Documentation

Team Name : BitBrains

Pratham Tarjule
Apoorv Verma

1. **Objective** :

   The primary objective is to develop an automated transaction monitoring framework using machine learning algorithms to predict the probability of a given savings account being associated with a money mule. We will utilize a dataset comprising account-level attributes, demographic information, transaction history, and other relevant features to train and evaluate our predictive models

2. **Dataset Overview**:

   There are a total of 100000 data points in the dataset with 178 features dividing into the following types of attributes:

   i) demog : Demographic details of the account holder
   ii) txn : Transaction details
   iii) others : undisclosed features (mostly related to logins)

   Target Values - 0 for non-fraud and 1 for fraud transactions. 98000 samples have a target value of 1 and 2000 samples have a target value of 0, clearly indicating a problem of class imbalance.

3. Data Visualization:

We employed various data visualization techniques, including scatter plots, distribution plots, and scatter matrices, to gain insights into the underlying patterns and relationships within our dataset. These visualization methods played a crucial role in our exploratory data analysis (EDA) phase, allowing us to understand the distribution of features, identify potential correlations, and detect any outliers or anomalies present in the data.

a) Scatter Plot:

Scatter plots were instrumental in visualizing the relationship between two numerical variables in our dataset. By plotting each data point as a point on the chart, scatter plots helped us observe the general trend, dispersion, and potential clusters within the data.

We leveraged scatter plots to explore correlations between pairs of numerical features, facilitating the identification of linear or non-linear relationships that could influence our modeling decisions.

b) Distribution Plot:

Distribution plot provided valuable insights into the distribution of individual numerical features in our dataset. These plots enabled us to visualize the spread and shape of each feature's distribution, helping us assess its skewness, central tendency, and presence of outliers.

.

c) **Box Plot**:

We employed boxplots as a data visualization technique to gain insights into the distribution and variability of our data. Boxplots provide a concise summary of the central tendency, spread, and skewness of a dataset, making them valuable tools for exploratory data analysis and outlier detection

4. Data Preprocessing

Data preprocessing is an important step in the data analysis pipeline, involving the cleaning, transformation, and formatting of raw data into a usable format for analysis. We have implemented the necessary methods to get the dataset in a clean format which can reduce the load on the model during the training process.
We created a class called “DataPreprocessor” to implement the following preprocessing steps

4.1 Dropping Unnecessary Columns

We identified features which had more than 80% of their values missing from the dataset. Since imputing these features will only add noise to the dataset, we dropped those columns. In addition, we identified features with only one unique value across all data points. Since these features do not provide any useful information for modeling, we dropped them from the dataset to streamline the analysis and improve computational efficiency.

4.2 Encoding Categorical Values

Now our dataset had some categorical features and since machine learning models can only interpret numerical values,we converted the categorical variables into numerical values by using the following two methods:

a) Label encoding:

Label encoding is a technique where categorical variables are assigned unique numerical labels. It's useful for algorithms that require numerical inputs, but it can introduce ordinality where none exists. We used this method on the features which had clear indication of ordinality in them. Also some of the features had a mix of categorical and numerical variables with categorical variable unique values found to be 1 in that feature. So we replaced them with 0.

b) One hot encoding.

Categorical one-hot encoding transforms categorical variables into binary vectors, with each category represented by a binary digit, effectively removing any ordinality and ensuring compatibility with a wider range of algorithms. We used this method on the remaining categorical features. Thus converting all the categorical inputs into numerical ones. We used the get_dummies method from scikit-learn to implement this and also dropped the first column of the dummy_variables so as to escape from the dummy variable trap.

4.3 Handling Missing values

a) Imputation using Mean, Median and Mode

As we progress through the data preprocessing stage, we implemented the handle_missing_values function to systematically address missing data within our dataset. This method distinguishes between categorical and numerical columns, filling missing values with appropriate proxies to maintain data integrity. Categorical columns are imputed with the mode, preserving their categorical nature, while numerical columns undergo imputation using the median for robustness against outliers. With this approach, we ensure the dataset is thoroughly prepared for subsequent analyses or modeling tasks.

b) KNN Imputation

In addition to traditional imputation techniques, we incorporated the knn_imputer method to further enhance our data handling capabilities. Since the traditional imputation can work fine on a smaller dataset with smaller missing values , a dataset as big as this needs a more advanced imputation method so we used KNN imputation algorithm.By leveraging the K-Nearest Neighbors (KNN) imputation algorithm, we aim to address missing values more comprehensively. Specifically targeting columns with missing entries, this method employs KNN imputation to estimate values based on the nearest neighbors of each missing data point.

5. Feature Selection:

Feature selection is the process of identifying and selecting the most relevant features from a dataset to improve model performance, reduce overfitting, and enhance interpretability.
We employed two separate feature selection techniques for categorical and numerical features.

a) Chi-Squared Test for categorical features:

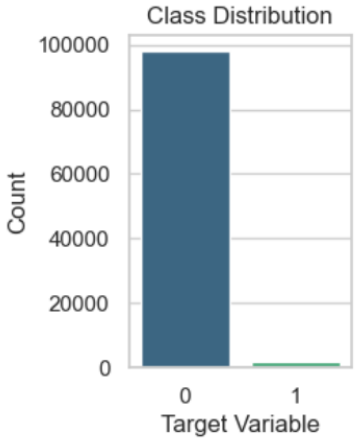
We applied the chi-square test for feature selection, leveraging the SelectKBest method from scikit-learn. Here, we specified the chi-square (chi2) as the scoring function and selected the top 11 features based on their scores. We chose k=11 since all the features after that had a very low score and it was giving the best result for our model.

b) Anova F-Test for numeric features

We started by extracting the numerical features (X_numerical) from our dataset and normalizing them using Min-Max scaling to ensure uniformity in their ranges.
The data is then split into training and testing sets using a stratified approach to preserve the distribution of the target variables. We employed the SelectKBest method from scikit-learn, specifying the ANOVA F-test (f_classif) as the scoring function.
By setting k=61, we aimed to select the top 61 numerical features based on their F-test scores, which indicate their relevance to the target variable.

This is how we selected the most important features from the dataset, totalling 72 features out of which 11 were categorical and 61 were numerical.

6. Oversampling - SMOTE



As we can see from the above image, this dataset is highly imbalanced. So to solve this problem, we used the SMOTE oversampling technique to equal the target classes. Class imbalance can lead to biased models that favor the majority class and perform poorly on minority class instances.
By oversampling the minority class using SMOTE, we aimed to balance the class distribution, improving the model's ability to generalize and make accurate predictions for both classes

8. Stratified K-fold cross validation:

We used Stratified K-fold cross validation with 5 splits since the target variable is imbalanced and traditional cross-validation methods may lead to biased evaluation results because they do not ensure that each fold maintains the same class distribution as the original dataset.
Stratified K-fold cross-validation addressed this issue by preserving the class distribution in each fold, making it particularly suitable for imbalanced datasets.

7. Algorithms Used

We used various ML models like Logistic Regression, SVM, LightGBM, etc. Among them XGBoost and Random Forests were the best performing models.

a) Random Forest

We got the following results from using the Random forest algorithm.

Model	Precision_1	Recall_1	F1-Score	ROC-AUC score
Random Forest	0.88	0.92	0.95	0.958

b) XGBoost

Since XGBoost was performing better than Random Forest in its vanilla form, we decided to use different techniques to optimize XGBoost. So we got three versions of XGBoost:
i) Vanilla XGBoost - simple XGBoost with default hyperparameters
ii) Hyper XGBoost - XGBoost with optimal hyperparameters
iii) HyperCV XGBoost - XGboost with optimal hyperparameters and using Stratified K-fold CV

Model	Precision_1	Recall_1	F1-Score	ROC-AUC score
Vanilla XGBoost	0.92	0.91	0.96	0.955
Hyper XGBoost	0.91	0.92	0.96	0.956
HyperCV XGBoost	0.91	0.93	0.96	0.999

7. Hyperparameter Tuning - GridSearch (Improving Precision and Recall)

We defined a grid of hyperparameters to search over using a Python dictionary. We configured GridSearchCV to perform an exhaustive search over the hyperparameter grid . We set the number of folds for cross-validation (cv) to 3 and used the F1-score as the evaluation metric. After Grid Search completed, we extracted the best hyperparameters and the corresponding best model from the search results.
We evaluated the performance of the best model on the held-out test set by making predictions and generating a classification report and confusion matrix. Following are the optimal parameters of the XGBoost model.

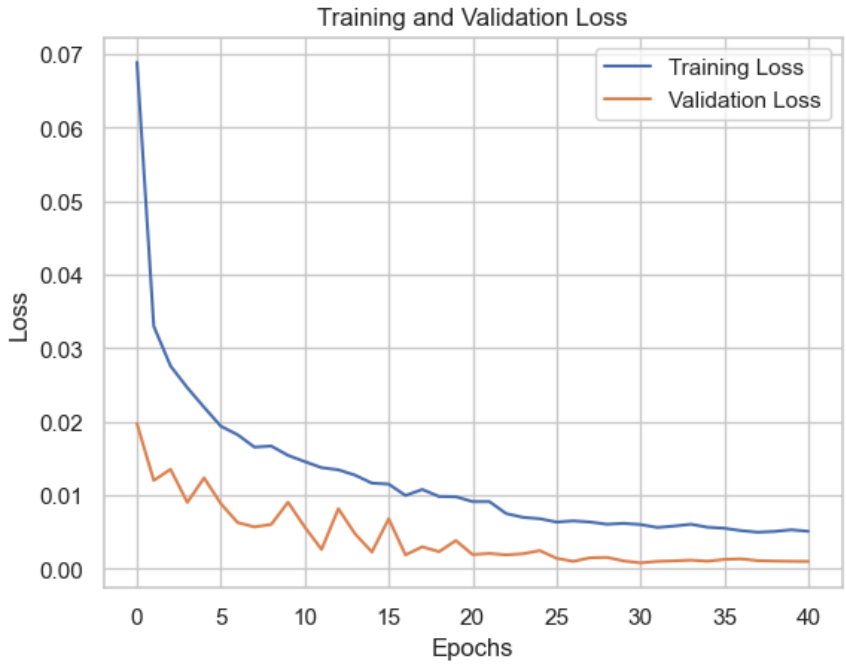
Best Hyperparameters: {'colsample_bytree': 0.45, 'gamma': 0.2, 'learning_rate': 0.05, 'max_depth': 11, 'min_child_weight': 0.5, 'n_estimators': 700, 'reg_alpha': 0.2, 'scale_pos_weight': 1, 'subsample': 0.45}

9. Deep Neural Network Model:

We constructed and trained a deep learning model using TensorFlow and Keras.The model architecture consists of multiple densely connected layers with rectified linear unit (ReLU) activation functions.
Batch normalization layers are inserted after each dense layer to stabilize and accelerate the training process. Dropout layers are incorporated to prevent overfitting. Early stopping is employed to halt training if the validation loss does not improve for a specified number of epochs (patience).
ReduceLROnPlateau is utilized to decrease the learning rate of the validation loss plateaus, which helps in fine-tuning the model's convergence.

We tried various types of architectures but the neural network’s recall was coming low as compared to XGBoost and the overall performance was also below par with XGBoost.

Below is the training and validation loss.



10. Metrics Used:

We utilized several key metrics to evaluate the performance of our classification models. These metrics provide insights into different aspects of model performance, including its ability to correctly classify instances of each class and its overall discriminative power.

a) **Precision**
b) **Recall**
c) **F1-Score**
d) **ROC-AUC score**
e) **Confusion Matrix**

11. Conclusion :

Model	Precision_1	Recall_1	F1-Score	ROC-AUC score
Random Forest	0.88	0.92	0.95	0.958
Vanilla XGBoost	0.92	0.91	0.96	0.955
Hyper XGBoost	0.91	0.92	0.96	0.956
HyperCV XGBoost	0.91	0.93	0.96	0.999
Deep Neural Network	0.88	0.91	0.95	0.955

We conclude that out of all the models that we trained, HyperCV XGBoost was the best performing model. We used that model on the validation data set that was provided to us for the final submission.