Required Libraries:

I am using regular libraries like

- NUMPY
- PANDAS
- MATPLOTLIB
- SEABORN
- SKLEARN

For Data Collection, EDA, Model Building, Visualization etc.,

Apart from that I am using

- Ydata Profiling
- Sweetviz
- Pycaret

For deep analysis and visualization and Model selection.

Data Collection:

The first thing we need to do is collecting the data.

A ‘csv’ file named ‘credcardfault’(given) was uploaded in python platform using panda’s library.

The data contains 284807 rows and 31 columns.

Since the data is huge, I am using google colab instead of my local machine to avoid overload.

The following are the columns/features from given data:

1. 'Time': Time of Transaction
2. 'V1-V28': Features of Transaction
3. 'Amount': Amount of Transaction
4. 'Class': Type of Transaction-Normal/Fraud

The datatypes of all the features in data are in the float except for ‘Class’ which is our target feature.

Data Cleaning:

Luckily, the data has no null values, but has duplicated values.

After dropping the duplicated values, we now have 283726 rows.

Data Analysis:

We can easily analyse the data using ydata_profiling.

The following are the columns/features from given data:

1. 'Time': Time of Transaction

2. 'V1-V28': Transaction details
3. 'Amount': Amount of Transaction
4. 'Class': Type of Transaction-Normal/Fraud

The datatypes of all the features in data are in the float except for ‘Class’ which is our target feature.

The ‘Class’ is the classification feature which contains 0 and 1.

Through Profile Report from ydata_profiling, we found that:

- Amount is highly correlated with V2
- V21 and V22 are highly correlated with each other
- Class column/feature is highly imbalance

There are more than 99 % Normal Transactions and very minute percentage of 0.16% are Fraud Transactions.

Fraud transaction occurring only for little amount, as I am guessing to avoid being caught, since people don’t take much care for minute amount.

Feature Engineering:

***Correcting the Time Column**:*

The feature ‘Time’ contains the seconds elapsed between the first transaction in the dataset and subsequent transactions.

As Time is given in relative fashion, we are using pandas.timedelta which represents duration, the difference between two time and dates.

After correcting the ‘Time’ column in an hour format we drop the ‘Time’ column since we no longer needing it.

Now we are left with features: ['V1'- 'V28', 'Amount', 'Class', 'Time Hour']

Correlations:

Positive Correlation:

- Amount and V7
- Amount and V12
- Amount and V20

Negative Correlation:

- Amount and V2
- Amount and V5
- Amount and V17

Model building:

Firstly, I separated the data into train and test sets in 80:20 ratio. And then created a common function for confusion matrix which visualizes for all models.

Now let's create an empty data frame with column names methodology, model accuracy, ROC value and threshold. This will be required to store all the results that we will be creating.

So, we will be creating many models using different strategies or different methodologies. So, this data frame results will store all the results for all the models that we are going to create which will help us to compare at the end that which model is performing better than the other.

The models we are creating and appending the results in Data frame is:

- Logistic Regression
- KNN
- Decision Trees
- Random Forest
- SVM
- XGBOOST

With StratifiedKFold Cross Validation technique, Logistic Regression with L2 Regularisation given the best results with 99% accuracy.

Since the target feature is highly imbalanced, we cannot rely on model with out handling imbalance.

Handling Imbalance:

We can treat Imbalance data with:

- Random Oversampling
- SMOTE
- ADASYN

After performing all kind of methods, it seems XGBOOST model with Random Oversampling with StratifiedKFold CV has provided the best results under the category of all oversampling techniques.

And then after tuning the hyperparameters of this model to get best results.

We end up with following results:

Model Accuracy: 0.9993655828707375

XGboost roc_value: 0.9871069819296697

XGBoost threshold: 0.017604010179638863

Feature Importance:

Features like V14, V10, V12 showed highest importance in model building of XGBOOST.

Using PyCaret:

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that exponentially speeds up the experiment cycle and makes you more productive.

PyCaret is a library which will choose the best model, for data with the imbalance features.

So, through Pycaret, I compared the all models within a single command

compare_models()

This function trains and evaluates the performance of all the models available in the model library using cross-validation.

The output of this function is a scoring grid with average cross-validated scores.

It gave us the Accuracy, AUC, Recall, Precision, F1, Kappa, MCC scores for all classification

For imbalanced output, we check F1 score and Kappa score.

Out of all models Extreme boosting (XG Boost) Model has high F1 score and Kappa score.

So, we are creating XGBOOST

This function trains and evaluates the performance of a given model using cross-validation.

The output of this function is a scoring grid with cross-validated scores along with mean and standard deviation.

Hyperparameter Tuning:

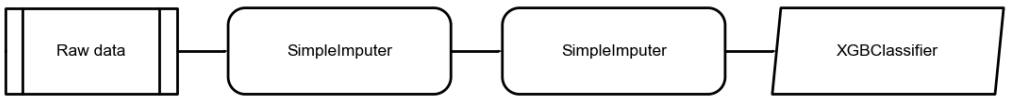
The output of this function is a scoring grid with cross-validated scores of the best model.

Search spaces are pre-defined with the flexibility to provide our own.

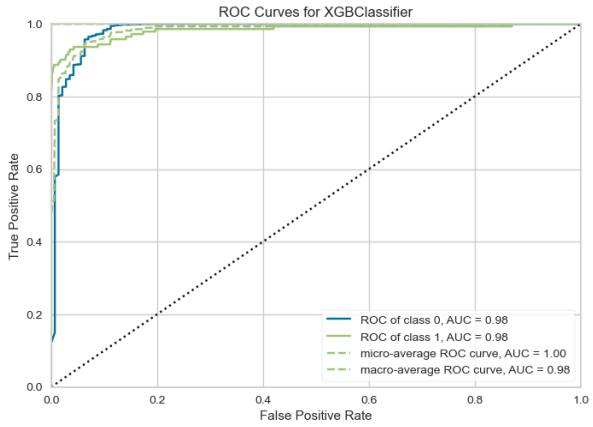
Hyperparameter tuning done by fitting 10 folds for each of 10 candidates, totalling 100 fits.

Then it has been observed that original model was better than the tuned model, hence it will be returned.

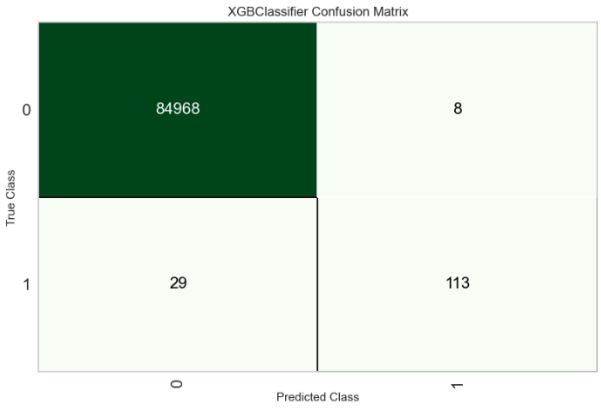
Pipeline Plot:



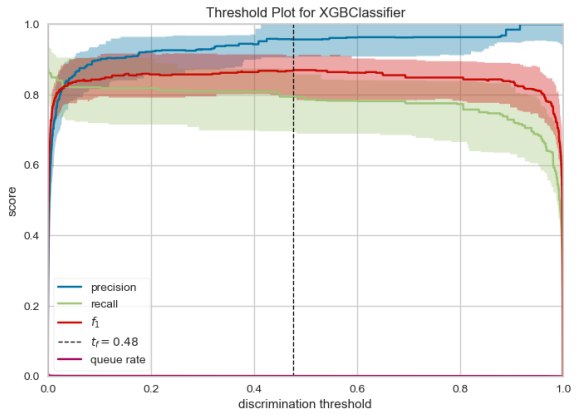
AUC Curve:



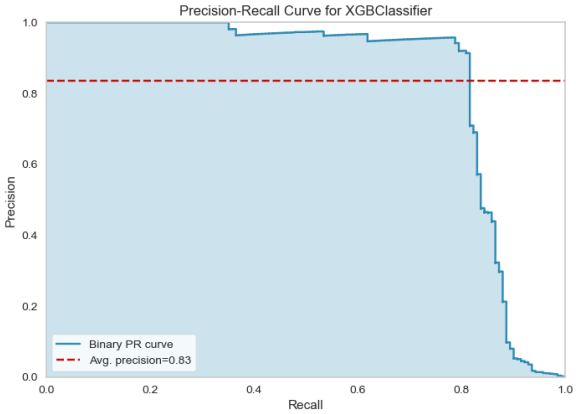
Confusion Matrix:



Threshold Plot:



Precision Recall Curve:

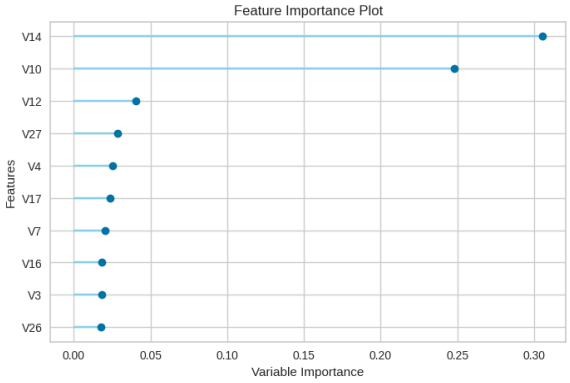


Hyper Parameter Tuning:

After Fitting 10 folds for each of 10 candidates, totalling 100 fits.

Original model was better than the tuned model.

Feature Importance:



Conclusion:

In the oversample cases, of all the models we build found that the XGBOOST model with Random Oversampling with StratifiedKFold CV gave us the best accuracy and ROC on oversampled data. Post that we performed hyperparameter tuning and got the below metrices:

XGboost roc_value: 0.9815403079438694

XGBoost threshold: 0.01721232570707798

However, of all the models we created we found Logistic Regression with L2 Regularisation for StratifiedKFold cross validation (without any oversampling or undersampling) gave us the best result.

Using Pycaret Library also, we found that XGBOOST is the best model for this data set.

F1 Score: 0.8432

Kappa Score: 0.8430

Deployment Plan:

PyCaret also provide a function that deploys the transformation pipeline and trained model on cloud.

To deploy our trained model into amazon, we use below command:

```
deploy_model(model = xgboost, model_name = 'xgb-for-deployment', platform = 'aws',
                    authentication = {'bucket' : 'S3-bucket-name'})
```

To deploy a model on AWS S3 (‘aws’), the credentials have to be passed. The easiest way is to use environment variables in our local environment. Following information from the IAM portal of amazon console account are required:

AWS Access Key ID

AWS Secret Key Access