [] # to store final validation predictions
scores = [] #to store the scores
xtest, ytest = df_test.copy(), df_test['Class'].copy()


for fold, (train_indicies, valid_indicies) in enumerate(skf.split(X_train,y_train )):   # creating Stratify-5-Folds
    xtrain, ytrain = X_train.iloc[train_indicies], y_train.iloc[train_indicies]         # creating xtrain, ytrain
    xvalid, yvalid = X_train.iloc[valid_indicies], y_train.iloc[valid_indicies]         # creating xvalid and yvalid


    xtrain = xtrain[useful_features]                                                    # Removing Class variable
    xvalid = xvalid[useful_features]                                                    # Removing Class variable
    xtest = xtest[useful_features]                                                      # Removing Class variable

    # Model building using parameters obtain from Optuna hyper-parameter tuning
    params = {'n_estimators': 16, 'max_depth': 10}
    model_rf = RandomForestClassifier(**params,criterion = 'entropy',random_state= 7)   # instantiate RandomForestClassifier Class
    model_rf.fit(xtrain, ytrain)                                                        # Training the model on training set

    preds_valid = model_rf.predict_proba(xvalid)[:, 1]                                  # Predicting Validation set
    test_preds = model_rf.predict_proba(xtest)[:, 1]                                    # Predicting Test Set
    final_test_predictions_rf.append(test_preds)                                        # Appending test predictions to list
    final_valid_predictions_rf.append(preds_valid)                                      # Appending valid predictions to list
    roc_auc_score_valid = metrics.roc_auc_score(yvalid, preds_valid)                    # Evaluating valid predictions using roc_auc_curve
    roc_auc_score_test = metrics.roc_auc_score(ytest, test_preds)                       # Evaluating test predictions using roc_auc_curve
    print(f'Fold {fold} AUC_valid: ', roc_auc_score_valid)                              # printing fold and respective roc_auc for validation set
    print(f'Fold {fold} AUC_test: ', roc_auc_score_test)                                # printing fold and respective roc_auc for test set
    scores.append(roc_auc_score_valid)                                                  # Appending roc_auc_score to list.

print(np.mean(scores), np.std(scores))                                                  # printing mean and standard deviation of scores
```

Model Building (Random Forest)

- We split the data using stratifykfold
- Next we used the parameters from Optuna and instantiate RandomForestClassifier class.
- Finally prediction was done on validation set and test set. Evaluation metrics used was roc_auc_score.
- Results of each iteration was stored in respective lists.
- Mean and Standard Deviation of scores was also printed and it shows that model performed very nice.
- Similarly multiple models (Xgboost and MLPClassifier) were trained using the best parameters obtained from Optuna hyper-parameter tuning.

20

MODEL-EVALUATION



	precision	recall	f1-score	support
0	0.99	1.00	0.99	54791
1	1.00	0.99	0.99	54792
accuracy			0.99	109583
macro avg	0.99	0.99	0.99	109583
weighted avg	0.99	0.99	0.99	109583

Model Evaluation

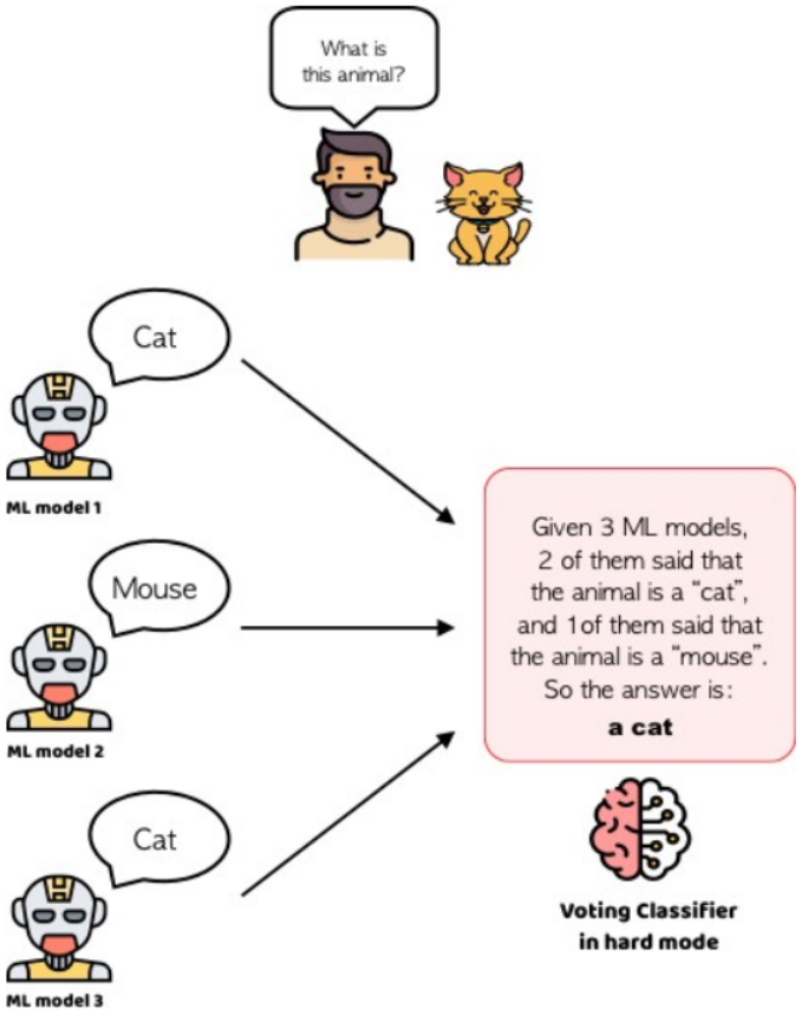
- Every model is evaluated using Confusion matrix and Classification report using test_data
- We got accuracy = 0.99, F-1 score = 0.99
- Also we can see that model made very low amount of type-1 and type-2 error.
- Similarly all the models are evaluated.

21

FINAL MODEL - VOTING CLASSIFIER

Voting Classifier

- This type of ensemble is one of the most intuitive and easy to understand. The Voting Classifier is a homogeneous and heterogeneous type of Ensemble Learning, that is, the base classifiers can be of the same or different type. As mentioned earlier, this type of ensemble also works as an extension of bagging (e.g. Random Forest).
- The architecture of a Voting Classifier is made up of a number “n” of ML models, whose predictions are valued in two different ways: hard and soft. In hard mode, the winning prediction is the one with “the most votes”. In Figure 2 we see an example of how the Voting Classifier works in hard mode.

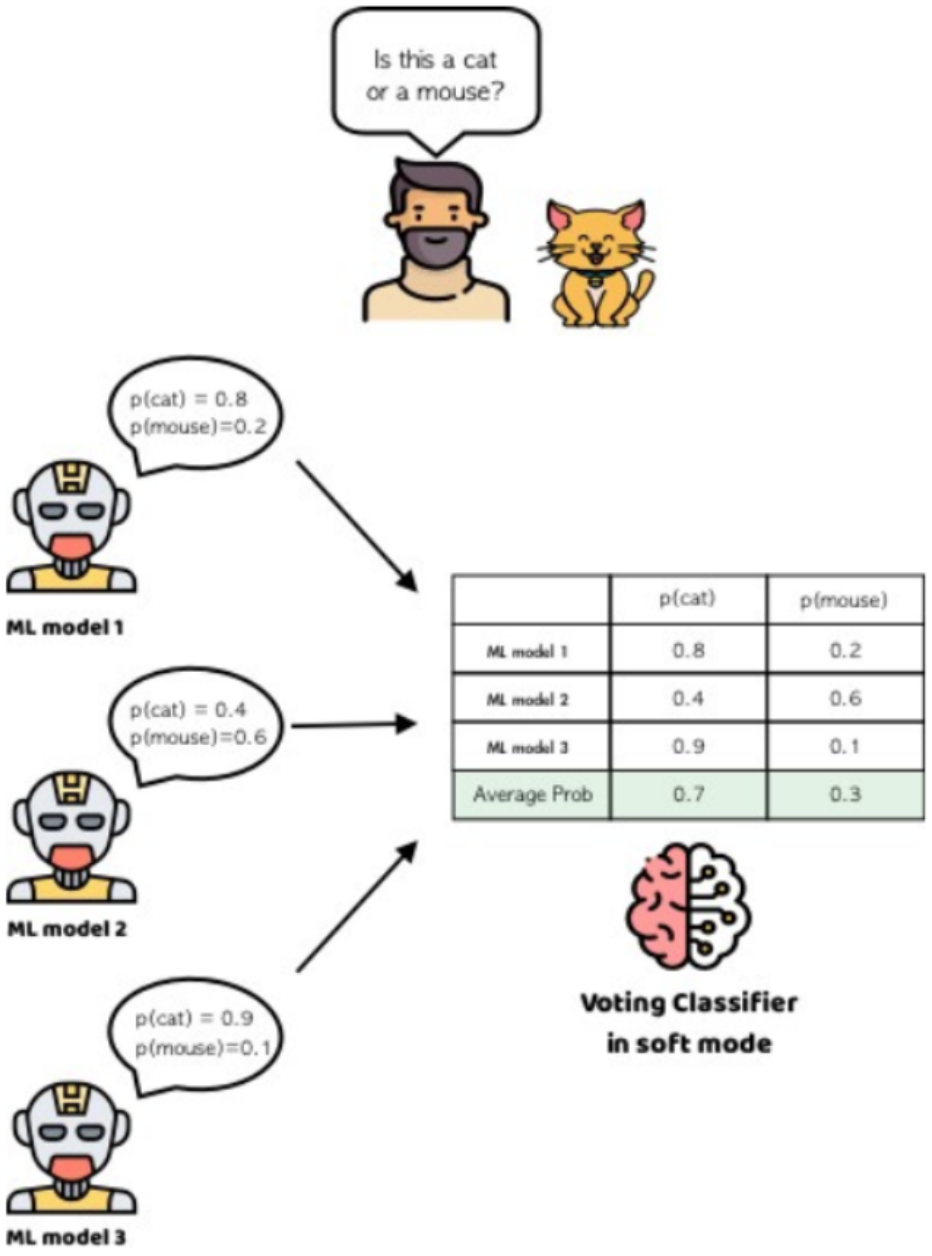


22

FINAL MODEL - VOTING CLASSIFIER. .

Voting Classifier

- On the other hand, the Voting Classifier in soft mode considers the probabilities thrown by each ML model, these probabilities will be weighted and averaged, consequently the winning class will be the one with the highest weighted and averaged probability. In Figure 3 we see an example of how the Voting Classifier works in the soft mode.



23

FINAL MODEL- VOTING CLASSIFIER..

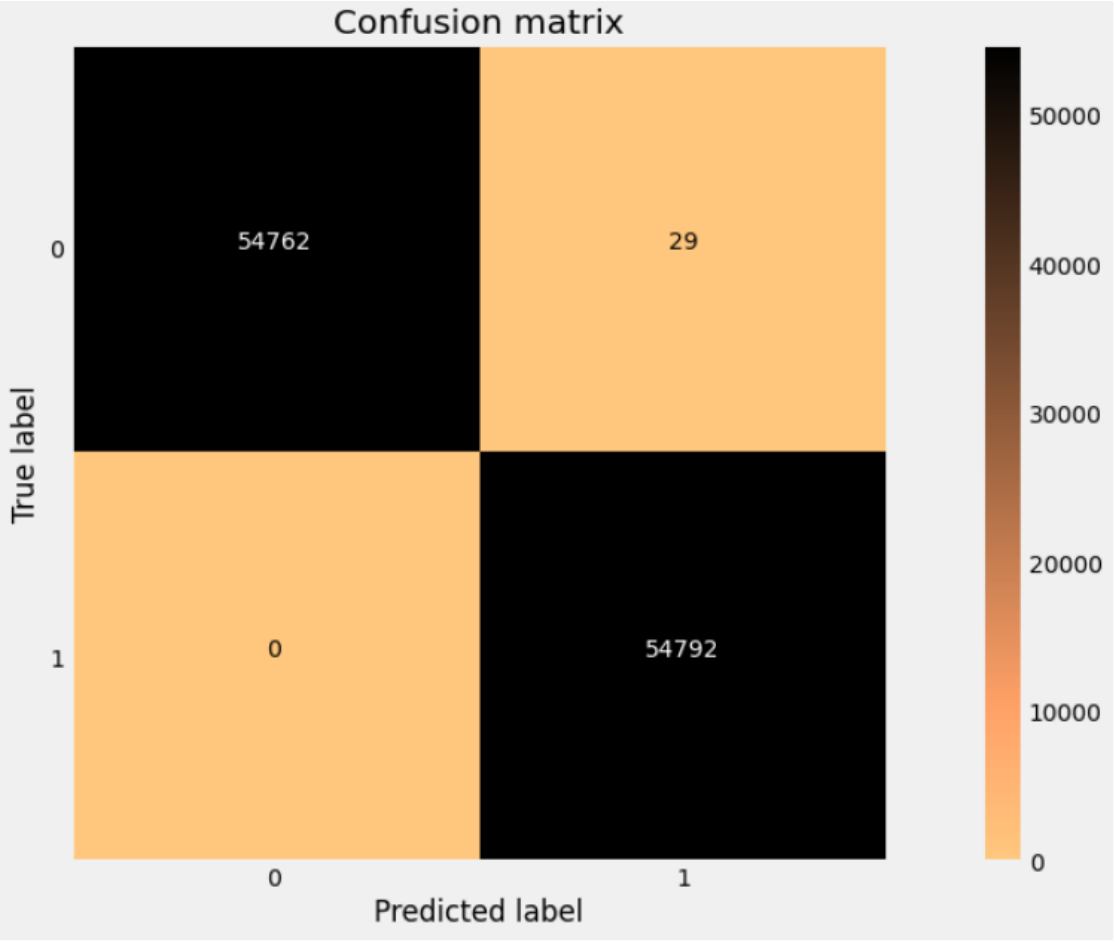
```
%%time
useful_features = [c for c in df_train.columns if c not in ("Class")]

final_test_predictions_vclf = [] #to store final test predictions
final_valid_predictions_vclf = [] # to store final validation predictions
scores = [] #to store the scores

for fold, (train_indicies, valid_indicies) in enumerate(skf.split(X_train,y_train )): # Creating Strtify-5-Folds
    xtrain, ytrain = X_train.iloc[train_indicies], y_train.iloc[train_indicies]       # Creating xtrain and ytrain
    xvalid, yvalid = X_train.iloc[valid_indicies], y_train.iloc[valid_indicies]       # Creating xvalid and yvalid
    xtest, ytest = df_test.copy(), df_test['Class'].copy()                            # Creating xtest and ytest

    # Removing Class varaibles from xtrain, xvalid and xtest
    xtrain = xtrain[useful_features].values
    xvalid = xvalid[useful_features].values
    xtest = xtest[useful_features].values

    model_vclf = VotingClassifier(estimators=[ ('ANN', model_mp1),                    # Passing first model- MLPClassifier
                                               ('XGBoostClassifier', model_xgb1),     # Passing second model - XGBoost
                                               ('RandomForestClassifier', model_rf1)], # Passing third model - RandomForestClassifier
                                  voting='soft')                                      # Used Soft voting
    model_vclf.fit(xtrain, ytrain)                                                    # Training the model

    preds_valid = model_vclf.predict_proba(xvalid)[:, 1]                              # Prediction on validation data
    test_preds = model_vclf.predict_proba(xtest)[:, 1]                                # Prediction on test data
    final_test_predictions_vclf.append(test_preds)                                    # Appending test predictions to list
    final_valid_predictions_vclf.append(preds_valid)                                  # Appending validation predictions to list
    roc_auc_score_valid = metrics.roc_auc_score(yvalid, preds_valid)                  # Evaluating validation predictions using roc_auc_score
    roc_auc_score_test = metrics.roc_auc_score(ytest, test_preds)                     # Evaluating Test predictions using roc_auc_score
    print(f'Fold {fold} AUC_valid: ', roc_auc_score_valid)                            # Printing fold and AUC_validation
    print(f'Fold {fold} AUC_test: ', roc_auc_score_test)                              # Printing fold and AUC_test
    scores.append(roc_auc_score_valid)                                                # Appending validation score to list


print(np.mean(scores), np.std(scores))                                                # Printing Mean and Standard Deviation of Scores.
```

Final model- Voting Classifier

- We split the data using stratifykfold
- Next we passed three models (RandomForest, XGBoost and MLPClassifier) as parameter to Voting Classifier and used soft voting technique.
- Finally prediction was done on validation set and test set. Evaluation metrics used was roc_auc_score.
- Results of each iteration was stored in respective lists.
- Mean and Standard Deviation of scores was also printed and it shows that model performed very nice.
- Model was further assessed using Confusion Matrix and Classification report.

24

EVAULATION- VOTING CLASSIFIER..



	precision	recall	f1-score	support
0	1.00	1.00	1.00	54791
1	1.00	1.00	1.00	54792
accuracy			1.00	109583
macro avg	1.00	1.00	1.00	109583
weighted avg	1.00	1.00	1.00	109583

Model Evaluation

- Every model is evaluated using Confusion matrix and Classification report using TEST_DATA
- We got accuracy = 100%, F-1 score = 100%
- Also we can see that model made very low amount of type-1 and type-2 error.

24

CONCLUSION

- We started with given data which was unbalanced
- Challenge was to balance the dataset which was achieved by using SMOTETomek. Impact of unbalanced data was explained by training and comparing 3 different models using sample of data.
- Standardization of variables was required as Amount and Time variables had huge values which could have degraded the resultant scores.
- Outliers were handled using capping and flooring technique and same was visualized in the notebook.
- We hyper-tuned 3 models using Optuna:
  - Random Forest
  - Xgboost
  - MLPClassifier (Neural Network)
- These hyper-tuned parameters were used to build respective models and every model was evaluated separately using:
  - roc_auc_score during hyper-parameter tuning and training phase
  - confusion_matrix and classification_report on test data during prediction phase.
- Finally these 3 models (RF, XGB and MLP) were passed to Voting Classifier with soft-voting technique and final predictions were made.
- Evaulation of Voting Classifier:
  - roc_auc_score during training phase
  - confusion_matrix and classification_report on test data during prediction phase.

REFERENCE

- https://imbalanced-learn.org/dev/references/generated/imblearn.combine.SMOTETomek.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- https://optuna.org/
- https://www.regexsoftware.com/
- https://www.kaggle.com/mlg-ulb/creditcardfraud
- https://mlg.ulb.ac.be/wordpress/
- https://www.researchgate.net/project/Fraud-detection-with-machine-learning