ML Maestros

Convolve 2.0

1 Abstract

The rise of online banking has streamlined customer interactions with financial institutions but has also facilitated fraudulent activities, notably through the exploitation of online channels by money mules. We present an automated transaction monitoring approach utilising machine learning to identify possible money mule accounts, with a particular focus on Bank A at IIT Bombay. Our objective is to forecast the probability that an account would be linked to illegal activity given a dataset containing account information and transaction history. Using labelled data, we will train machine learning models and assess their effectiveness using a different validation dataset. Preprocessing the data, feature engineering, choosing a model, and utilising the right metrics to assess success are all part of our process. Our study aims to protect consumer interests and proactively limit financial risks by improving Bank A's fraud detection capabilities. To support continuing work in fraud detection tactics, we will record our approach, important findings, and suggestions.

Analysing Data

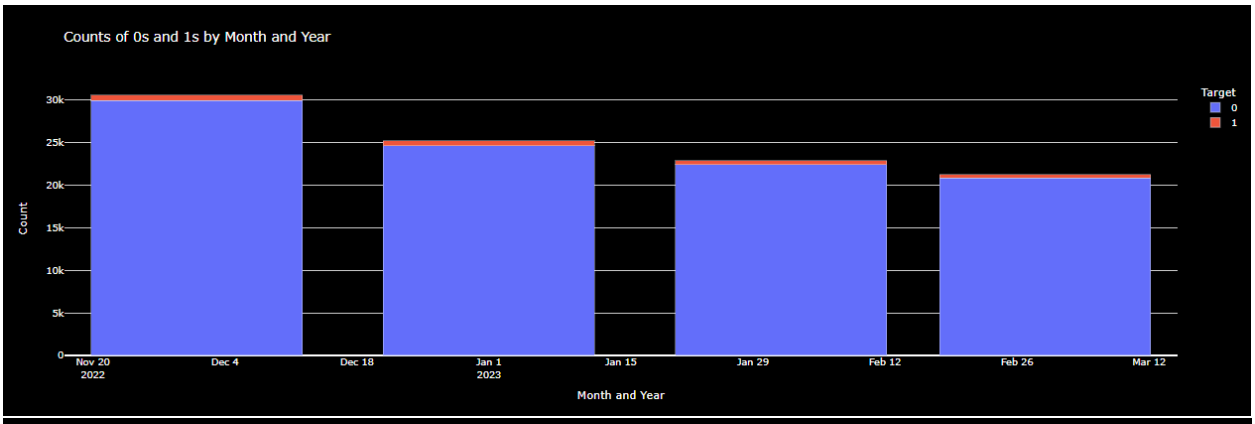
Feature Selection

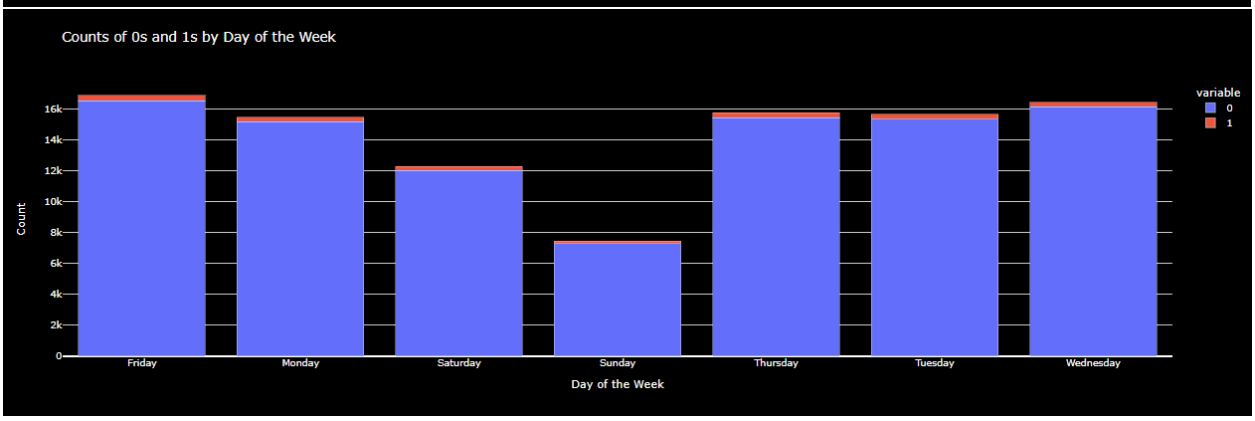
Model Training and Selection

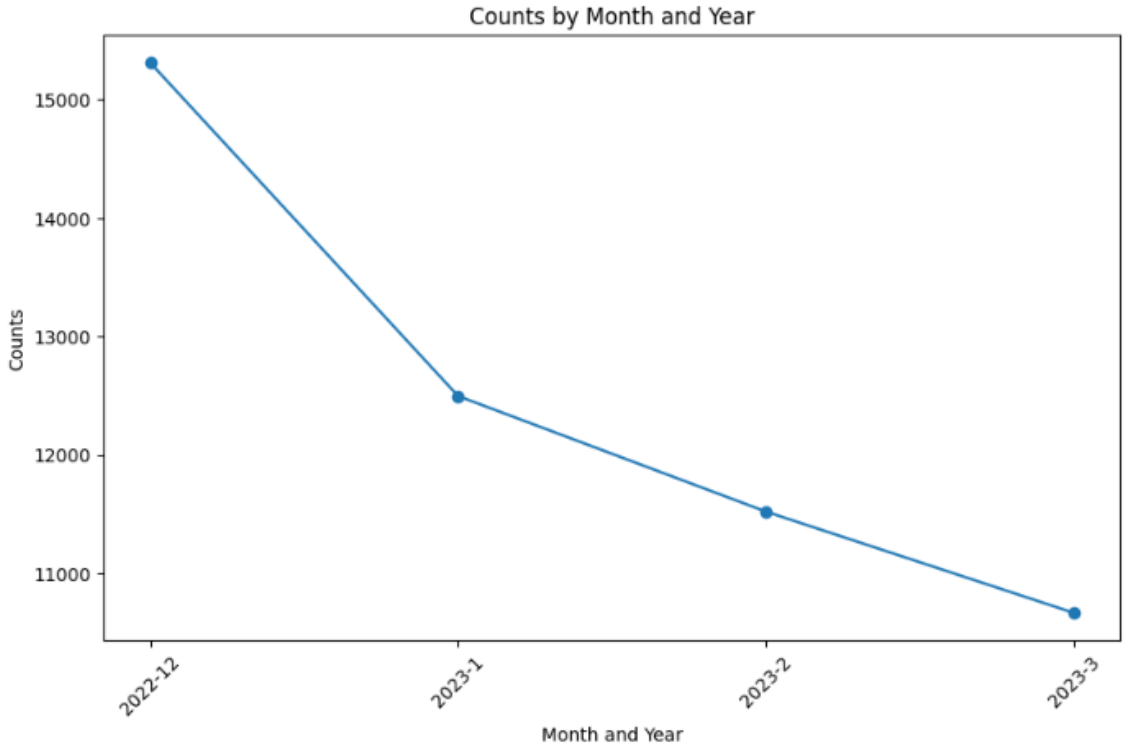
Conclusion & Results

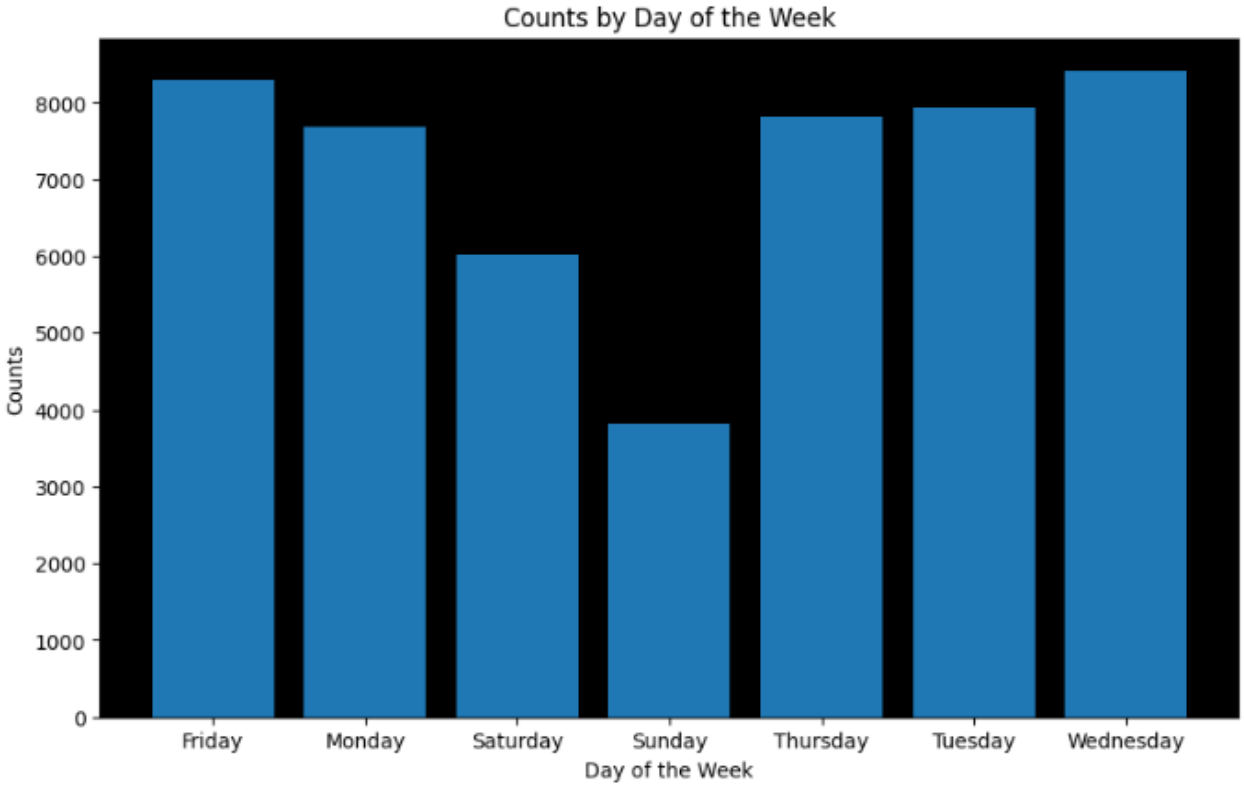
2 Analysing Data

In this section, we delve into the analysis of the provided dataset to gain insights into the characteristics and patterns of the data. We explore the distribution of various features, identify any missing values or outliers, and assess the correlation between different variables. Additionally, we conduct exploratory data analysis (EDA) to uncover any trends or anomalies that may be indicative of fraudulent activities. By understanding the underlying structure of the data, we lay the foundation for building effective machine-learning models. We thoroughly analyzed the data and tried to find patterns. We calculated the number of mule accounts spread across different days of the week, months of the year, we also tried finding the relation between number of accounts opened spread across days of the week and months of the year.

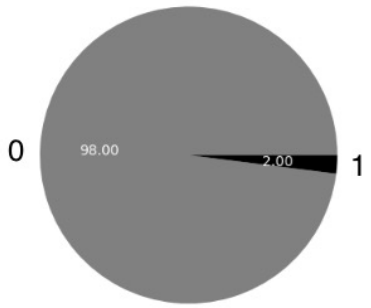

Counts of 0s and 1s by Month and Year
Target
0
1
Count
30k
25k
20k
15k
10k
5k
0
Nov 20
2022
Dec 4
Dec 18
Jan 1
2023
Jan 15
Jan 29
Feb 12
Feb 26
Mar 12
Month and Year


Counts of 0s and 1s by Day of the Week
variable
0
1
Count
16k
14k
12k
10k
8k
6k
4k
2k
0
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
Day of the Week


Counts by Month and Year
Counts
15000
14000
13000
12000
11000
2022-12
2023-1
2023-2
2023-3
Month and Year



- **Inference:** By plotting these graphs we saw that the data was not skewed and was more or less evenly distributed throughout days and months.



- We found out there were almost 2% mules in the total dataset for testing, this implies that for 100000 records 1000 people were mules.

We also tried finding columns that had the most unique values. If there were many categorical columns then the size of the grid in one hot encoding would become too large, thus we tried eliminating categorical values.

```
Column 'country_code' has 49 unique values.
Column 'demog_2' has 24 unique values.
Column 'income' has 12 unique values.
Column 'city_tier' has 9 unique values.
Column 'occupation' has 8 unique values.
Column 'demog_4' has 7 unique values.
Column 'demog_9' has 2 unique values.
Column 'demog_10' has 1 unique values.
Column 'demog_12' has 1 unique values.
Column 'demog_13' has 2 unique values.
Column 'demog_14' has 2 unique values.
Column 'demog_15' has 2 unique values.
Column 'demog_16' has 2 unique values.
Column 'demog_17' has 2 unique values.
Column 'demog_18' has 2 unique values.
Column 'demog_19' has 2 unique values.
Column 'demog_20' has 2 unique values.
Column 'demog_21' has 2 unique values.
Column 'demog_22' has 2 unique values.
Column 'os' has 2 unique values.
Column 'email_domain' has 10 unique values.
Column 'demog_40' has 3 unique values.
Column 'demog_43' has 3 unique values.
Column 'day_of_week' has 7 unique values.
```

We did the following to remove unnecessary data-

- In the categorical columns which had more than 80% of the value as NaN, we removed those columns entirely.
- Categorical columns which had certain categories that appeared less than 1% of the times, we clubbed all such categories into a category called 'others'.
- We converted the account opening date to 3 columns based on the months to reduce unique values while incorporating one hot encoding.

3 Feature Selection

Weight of Expectation and Information Value

- Weight of Evidence (WOE) and Information Value (IV) are statistical techniques commonly used in credit scoring, risk modeling, and other areas of predictive modeling. They help assess the predictive power of independent variables and their ability to differentiate between different outcomes, especially in binary classification problems.
- **Weight of Evidence (WOE):**
  - **Definition:**
    - WOE measures the strength of the relationship between a predictor variable and the binary target variable.
    - It is commonly used in the context of logistic regression and credit scoring.
  - **Calculation:**
    - For each category (bin) of a predictor variable, WOE is calculated as the natural logarithm of the ratio of the proportion of target events to the proportion of non-events.
    - WOE is calculated using the following formula:

$$\text{WOE} = \ln\left(\frac{\text{Proportion of Events}}{\text{Proportion of Non-Events}}\right)$$

- **Interpretation:**
  - If WOE is positive, it indicates that the odds of the event occurring are higher for that group.
  - If WOE is negative, it suggests that the odds of the event occurring are lower for that group.
  - A WOE close to zero implies that the event and non-event odds are similar.
- **Application:**
  - WOE is used to transform categorical variables into continuous variables that are more suitable for modeling.
  - It helps in identifying which categories of a predictor contribute more or less to the predictive power of the model.

- **Information Value (IV):**
  - **Definition:**
    - IV is a measure of the overall predictive power of a predictor variable.
    - It aggregates the contribution of each category (bin) of the predictor variable to the model's predictive power.
  - **Calculation:**
    - IV is calculated as the sum of the differences between the proportions of events and non-events, weighted by the WOE for each category:

$$\text{IV} = \sum ((\text{Proportion of Events} - \text{Proportion of Non-Events}) \times \text{WOE})$$

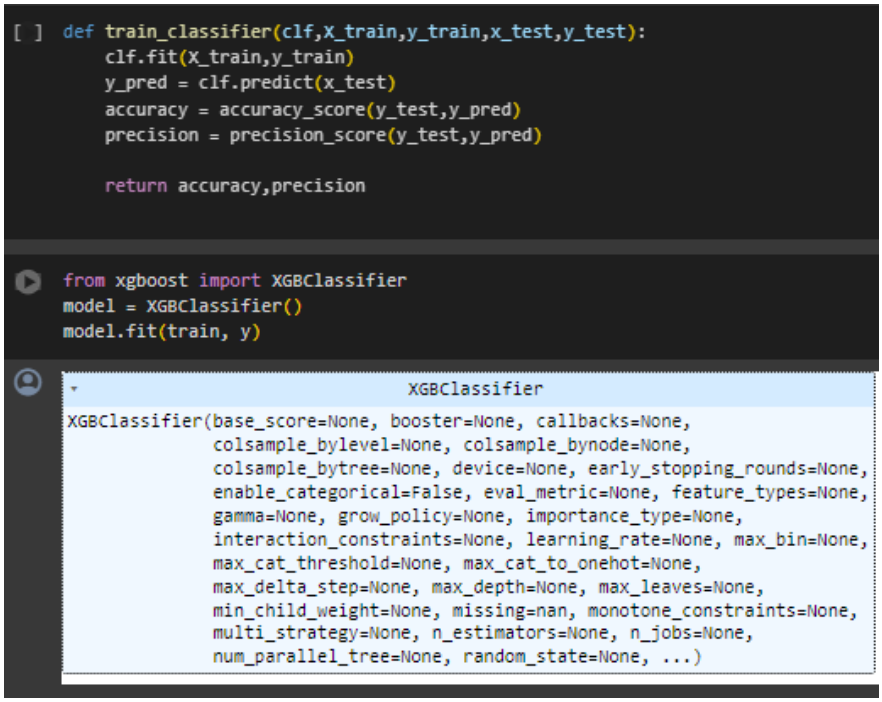
  - **Interpretation:**
    - A higher IV indicates that the predictor variable has more predictive power.
    - IV values are often interpreted as follows:
      - Less than 0.02: Not useful for prediction
      - 0.02 to 0.1: Weak predictive power
      - 0.1 to 0.3: Moderate predictive power
      - 0.3 or higher: Strong predictive power
  - **Application:**
    - IV helps in variable selection by identifying which variables are most informative for the model.
    - Variables with low IV may be candidates for removal from the model, as they contribute less to predictive accuracy.

In summary, WOE and IV are useful tools in building predictive models, particularly in scenarios where the target variable is binary. They provide insights into the relationship between predictor variables and the target variable, aiding in variable transformation and selection for improved model performance.

We have used WOE and IV finally to find the columns that can be used effectively for prediction. After using one hot encoding, we have also used SelectkBest features to find the best 50 features other than those obtained by WOE and IV. We tried various other approaches, we tried selecting features using SelectKBest with MinMax Scaler, Lasso feature selection, Mutual Information,PCA ,compared results and finally stuck to WOE and IV along with SelectkBest.

4 Model Training and Selection

We split the data in the " dev_data_to_be_shared " into train and test with 80% and 20% respectively. We chose XGBoost as our model after trying various other models such as neural networks, Logistic Regression, Random Forests among others.



```
[ ] def train_classifier(clf,X_train,y_train,x_test,y_test):
        clf.fit(X_train,y_train)
        y_pred = clf.predict(x_test)
        accuracy = accuracy_score(y_test,y_pred)
        precision = precision_score(y_test,y_pred)

        return accuracy,precision
```

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(train, y)
```

```
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, ...)
```

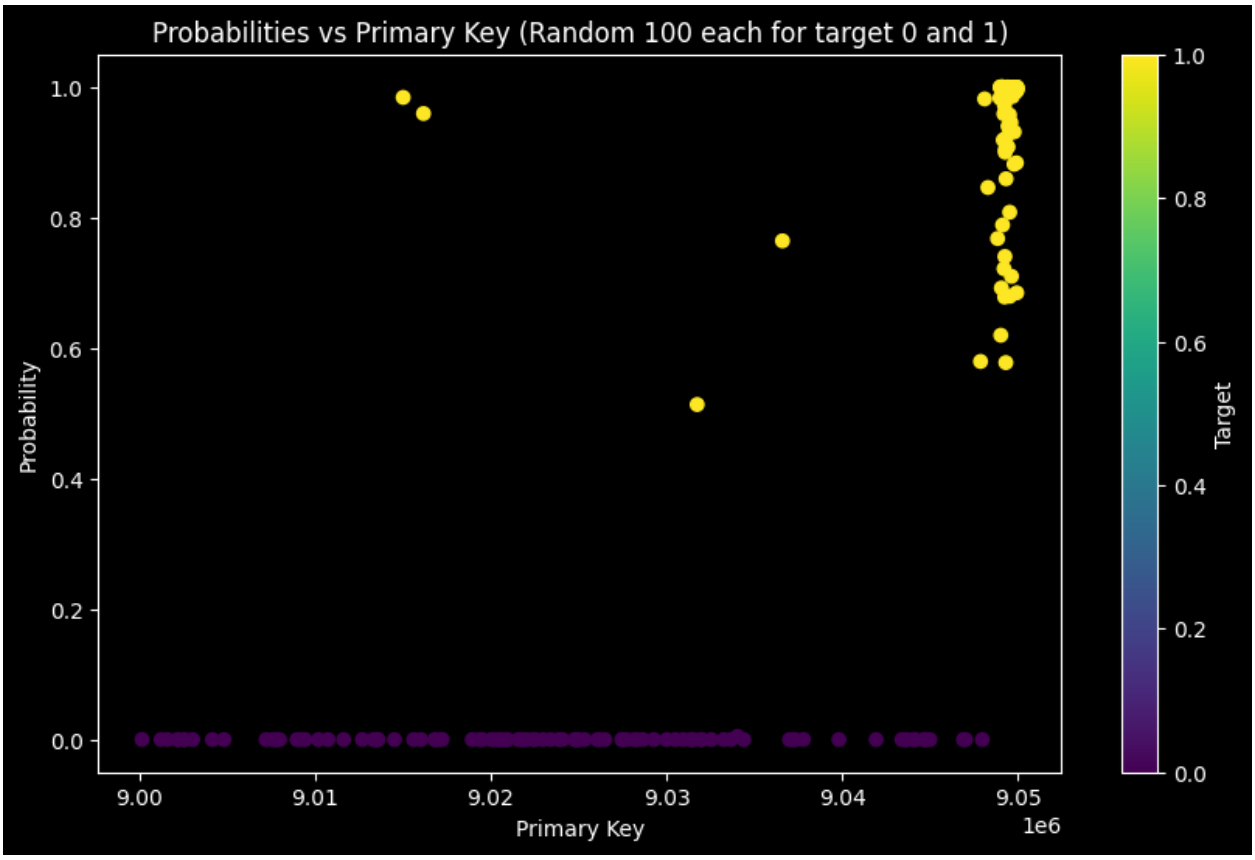
- **XGBoost:** XGBoost, short for Extreme Gradient Boosting, is a powerful and efficient machine learning algorithm that belongs to the family of gradient boosting methods. Developed to address shortcomings in traditional gradient boosting techniques, XGBoost excels in both speed and performance. It leverages a combination of tree ensemble models, regularization techniques, and parallel computing to achieve state-of-the-art results across various machine learning tasks. XGBoost is widely used in competitions on platforms like Kaggle and has become a popular choice in both research and industry for tasks such as classification, regression, and ranking. Its ability to handle large datasets, feature importance analysis, and flexibility in customization make it a versatile tool for building robust predictive models.

We also tried running various models on our set, repeatedly running on different split sizes and feature extraction. These are the models we applied:

```
clfs = {
    'KN': knc,
    'GNB': gnb,
    'BNB': bnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'ETC': etc,
    'GBDT': gbdt,
}
```

5 Conclusion & Results

After we applied these steps we got an accuracy of **99.75%** on the train we split, We then used the model to predict mules on the validation_data, we got **974** ones i.e. the number of mules.



6 Final Code & CSV File

- **Google Colab Code**
- **CSV File**

Contact

Name	Email
Naman Sethi	namansethi22@iitk.ac.in
Tamoghna Kumar	tamoghna22@iitk.ac.in
Prem Kansagra	premk22@iitk.ac.in
Mayank Agrawal	mayanka22@iitk.ac.in