Project Name:

CAPSTONE PROJECT

Project Title:

BYDEFAULT-PREDICTION OF CREDIT CARD FRAUD

For the

Data Scientist Bootcamp Learner for Certification

Prepared by:

Emmanuel Daniel Chonza

November 2023

Table of Contents

Introduction

In January 2023 I registered for the Data Scientist Bootcamp training at Knowledgehut upGrad aimed at learning and meeting the minimum requirements for being certified Data Scientist. The course was executed in terms of self-paced videos and resources available on PRISM while at same time undertaking live sessions every weekend. The training process reached an end with a requirement that every learner should undertake a project of choice among the project ideas provided by the capstone project guide.

Objective of the paper

It is based on this background, the project 'bydefault-prediction of credit card fraud', which this paper is aimed at documenting. Hence, the project is entirely a demonstration of how the learning process has supported my personal journey providing with key Data Science skills set.

Data Science Project

As a fulfilment for the certification of Data Scientist, it is mandatory for the learner to run a Data Science related project aimed at demonstrating practical use of the learned skills taking a real-world problem. e understanding of key issues the course has provided. It in this

Problem Statement:

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash. It is important that credit card companies recognize fraudulent credit card transactions so that customers are not charged for items they did not purchase.

Data Source and Dataset Description:

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where 492 frauds out of 284,807 transactions were identified.

The dataset contain 30 features which cover the credit card transactions, which this project should interrogate in establishing patterns and relationship amongst and how they relate the target variable. Below are features:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Predictor features are: Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, V22, V23, V24, V25, V26, V27, V28, and Amount while the target feature being Class. The Class is a binary feature where fraudulent transactions are represented by class 1 while the clean transactions are represented by class 0.

Existence of Imbalanced Dataset:

Only 492 frauds out of 284,807 transactions recorded in two days shows highly unbalanced class values, the positive class (frauds) account for only 0.172% of all transactions. That means 99.83% were clean transactions.

Objective of the Capstone Project:

To build a Machine Learning classification model, which will provide a Classifier to predict whether a transaction is fraudulent or not.

Project Methodology:

To Accomplish the Project below are the Methodological Steps to Follow:

- Exploratory Data Analysis: Analyze and understand the data to identify patterns, relationships, and trends in the data by using Descriptive Statistics and Visualizations.
- Data Cleaning: This might include standardization, handling the missing values and outliers in the data.
- Dealing with Imbalanced data: This data set is highly imbalanced. The data should be balanced using the appropriate methods before moving onto model building.
- Feature Engineering: Create new features or transform the existing features for better performance of the ML Models.
- Model Selection: Choose the most appropriate model that can be used for this project.
- Model Training: Split the data into train & test sets and use the train set to estimate the best model parameters.
- Model Validation: Evaluate the performance of the model on data that was not used during the training process. The goal is to estimate the model's ability to generalize to new, unseen data and to identify any issues with the model, such as overfitting.
- Model Deployment: Model deployment is the process of making a trained machine learning model available for use in a production environment.

Loading of Dataset

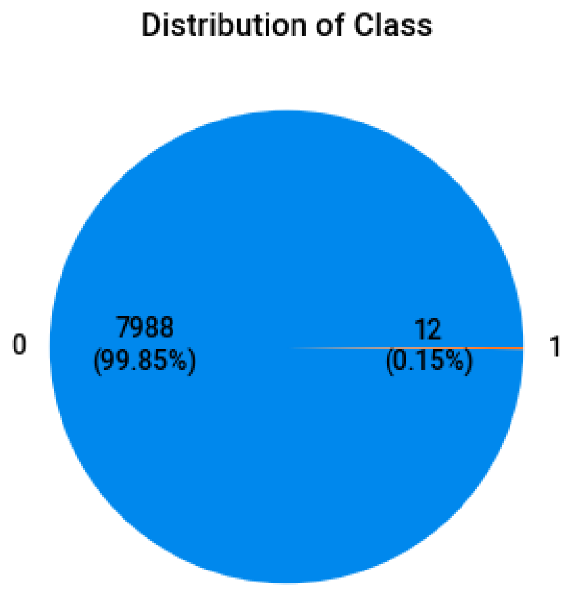
Different python libraries were used to support dataset loading and perform Exploratory Data Analysis (EDA) conveniently. The libraries are as below:

- import pandas as pd.
- import seaborn as sns.
- from matplotlib import pyplot as plt.
- import numpy as np.
- %matplotlib inline.

Loading datasets from the locally on the machine to the Jupyter Notebook was done using the pd.read_csv() method of the imported pandas (pd) inserting the file path stored as df.

Handling the Imbalanced Dataset

The imbalanced datasets pose a common challenge for machine learning practitioners in binary classification problems. This scenario frequently arises in practical business applications like fraud and spam filtering, rare disease discovery, and hardware fault detection. Class imbalance occurs when one class of the target variable has significantly fewer samples compared to another class, which can lead to a biased or poorly performing model. Therefore, it is obvious that it is clear how important that this challenge should be dealt with properly for efficient and effective model performance. Below is the distribution of target feature classes before imbalance dataset is dealt with.



Resampling data is one of the most preferred approaches to deal with an imbalanced dataset. There are broadly two types of methods for this i) Undersampling ii) Oversampling. In most cases, oversampling is preferred over undersampling techniques. The reason being, in undersampling the technique tends to remove instances from data that may be carrying some important information. While undersampling and oversampling are first place techniques, other mixed techniques are adopted by Data Scientist in finding better ways of dataset resampling for improved machine learning models' performance.

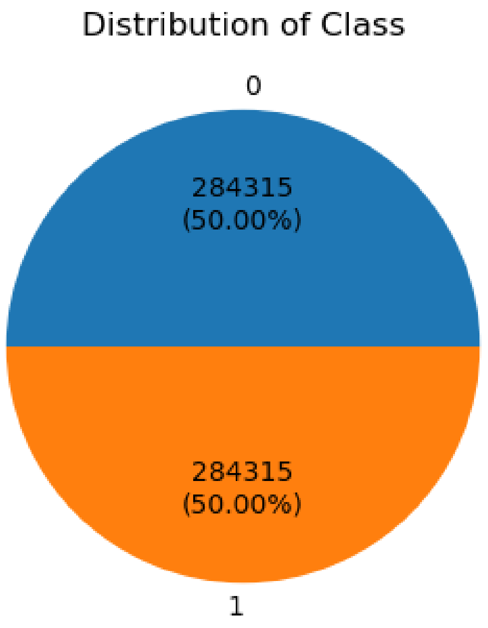
The Synthetic Minority Oversampling Technique (SMOTE) is one of those techniques that are somewhat better compared to the undersampling and oversampling techniques. SMOTE is specifically designed to tackle imbalanced datasets by generating synthetic samples for the minority class, hence adopted for this project as the sole technique to resolve the underlying dataset imbalance present for the following reasons:

- Identify the Minority Class: In a dataset with class imbalance, SMOTE focuses on the minority class.
- Generate Synthetic Samples: SMOTE works by generating synthetic samples for the minority class. It does this by selecting a sample from the minority class and finding its k-nearest neighbors within that class.
- Interpolation: Once the nearest neighbors are identified, SMOTE creates synthetic samples by interpolating between the selected sample and its neighbors. The synthetic samples are created by selecting a fraction of the distance between the selected sample and each of its neighbors.
- Balance the Classes: By generating synthetic samples for the minority class, SMOTE helps balance the class distribution. This makes the dataset more balanced and prevents the model from being biased toward the majority class.
- After Experimenting both the Undersampling and OverSampling Methods/techniques, it turned out that SMOTE was working well on the model performance, hence chosen.

SMOTE Approach:

Imported from the sklearn.over_sampling, SMOTE is applied to resample to variable split of X, y of the DataFrame using the SMOTE(random_state='n').fit_resample(X, y) method.

After applying the SMOTE, the resempled variables are concatenated using the pandas pd.concat() method to recover back the split DataFrame. Below is the distribution of target feature classes after resampling was applied.



Performing Exploratory Data Analysis (EDA)

The features' patterns and correlation were plotted being part of exploration Data Analysis (EDA) using the library 'sweetviz' imported using 'import sweetviz as sv', a 'report' was created using the sv.analyze() method from the created DataFrame, 'df'.

While the report above is comprehensive, individual codes written separately demonstrate how further checking of presence of null values, unique values, and duplicates could be performed. Note that given the nature of business the dataset – credit card transactions; duplicates were retained as each feature counts for prevalence of fraudulent.

As a way to prepare dataset for machine learning model training, variable (feature) time was converted to the convenient datetime format using the 'pd.to_datetime(df['Time']' , unit='s') method, which subsequently was converted to a binary considering two datetime clusters with a 0 and 1 values in integer datatype.

Part of EDA was to ensure each feature being in appropriate datatype, which was a key action performed in preparing features into float and integer for further features transformation before building the model and training on the set. Below is a look of features and respective datatype:

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

Feature Engineering

Feature engineering is the process that takes raw data and transforms it into features that can be used to create a predictive model using machine learning or statistical modeling, such as deep learning – preparing an input data set that best fits the machine learning algorithm as well as enhancing the models' performance.
Based on the preliminary model training and visualization of the feature importance, it turned obvious that features in the below order of the importance could be taken off the model to see how performance metrics improve with a removal a few features. This trial process led to identification of optimal features set to be retained by the model. Among the features, the following were dropped out: 'V2', 'V5', 'V20', 'V21', 'V22', 'V24', 'V25', 'V27', 'V28', and 'Time_of_day'.

Feature Scaling, Standardization, and Normalization

Key aspects of feature engineering are scaling, normalization, and standardization, which involves transforming the data to make it more suitable for modeling. These techniques can help to improve model performance, reduce the impact of outliers, and ensure that the data is on the same scale.

Feature Scaling

Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.
Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process.

Features Normalization

Normalization is a data preprocessing technique used to adjust the values of features in a dataset to a common scale. This is done to facilitate data analysis and modeling, and to reduce the impact of different scales on the accuracy of machine learning models.
Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.
Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature, respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator, and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

Features Standardization

Standardization is another scaling method where the values are centered around the mean with a unit standard deviation. That's the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.
Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

$\mu$is the mean of the feature values and$\sigma$is the standard deviation of the feature values. Note that, in this case, the values are not restricted to a particular range.

For the purpose of this project, only standardization of the continuous features was applied using the *StandardScaler() method, which* was imported from the 'sklearn.preprocessing'

Normalization could have been performed using the MinMaxScaler() method imported from the same 'sklearn.preprocessing', but categorical variables 'Time_of_day' and

‘Class’, the target were already in binary 0 and 1 classes. However, the general MinMaxScaler() method was prepared to be included in the model pipeline in case future model modification is made after more features are involved.

With the processes above, the pipeline below was created:

```
Pipeline

Preprocessor: ColumnTransformer

Categorical


scaling
MinMaxScaler

XGBClassifier

Pipeline

Preprocessor: ColumnTransformer

categorical


XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, device=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=None, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
multi_strategy=None, n_estimators=None, n_jobs=None,
num_parallel_tree=None, random_state=None, ...)
```

Model Selection:

It is already from the model pipeline above, an XGBClassier() was built. The XGBClassifier model was selected because of the following factors, which promote the model hyperparameter fine-tuning due to the model flexibility on:

- Regularization techniques required.
- Handling of missing data the case.
- Model interpretability ability to plot feature importance, confusion matrix, and classification report.

- Scalability of the model as the model could handle large dataset clients may have.
- Community support making reliability to access key information for future model usability and improvement.

But what is XGBoost Model?

The XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm used for regression and classification (binary and multiclass) tasks on tabular datasets implementing a gradient boosting on trees technique, leading to a remarkable machine learning performance because of handling of a variety of data types, relationships, distributions, and the variety of hyperparameters that you can fine-tune.

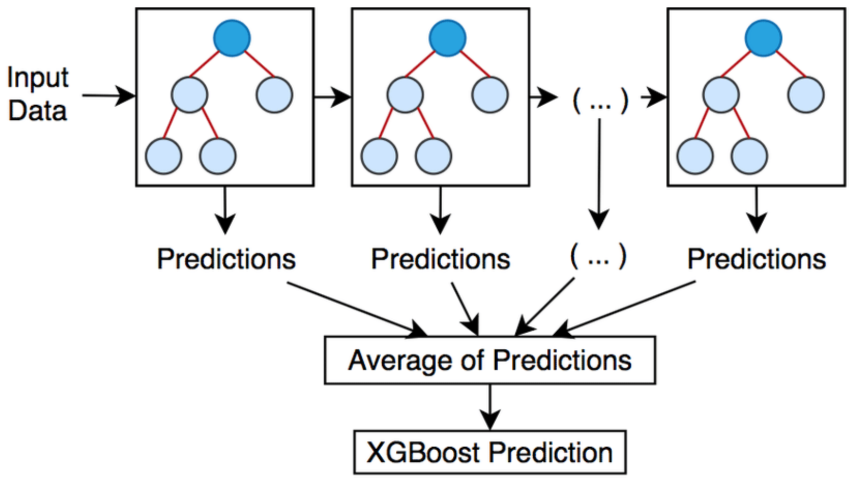
The following are the benefits XGBoost machine learning models offer:

- Latest version - The open-source XGBoost algorithm typically supports a more recent version of XGBoost. To see the XGBoost version that is currently supported, see XGBoost SageMaker Estimators and Models.
- Flexibility - Take advantage of the full range of XGBoost functionality, such as cross-validation support. You can add custom pre- and post-processing logic and run additional code after training.
- Scalability - The XGBoost open-source algorithm has a more efficient implementation of distributed training, which enables it to scale out to more instances and reduce out-of-memory errors.
- Extensibility - Because the open source XGBoost container is open source, you can extend the container to install additional libraries and change the version of XGBoost that the container uses. For an example notebook that shows how to extend SageMaker containers, see Extending our PyTorch containers.

The Boosting technique trains models serial-wise, with every new learning model trained to correct the errors made by its predecessors. In Gradient Boosting, the model tries to optimize the loss function of the previous learning tree by adding a new adaptive model that combines weak learning models. This reduces the loss function. The technique, thus, controls the learning rate and reduces variance by introducing randomness. Overall, this technique is known as “Ensemble”, which creates multiple models and then combine them to produce improved results. Ensemble methods in machine learning usually produce more accurate solutions than a single model would. This performance can be improved even further by using XGBoost. The objective function of the XGBoost algorithm is the sum of a specific loss function evaluated over all the predictions and the sum of regularization term for all predictors, such as:

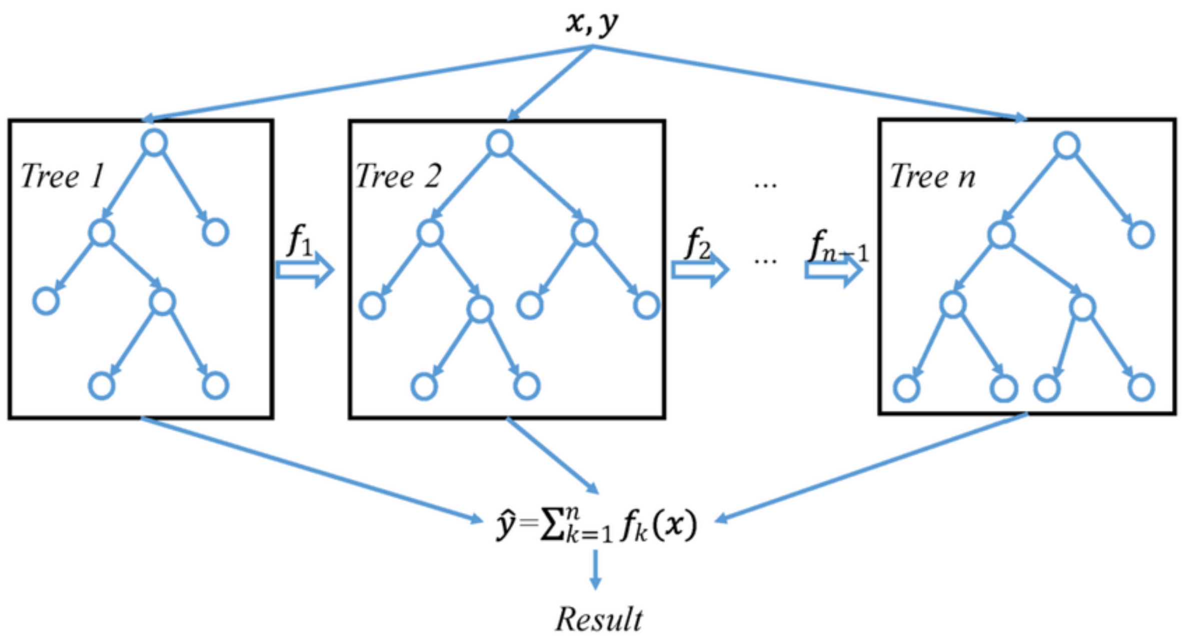
$$O(\theta) = \sum_{i} l(y_i - \hat{y_i}) + \sum_{j} \Omega(f_j)$$

Extreme Gradient Boosting (XGBoost) Model Algorithm



Source: 3: XGBoost model (Source: Self). | Download Scientific Diagram (researchgate.net)

Extreme Gradient Boosting (XGBoost) Model Architecture



Source: A general architecture of XGBoost | Download Scientific Diagram (researchgate.net)

XGBoost and its Uniqueness in optimizing performances comparatively

The XGBoost Classifier is scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted tree algorithms.

A few benefits why one may consider using this Classifier over others, below are the unique factors making the algorithm peculiar over other machine learning models:

- **Capability to handle sparse data:** XGboost is capable of handling sparse data and hence missing value treatments are not necessary.
- **Block structures and parallel processing:** Unlike many other machine learning algorithms, XGBost can concurrently use multiple cores of the CPU at the same time owing to its block structure in the system design. Because of this capability, XGBoost can work exceptionally faster and can converge well.
- **Cache awareness and out-of-core computing:** XGBoost has been designed keeping in mind the optimal use of hardware as well. Owing to this property, the algorithm works by allocating internal buffer memories at each step and hence uses the cache in the most efficient way. To add to this, the algorithm, while handling very large datasets in typical big data problems can compress the large data into small versions thus optimizing the disk space and computational speed. This property is termed as '*out of core*' computation.
- **Built-in cross-validation:** XGBoots algorithm by its design could cross-validate models while developing. This reduces the chance of overfitting and thus largely helps in maintaining the bias-variance trade-off.
- **Tree pruning:** XGBoost makes splits up to the ***max_depth*** specified and then starts pruning the tree backward and removing splits beyond which there is no positive gain. This process of backward tree pruning stops XGBoost from being a greedy algorithm and doesn't result in an overfit model.
- **Latest version:** The open-source XGBoost algorithm typically supports a more recent version of XGBoost. To see the XGBoost version that is currently supported, see XGBoost SageMaker Estimators and Models.
- **Flexibility:** Take advantage of the full range of XGBoost functionality, such as cross-validation support. You can add custom pre- and post-processing logic and run additional code after training.
- **Scalability:** The XGBoost open-source algorithm has a more efficient implementation of distributed training, which enables it to scale out to more instances and reduce out-of-memory errors.
- **Extensibility:** Because the open source XGBoost container is open source, you can extend the container to install additional libraries and change the version of XGBoost that the container uses. For an example notebook that shows how to extend SageMaker containers, see Extending our PyTorch containers.
- **Performance and Accuracy:** It often outperforms other traditional machine learning algorithms, especially in structured/tabular data scenarios, combining the predictions from multiple weak learners (decision trees) to creating a strong learner while handling non-linearity even with complex relationships across features.

- **Regularization Techniques:** It incorporates regularization LASSO (L1) and Ridge (L2) techniques, which help prevent model training overfitting while improving the model generalization ability. Regularization penalizes overly complex models, which is essential for preventing fitting noises in the training data, which is crucial in binary classification tasks where overfitting may lead to poor generalization.
- **Handling Missing Data:** XGBoost model has built-in capabilities to handle missing data, reducing the need for extensive data preprocessing using a technique called "Sparsity Aware Split Finding" to efficiently handle missing values during the training process. This could be beneficial when dealing with real-world datasets that often have missing or incomplete information (not though for this project).
- **Feature Importance Analysis:** It provides a feature importance analysis, which brings a clear understanding of each feature contribution to the model's prediction, which is valuable model interpretation aspect and factor for model clarity to stakeholders.
- **Scalability/Parallel and Distributed Computing:** the Model is scalable during training and testing in the way can scale out to more instances and reduce out-of-memory errors. It is designed for parallel and distributed computing making it efficient for large datasets and speeding up the training process. The model efficiently utilizes resources and could be parallelized across multiple CPU cores or even distrusted across clusters of machines, which is essential for real-world situations with large datasets.
- **Flexibility and Customization:** XGBoost is highly flexible and allows for customization of hyperparameters to fine-tune the model's performance. The Data Scientist has that flexibility to adjust various hyperparameters such as learning rate, tree depth, and number of trees in optimizing the model for prediction role. Hence, fine-tuning for different use cases.
- **Wide Adoption and Community Support:** Taking advantage of shared knowledge, resources, and continuous improvements and hence faster issue resolution and access to wealth of resources.
- **Integration with SageMaker and Cloud Service:** XGBoost is seamlessly integrated into cloud services like AWS SageMaker (where this model is deployed), making it easier for deployment and scalability. SageMaker provides tools for training, testing, and deploying, and monitoring the XGBoost models at scale.

Summing up, XGBoost model is a powerful and versatile algorithm that excels in binary classification tasks. Its performance, regularization techniques, handling of missing data, interpretability, and community support are compelling factors for being chosen for this project.

XGBoost Parameters

There are three different kinds of parameters.

- **General Parameters:** For overall functioning like the type of model (classification/regression), displaying error message, and so on.
- **Booster parameters:** These are the main sets of parameters that guide individual trees at every step. Some of these booster parameters are listed below:
  - Eta: Learning rate.
  - Max_depth: The maximum depth of the component decision tree.
  - Max_leaf_nodes: The maximum number of terminal nodes in the decision tree.
  - Subsample: fraction of observation which is to be selected for random sampling of each tree.
  - colsample_bytree: Kind of the maximum number of features. Denotes the fraction of columns to be random samples for each tree.
- **Learning task parameters:** As the name indicates, these parameters define the optimization objective and the metric (RMSE, MAE, LogLoss, Error, AUC, etc.), which are calculated at every step during the ensemble technique.

Model Training and Validation

Model Training on the Train Set

The model was trained through the created pipeline (pipeline_xgb_model.fit()) by fitting in the input variable X_train, and y_train.

```
pipeline_xgb_model.fit(X_train, y_train)
```

From the above pipeline, the predictor is created as below:

```
y_train_pred = pipeline_xgb_model.predict(X_train)
```

Then model performance accuracy calculated using the accuracy score from the sklearn.metrics.

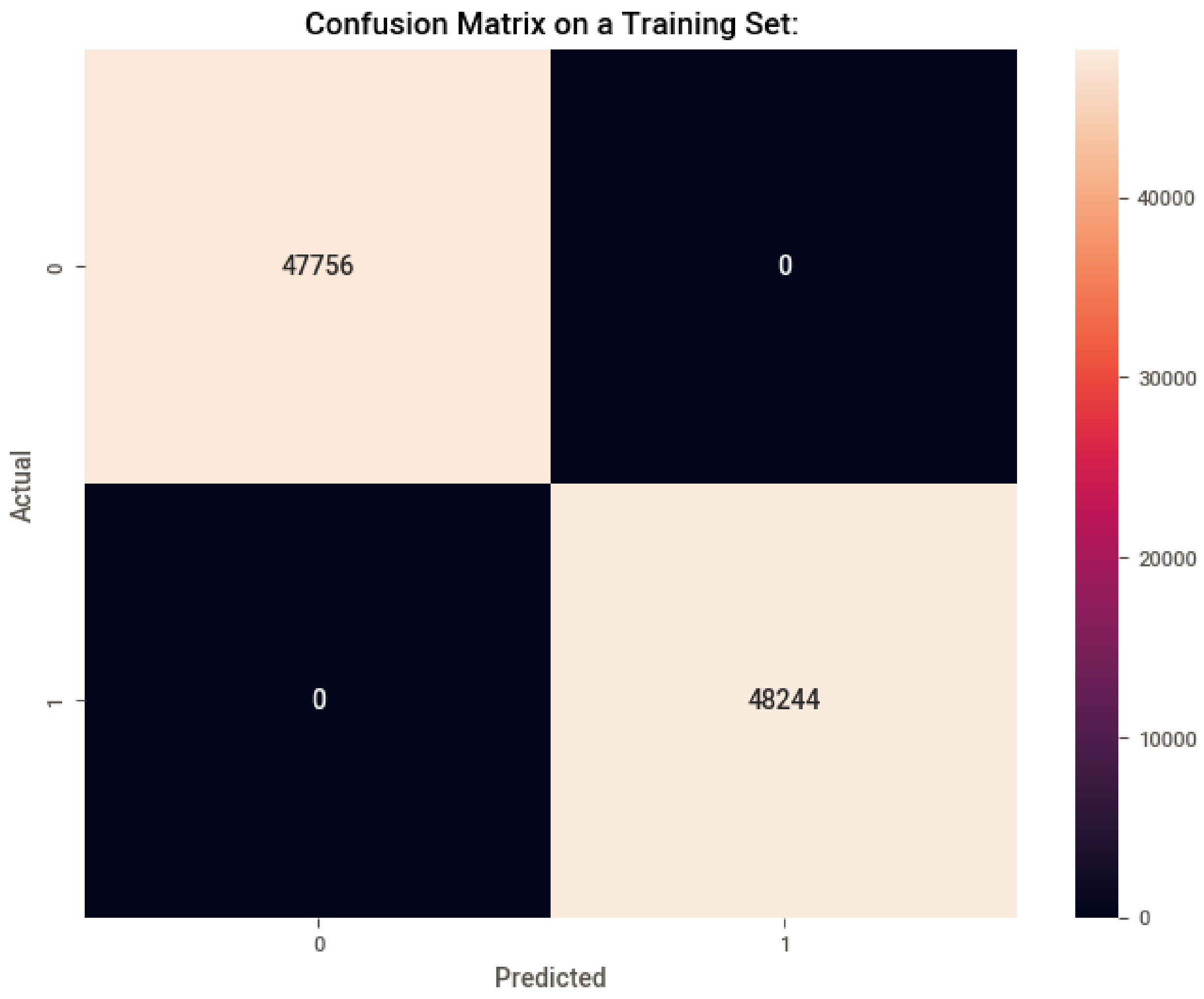
```
train_accuracy = accuracy_score(y_train, y_train_pred)

accuracy = round(train_accuracy*100,)
```

Then accuracy printed using the below code:

```
print(f"Training Accuracy: {accuracy}%")
```

The model performed well as accuracy was at 100% consistently indicated across the confusion matrix visualization and running the classification report where precision was at 100%, recall at 100%, and f1-score at 100% as well.

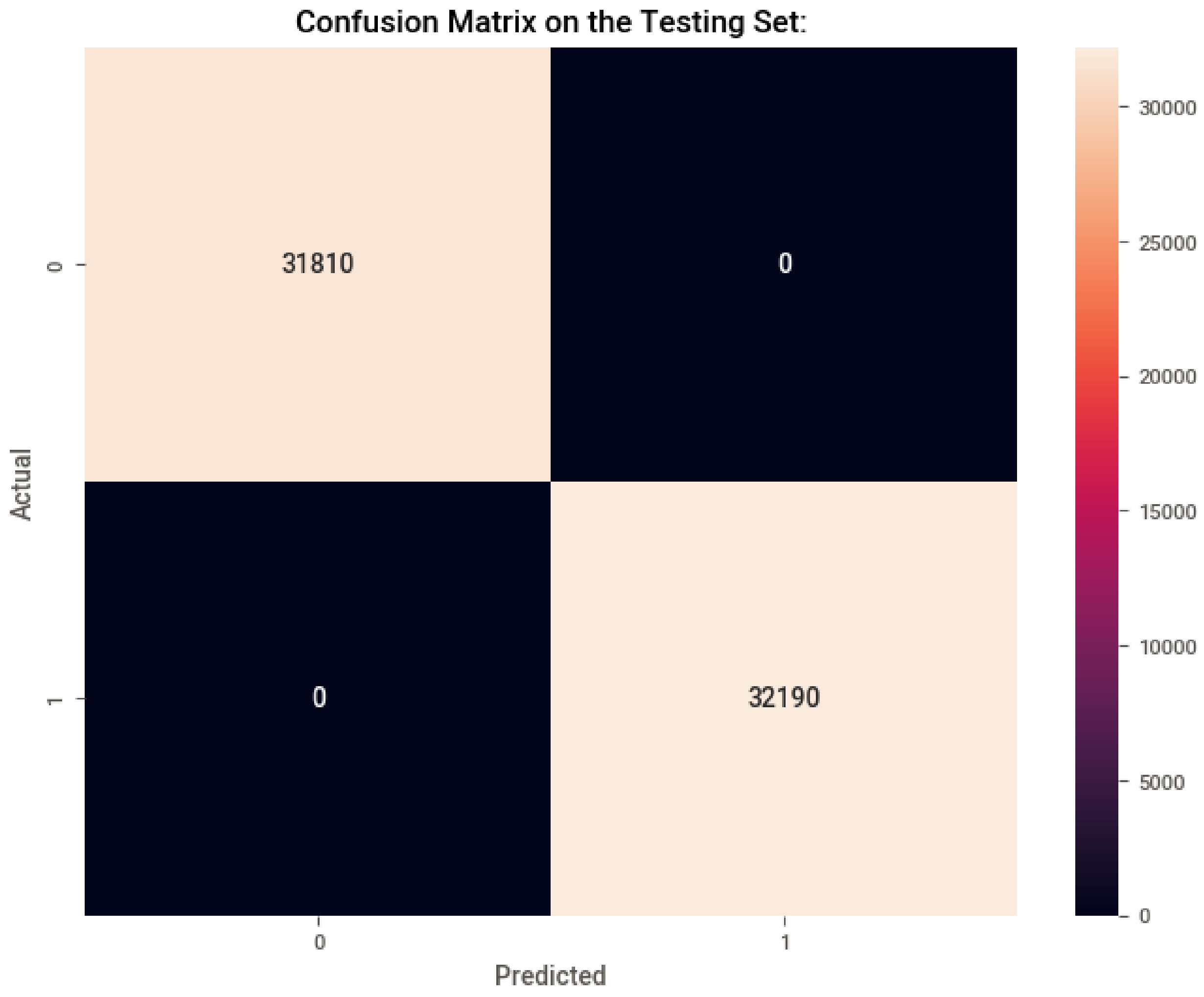


Model Validation/Testing using the Test Set

Again, the same procedure was applied such that fitting of the test input variables on the pipeline_xgb_model was done:

y_pred = pipeline_xgb_model.predict(X_test)

Checking on the Model Testing Performance Metrics, it consistently produced similar figures across the accuracy, precision, recall, and f1-score (100%) confirmed by the confusion matrix plot below:

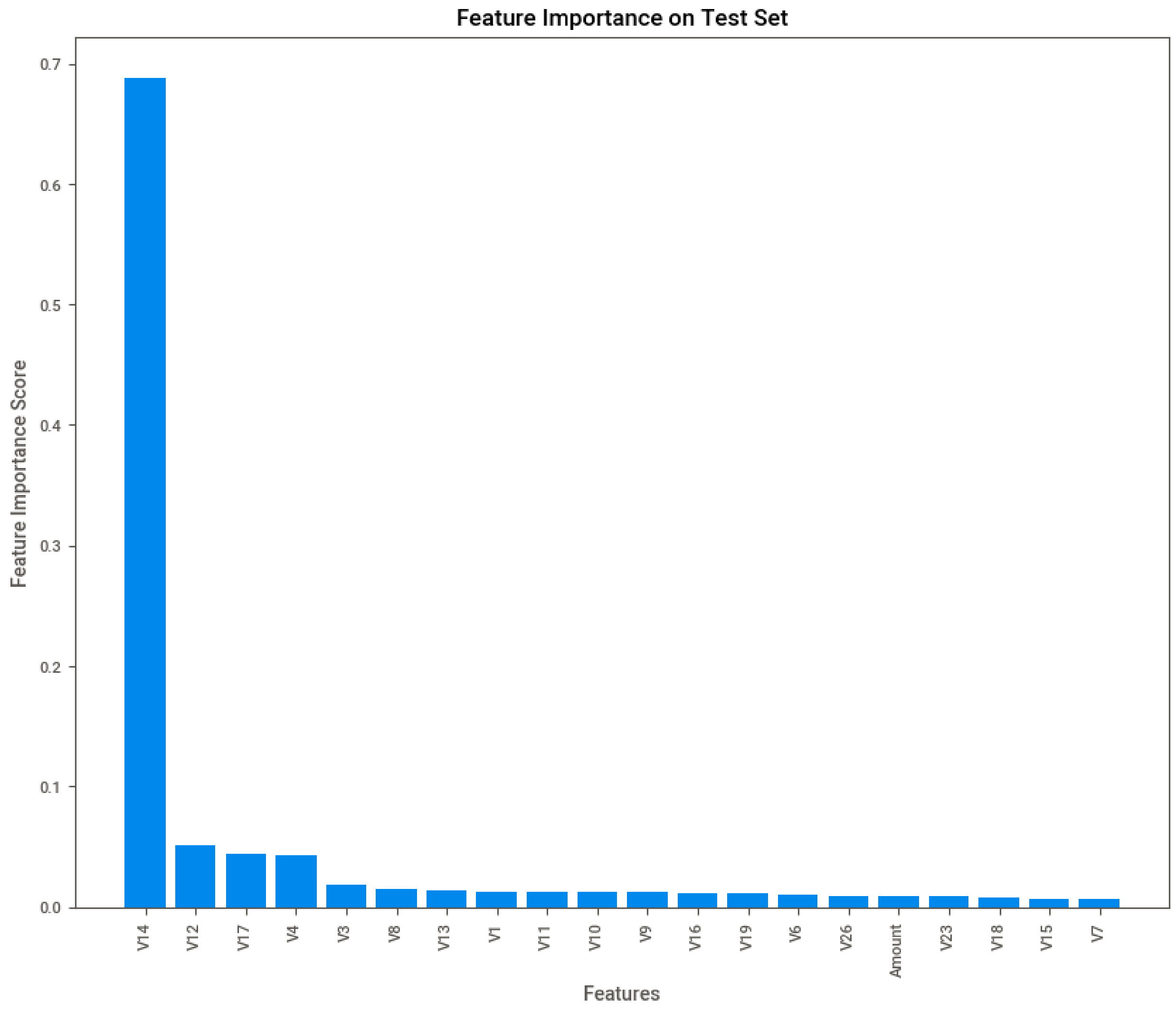


Display of Actual Vs Predicted and Respective Probability Values

Actual Class	Predicted Class	Probability
1	1	3.030300e^{-04}
1	1	1.192093e^{-07}
0	0	9.996675e^{-01}
1	1	2.384186e^{-07}
1	1	5.960464e^{-07}
1	1	1.192093e^{-07}
1	1	4.649162e^{-06}
0	0	9.99872e^{-01}

Key features as per the Model Feature Importance Order

After dropping the least features on the feature importance order, V14, V12, V17, V4, V3, V13, V1, V11, V10, V9, V16, V19, V26, Amount, V23, V18, V15, and lastly V7 were incorporated to train and test the model leading to the producing a perfect predictor for determining whether a credit card transaction is fraudulent. Note that both the training and testing feature importance plot have similar order of feature importance.



Model Performance Metrics Interpretation

- Based on the confusion matrix actuals vs predicted perfectly performed as there were zero missed predictions.
- Based on the Classification Report: Precision, Recall, F1-Score, and Accuracy are consistently 100% in line with the confusion matrix visualization.
- V14 feature is the most important in predicting whether transaction fraudulent followed by V12, V7, V4, V17, V3, V8, V1, ... etc. This may help Banks refocus time and resources only on those highly ranked features in predicting likelihood of fraudulent credit card transactions.
- Focusing on important features leads to time saving and reduced computational complexity (cost saving).

Model Usability

- High Net Promoter Score (NPS) -- Banks are doing great with far more happy customers than unhappy ones.
- Potential Overall Cost Savings: Using this model by the Bank may save money in preventing fraudulent transactions. It can also improve customer satisfaction by reducing the number of legitimate transactions incorrectly flagged as fraudulent.
- Model Resilience and reliability. This model may maintain its fraudulent predictive power (performance) on unseen data, which is important in a real-world setting where transactions distribution changes over time. Hence, increased demand for the credit cards services.
- Monitoring and Adaptation: Even with a high-performing model, it's essential to continuously monitor its performance and adapt it as needed. Fraudsters can change their tactics, and the model should be updated to address evolving threats related to credit cards' transactions.
- Customer Communication: In the banking domain, clear communication with customers is essential. Customers should be informed about the bank's fraud detection methods and how they can protect themselves. An overzealous fraud detection system may sometimes lead to customer inconvenience.
- Marketing Strategy: The bank may use the high-performance model promoting itself as a secure place to conduct financial transactions, emphasizing the commitment to protecting customers from fraud.
- Regulatory Compliance: Ensure that the model complies with relevant regulations and data privacy laws. Transparency in model development and usage is critical.
- Education and Training: Continuously educating and training the Bank employees on how to use and interpret the model results, as well as how to take appropriate actions when fraud is suspected.

Future Work

I am motivated to further advance my data science technical skills set through engagement in many supervised and unsupervised machine learning model building on real-world problems as much as opportunities emerge. I look forward to the opportunities excelling in the Data Science field.

Reference:

3: XGBoost model (Source: Self). | Download Scientific Diagram (researchgate.net)

6.3. Preprocessing data — scikit-learn 1.3.2 documentation

A general architecture of XGBoost | Download Scientific Diagram (researchgate.net)

A Guide on XGBoost hyperparameters tuning | Kaggle

Amazon SageMaker Model Building Pipeline — sagemaker 2.197.0 documentation

amazon-sagemaker-examples/introduction_to_amazon_algorithms/xgboost_abalone/xgboost_abalone_dist_script_mode.ipynb at main · aws/amazon-sagemaker-examples (github.com)

argparse — Parser for command-line options, arguments and sub-commands — Python 3.12.0 documentation

Calculating Accuracy of an ML Model. | by Abhigyan | Analytics Vidhya | Medium

Customer Churn Prediction with XGBoost — Amazon SageMaker Examples 1.0.0 documentation (sagemaker-examples.readthedocs.io)

Development - pip documentation v24.0.dev0 (pypa.io)

Feature Engineering: Scaling, Normalization and Standardization (analyticsvidhya.com)

Frameworks — sagemaker 2.197.0 documentation

Getting Started - pip documentation v24.0.dev0 (pypa.io)

How to Check the Accuracy of Your Machine Learning Model (deepchecks.com)

Hyperparameter tuning in XGBoost. This tutorial is the second part of our… | by Cambridge Spark | Cambridge Spark

Learn How to Use Python's Pandas Profiling Library | Zuar

ML | XGBoost (eXtreme Gradient Boosting) - GeeksforGeeks

Packaging Python Projects - Python Packaging User Guide

Pipeline Steps - Amazon SageMaker

sagemaker · PyPI

sagemaker-python-sdk/src/sagemaker/xgboost at master · aws/sagemaker-python-sdk (github.com)

Scikit-Learn — sagemaker 2.171.0 documentation

Scikit-Learn — sagemaker 2.196.0 documentation

seaborn.pairplot — seaborn 0.13.0 documentation (pydata.org)

The Packaging Flow - Python Packaging User Guide

Tuning XGBoost Hyperparameters - KDnuggets

Use Version 2.x of the SageMaker Python SDK — sagemaker 2.196.0 documentation

Use XGBoost with the SageMaker Python SDK — sagemaker 2.197.0 documentation

Using Scikit-learn with the SageMaker Python SDK — sagemaker 2.197.0 documentation

XGBoost - GeeksforGeeks

XGBoost — sagemaker 2.197.0 documentation

XGBoost Algorithm in Machine Learning - Shiksha Online

XGBoost Algorithm using Python – Machine Learning Geek

XGBoost Classes for Open Source Version — sagemaker 2.196.0 documentation

XGBoost Parameters — xgboost 2.0.1 documentation

XGBoost Parameters Tuning | Complete Guide With Python Codes (analyticsvidhya.com)