Bank Account Fraud Detection

Final Report of DSC148-WI24

Notebook: https://github.com/acai1031/Bank_Account_Fraud_Detection.git

Meiting Wu
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, California
mew013@ucsd.edu

Lin Zhou
Department of Mathematics
University of California, San Diego
La Jolla, California
liz009@ucsd.edu

Yishan Cai
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, California
yic075@ucsd.edu

Introduction

In the intricate world of the financial system, safeguarding against fraudulent activity is paramount for both institutions and consumers. Bank account fraud presents dynamic and sophisticated challenges that continually evolve with technological advancements. The prediction and detection of fraudulent attempts are crucial to protect the financial ecosystem from the pernicious effects of such activities.

This project is focused on a predictive model that can identify subtle indicators of fraudulent behavior within banking operations. Utilizing the Bank Account Fraud (BAF) dataset, which was published at NeurIPS 2022[1] and is available through the Kaggle [2]website, we aim to construct a model that engages with a rich bank account data, crafted to reflect real-world circumstances. The dataset is a vital resource for developing machine learning algorithms, fostering more robust fraud detection techniques, and contributing to enhanced security within the banking sector.

1. Dataset

1.1 Identify Dataset

The Bank Account Fraud (BAF) dataset, sourced from the Kaggle platform, closely resembles real-world data encountered in bank fraud detection. The dataset is carefully curated to challenge and validate the efficacy of machine learning (ML).

The BAF dataset is characterized by:

- **Realism**: It simulates the intricate details of actual fraud detection datasets, providing a realistic foundation for ML models to interact with and learn from.
- **Imbalance**: Same as real-world data distribution, fraudulent instances are significantly less prevalent, creating a rigorous test scenario for the predictive model.
- **Dynamism**: Including temporal data and distribution changes, the dataset captures the evolving nature of fraud tactics.
- **Privacy Protection**: Differential privacy measures, including noise addition and feature encoding, are applied to protect any real individual's identity. A generative model, the Conditional Tabular Generative Adversarial Network (CTGAN), has been utilized to produce data that is representative.

Our work with the BAF dataset is engineered to develop a predictive model that is adept at detecting potential fraud while addressing the complexities and ethical implications intrinsic to real-world financial data.

1.2 EDA

Basic Statistics

Comprising 1,000,000 instances, each with 32 attributes, our dataset encompasses multifaceted features essential for the detection and prediction of bank account fraud. These features encapsulate various aspects of an applicant's information, such as financial background indicated by "income", demographic details exemplified by "customer_age", and behavioral data through attributes like "days_since_request". Additionally, the dataset includes transactional characteristics such as "zip_count_4w", representing regional application frequencies, and "credit_risk_score", which provides an assessment of the risk associated with each application. We combine these features to offer a comprehensive profile of each bank account application, serving as a practical resource for developing a deep understanding of fraudulent account application activities.

[1] https://arxiv.org/abs/2211.13358

[2] https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022/

Data Type

Upon reviewing the data type distribution in our dataset, it's apparent that integer-based numerical features predominate, comprising both int64 and float64 types. The int64 type is the most prevalent, suggesting a significant presence of features with discrete numerical values, which could include counts, identifiers, or binary variables, such as "fraud_bool", “credit_risk_score” and so on. The presence of float64 types indicates quantitative features that offer a spectrum of data, possibly representing measurements, probabilities, or other continuously varying quantities like “income” or “velocity”.

Meanwhile, the object type typically consists of categorical or textual data, representing a smaller portion of our features. These categorical features will require encoding to numerical values, as most machine learning models are designed to handle numerical input. This conversion is critical to enable the application of algorithms such as logistic regression and XGBoost, which we aim to compare in our analysis. To facilitate this comparison and maintain model compatibility, we’ll convert these object-type features into a numerical format through appropriate encoding techniques in the next step.

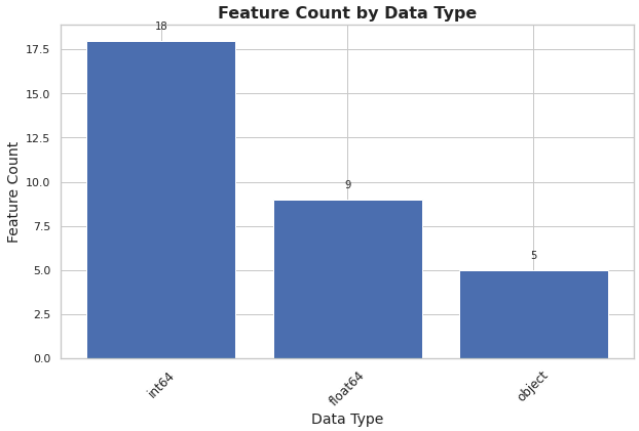

Figure 1: **Feature Count By Data Type**

Checking Missing Value

Obviously, there are no missing values across the entire dataset. This absence of missing data significantly streamlines the preprocessing phase and allows us to directly proceed with analyzing and modeling the data. The completeness of the dataset implies that the data provided by Kaggle may have undergone preliminary preprocessing. This preprocessing likely included steps to fill in or remove any missing values, ensuring the dataset's integrity and usability for machine learning tasks. Such meticulous preparation enhances the reliability of the dataset, allowing us to focus on deriving insights and building predictive models without the preliminary concern of handling missing data.

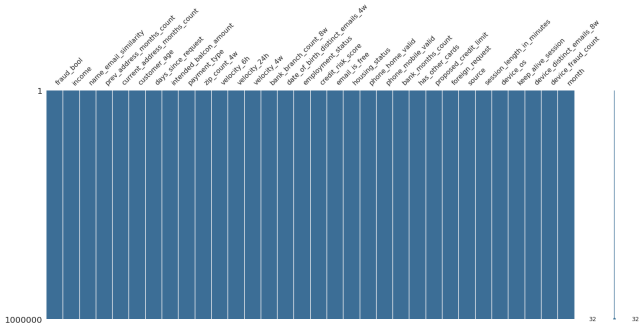

Figure 2: **Table of Checking Missing Value**

Interesting Findings in EDA

We explored most of the columns in the dataset. Some distributions of features disclose meaningful trends and imply the majority range.

zip_count_4w provides the distribution of bank account applications and can potentially disclose trends indicative of fraud. A univariate analysis of this feature reveals a right-skewed distribution, as shown in Figure 3. The majority of ZIP code areas have a relatively low number of applications, with the frequency sharply declining as the application count increases.

The peak, near 1000, suggests that most ZIP codes see only a handful of applications over a four-week period. However, there is a long tail extending towards the higher application counts, which may indicate that a few ZIP codes have a significantly higher number of applications. This could be due to a high population density in those areas or potentially suspicious activity if the numbers are anomalously high relative to regional norms.

For fraud detection purposes, ZIP codes with an exceptionally high **zip_count_4w** could warrant further investigation to determine whether these are hotspots for fraudulent applications. Analyzing the temporal trends of application volumes could also help identify any sudden spikes in application activity, which could be symptomatic of emerging fraud threats.

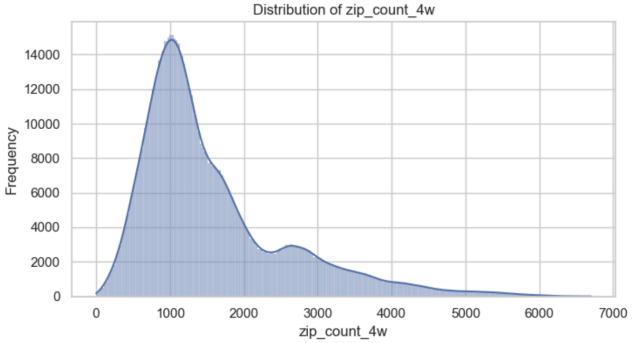

Figure 3: **Distribution of Application Count of Zipcode during 4 Weeks**

credit_risk_score quantifies the perceived risk of an application and is pivotal in fraud detection algorithms. In our dataset, it spans from -176 to 387, with a noticeable peak around the lower end of the positive spectrum. The distribution suggests that most applicants have a moderate risk score, with fewer instances falling into the high-risk category.

This feature's distribution demonstrates anomalies in this feature, such as extremely low and high scores or unusual patterns in the distribution, could be leveraged to pinpoint potential fraud cases.

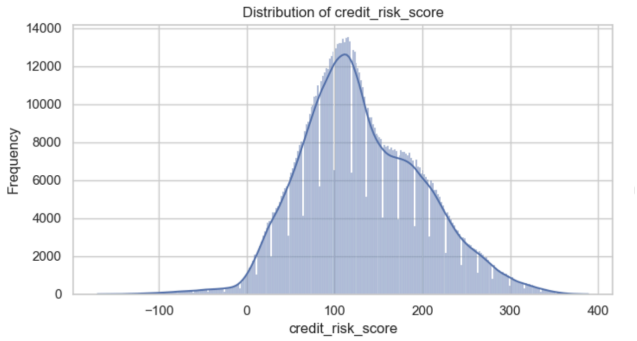

Figure 4: **Distribution of Credit Risk Score per Application**

1.3 Data Processing

One-Hot Encoding

We conducted one-hot encoding to convert categorical variables into a numerical format suitable for machine learning algorithms. We defined a function, get_dummies, to process columns ("payment_type", "employment_status", "housing_status", "source", and "device_os"), generating new columns for each category within these features. Each category was encoded as a series of columns with binary values reflecting the presence of each category in the original data. This expanded dataframe maintained clear labeling for interpretability and became more readable for machine learning applications.

Evaluating Skewness and Logarithmic Scaling

In the evaluation of skewness for our dataset, we only focus on the numerical dataframe. There are approximately 12 features, which apparently skew strictly larger than our threshold 1.

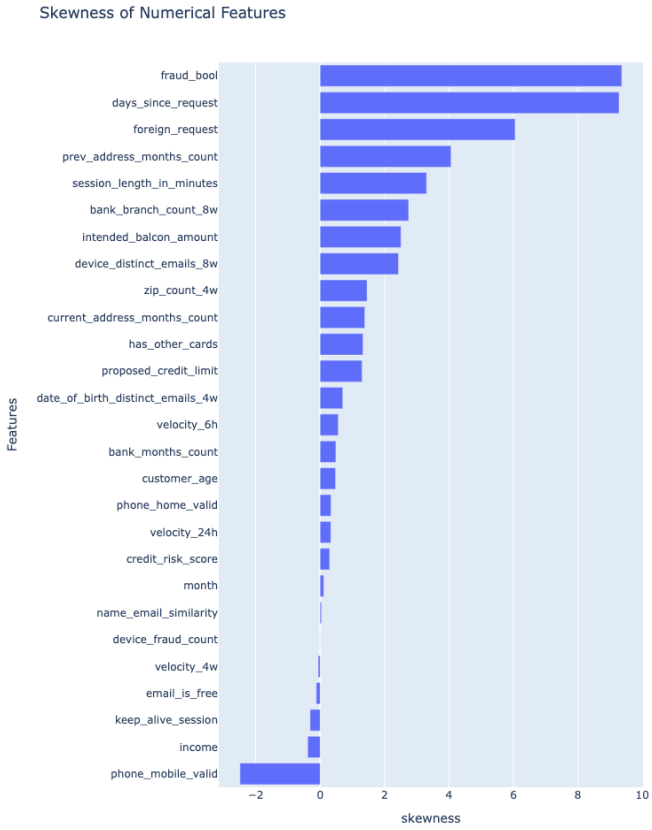

Figure 5: **Table of Checking Skewness**

We have selected the variables "proposed_credit_limit", "zip_count_4w", and "days_since_request" for logarithmic transformation due to their positively skewed distributions. It is essential to address this characteristic for specific statistical analyses that presume data normality. The chosen variables are suitable for log transformation as they do not contain boolean values or negative numbers, which could complicate the transformation process. Other variables exhibiting substantial skewness are not candidates for this transformation, either due to their boolean nature or the presence of negative values. Correcting skewness through transformations such as logging shows that post-logarithmic scaling reveals a marked improvement in the distribution of the data. The skewness value has decreased substantially from the previous one, suggesting a shift toward a more symmetric distribution.

```
Normality test for days_since_request:
--------------------------------------
Prev_Skewness: 9.278940672399802
Skewness: 9.278940672399802
Kurtosis: 106.56921373544326
--------------------------------------

Normality test for zip_count_4w:
--------------------------------------
Prev_Skewness: 1.4566544516033453
Skewness: 1.4566544516033453
Kurtosis: 2.139983496302856
--------------------------------------

Normality test for proposed_credit_limit:
--------------------------------------
Prev_Skewness: 1.301408024786162
Skewness: 1.301408024786162
Kurtosis: 0.16883858077752167
--------------------------------------
```

Figure 6: **Table of Checking Skewness**

Drop High Correlated Features

We proceeded to address potential multicollinearity in our dataset. Multicollinearity occurs when two or more variables are highly correlated and can introduce redundancy into our predictive models, possibly distorting the results.

We began by constructing a correlation matrix using the corr function, which computes the Pearson correlation coefficients between every pair of features. To visualize this matrix, we use Seaborn's heatmap function.

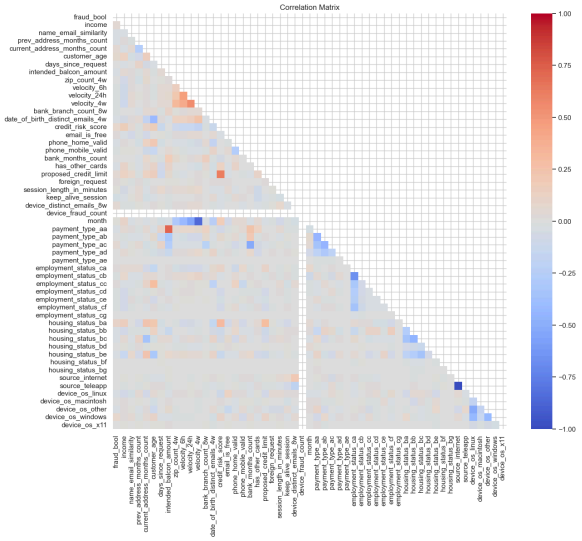

Figure 7: **Heat Map between Each Features**

We observe several features with a correlation higher than our threshold of 0.7. To systematically identify these pairs, we implemented a function, group_related_features, which iterated through the absolute values of the correlation matrix to find and group features with correlations exceeding the threshold.

We got three correlated feature pairs, ("intended_balcon_amount", "payment_type_aa"), ("velocity_4w", "month"), ("source_internet", "source_teleapp"), and then decide to drop one feature from each pair to reduce redundancy ("payment_type_aa", "velocity_4w", "source_teleapp were dropped). Additionally, we remove the device_fraud_count feature after noticing it only contained zeros, which would not contribute meaningful information to our models.

Dealing with Imbalanced Data

The Credit Card Fraud dataset illustrates a pronounced class imbalance, with an explicit contrast in the occurrence of non-fraudulent account applications (988,971) compared to fraudulent account applications (11,029). This significant discrepancy in distribution poses a challenge for predictive modeling, as it can bias the model towards the majority class, leading to poor generalization on the minority class.

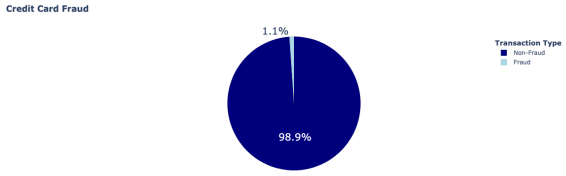

Figure 8: **Pie Chart for Binary Class Distribution before dealing with balance**

To enhance model performance and particularly improve its capability in detecting fraudulent account applications, we have decided to construct a sample of our data with a total of 50,000 instances. This sample size is substantial enough to provide our predictive models with sufficient data to learn from while managing computational efficiency. Within this sample, we will retain the entire 11,209 instances, labeled as fraud to preserve the crucial patterns they represent. To complement this, we would perform random sampling from the non-fraudulent instances to select 38,791 records, ensuring that our final dataset maintains a relatively balanced representation of both classes for robust model training.

In addition to the main analysis, we aim to show a demo at the end of our project. Given that our data adheres to strict privacy preservation protocols, generating additional data for this demo presents a challenge. To circumvent this, our strategy is to meticulously extract 50 instances from each class, resulting in a total of 100 instances for the demo. These instances will serve as a representative but concise subset, demonstrating the predictive

model's capabilities in a controlled setting. Subsequently, the remaining data will be combined, randomized and utilized as the training and testing dataset to further develop and evaluate our mode's performance.
Here is the improving distribution:

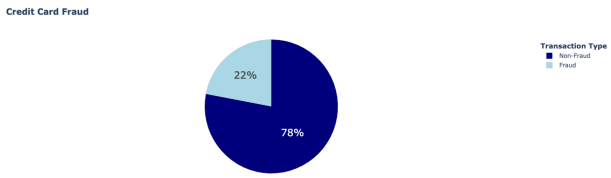

Figure 9: **Pie Chart for Binary Class Distribution after dealing with balance**

2. Predictive Task

2.1 Predictive Task

After basic cleaning, engineering, **given the basic information about the client, predicts the fraud_bool of the given client**. This is a binary classification problem.

2.2 Evaluation

The Area Under the Receiver Operating Characteristic Curve (ROC-AUC) stands as a pivotal evaluation metric for assessing the performance of classification models. Its value, ranging from 0 to 1, reflects a model's capability to correctly distinguish between classes—in this case, fraudulent and non-fraudulent transactions. The closer the ROC-AUC is to 1, the more proficient the model is at making these distinctions.[3]
In addition to ROC-AUC, Recall and the F1 score play crucial roles in evaluating our model's effectiveness. Recall, or the true positive rate, quantifies the proportion of actual fraud cases accurately identified by the model. Given the significant financial implications of fraud, a high recall rate is essential for pinpointing true positives effectively.
The F1 score, as the harmonic mean of Precision and Recall, provides a balanced measure of the model's precision (the accuracy of fraud predictions) and its recall.[4] This balance is crucial to avoid a disproportionate bias towards either detecting fraud (at the risk of mislabeling legitimate transactions as fraudulent) or overlooking fraudulent transactions to minimize false positives.

Generalization

To safeguard the generalizability of our model, we have divided our resampled fraud dataset into training and testing subsets, allocating 75% to training and 25% to testing. This approach ensures that if our model—tailored through feature selection, engineering, and hyperparameter tuning on the training dataset—exhibits strong performance on the testing dataset, it is well-equipped to predict fraud in a broader array of scenarios without overfitting to the training data

2.3 Baseline

Baseline dataset: randomized sample dataset
Baseline involves features: income, proposed_credit_limit, zip_count_4w, days_since_request
Four models are used as baseline: Logistic Regression, Decision Tree, XGBoost, and Catboost

Logistic Regression is a fundamental classification algorithm that predicts the probability of a binary outcome based on one or more independent variables. It models the relationship between the features and the probability of the target variable being in a specific class. This simplicity makes it highly interpretable, less prone to overfitting with proper regularization, and a solid baseline for evaluating more complex models. It is also particularly effective for binary classification problems, which should work well on predicting fraud_bool.[5]

Decision Tree is a versatile supervised learning algorithm that can be used for both classification and regression tasks. It works by splitting the data into subsets based on the value of input features, making it easy to understand and interpret. Decision Trees can capture nonlinear relationships between features and the target variable by dividing the space into smaller subsets.[6]

XGBoost (Extreme Gradient Boosting) utilizes boosting and gradient boosting techniques to combine weak learners into a strong model, optimizing the loss function to minimize errors. It uses a technique called tree pruning to control the complexity of decision trees, preventing them from growing too deep and overfitting the training data.

CatBoost is also based on Gradient Boosting like GBM, it ensembles multiple decision trees and uses gradient boosting to minimize prediction error. It is designed to provide a balance between speed and accuracy, making it highly effective for large datasets and complex problems where categorical data plays a significant role.[7]

Baseline models are trained on the same training set of the same features and the performance is evaluated using AUC(Area Under

---

3 https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
4 https://builtin.com/data-science/precision-and-recall
5 https://www.ibm.com/topics/logistic-regression
6 https://www.geeksforgeeks.org/decision-tree/
7 https://catboost.ai/

Receiver Operating Characteristic Curve) for predicting the same set of test data.

Baseline Result

XGBoost and Catboost (average AUC, Recall, F1 > 0.55) perform significantly better than Logistics Regression and Decision Tree (average AUC, Recall, F1 < 0.4) on the same testing set.

It's important to highlight that Logistic Regression's performance falls below our expectations compared to all other models. It exhibits low recall and F1-score, indicating its ineffectiveness in accurately identifying positive cases. However, the AUC score, while relatively reasonable, implies that the model can distinguish between positive and negative classes better than random chance, yet there's room for improvement in terms of precision and recall. This discrepancy in performance could be attributed to non-linear separable data and complex relationships, including higher-order interactions among features.

Among the two advanced models, the AUC, Recall, F1 scores of CatBoost are all above that of XGBoost. CatBoost also has the highest AUC score, indicating superior class separation. Therefore, XGBoost and CatBoost are selected as the preferred models for further optimization and fine-tuning to enhance their predictive capabilities.

Set	recall	f1	roc_auc
LogisticRegression Baseline	0.013557	0.026611	0.690803
DecisionTree Baseline	0.316090	0.315471	0.558376
XGBoost Baseline	0.561541	0.446335	0.688591
CatBoost Baseline	0.602212	0.464566	0.705721

Figure 10: **Table of combination of Baseline Metrics**

The data presented in the confusion matrix offers a comprehensive breakdown of the model's predictions, facilitating a thorough evaluation of its effectiveness. To enhance detection of fraudulent activities, our aim is to increase the instances correctly identified as true positives (true = 1, predicted = 1). At the same time, we seek to reduce the occurrences of false negatives, where true = 1 but predicted = 0, which incorrectly label fraudulent applications as normal.

True Positives (TP) represents the number of correctly predicted positive cases. CatBoost has the highest number of true positives (1688), followed by XGBoost (1574), and Decision Tree (886). Therefore, CatBoost performs best in correctly identifying positive cases, followed by XGBoost and then Decision Tree. In contrast, speaking of False Negatives (FN), it represents the number of positive cases incorrectly predicted as negative. Decision Tree has the highest number of false negatives (1917), followed by XGBoost (1229), and CatBoost (1115). A lower value of false negatives is desirable, indicating fewer positive cases being incorrectly classified as negative. Upon analysis, we may note that both the Decision Tree and Catboost models exhibited better performance compared to Decision Tree in this context.

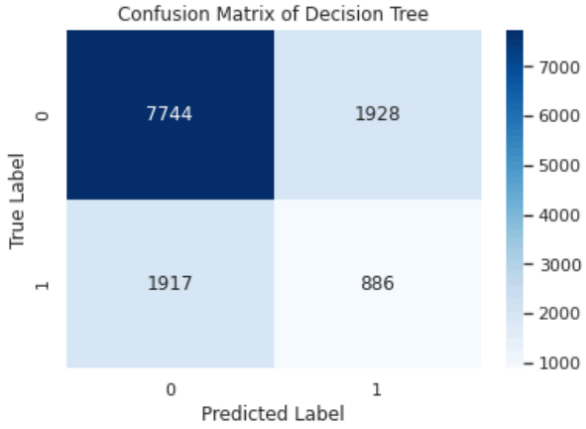

Figure 11: **Baseline Confusion Matrix of Decision Tree**

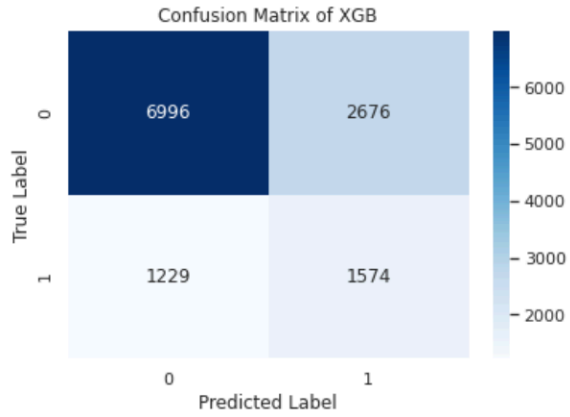

Figure 12: **Baseline Confusion Matrix of XGB**

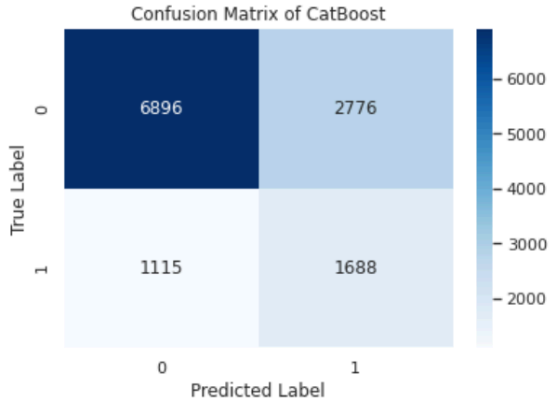

Figure 13: **Baseline Confusion Matrix of CatBoost**

3. Model

Initial Setting
Dataset: randomized sample dataset, same as baseline
Initial features: same as features used in baselines and top 10 importance features identified below
XGBoost Classifier and CatBoost Classifier are the model packages we choose to further optimize.

3.1 Important Features Identification

Since we want to learn the real correlation between features and "fraud_bool", we decided to use the dataframe before balancing. Certain features demonstrate a stronger correlation with the fraud_bool target variable. This implies that these features might have a more significant role in predicting fraudulent bank account applications.
Particularly, "housing_status_BA" and "device_os_windows" show a relatively higher correlation with the target variable. These insights suggest that there may be specific patterns within these features that are associated with fraudulent activity.
For "housing_status_BA", it could indicate that the housing status provided by the applicant may contain specific trends or anomalies that correlate with fraudulent behavior. It is advisable to further investigate the nature of these correlations. For instance, applicants from certain housing statuses might be more likely to submit fraudulent applications.
Similarly, "device_os_windows" indicating a strong correlation suggests that the operating system of the device used for the application might be a factor in fraud. It could imply that applications made from devices with Windows OS have different fraud patterns compared to other operating systems. It is crucial to delve deeper into this feature to understand if there is a particular vulnerability or pattern associated with Windows OS that could be exploited for fraudulent purposes.

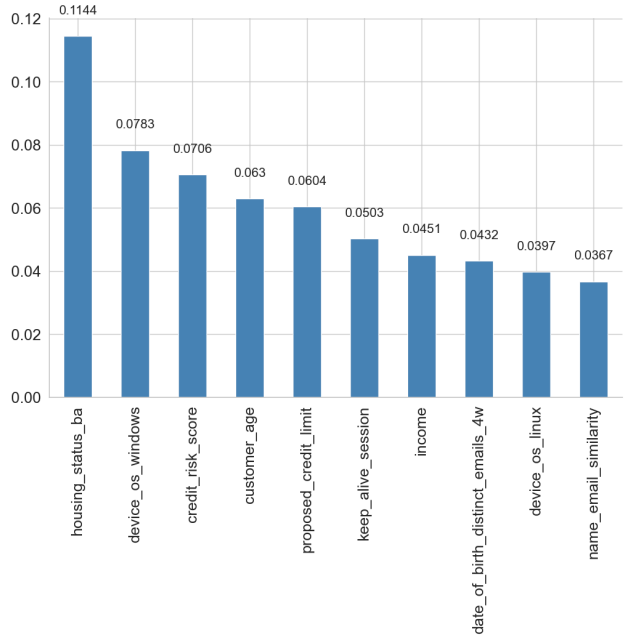

Figure 14: **Bar Chart of Top 10 Relative Features**

3.2 Hyperparameter Tuning

We conducted hyperparameter tuning on both prediction models to achieve locally optimized performance on the testing set. We utilized GridSearch with 5-fold cross-validation to identify the optimal hyperparameters.
For the XGBoost Classifier, with learning_rate = 0.3, max_depth = 6, n_estimators = 90, and reg_lambda = 7, the performance of the XGBoost model improved significantly. Specifically, the AUC score on the testing set increased from 0.689 to 0.834. Meanwhile, for the CatBoost Classifier, setting depth = 6, iterations = 300, l2_leaf_reg = 0.7, and learning_rate = 0.1 led to substantial improvements in model performance. The AUC score improved from 0.706 to 0.846 on the testing set.

3.3 Model Evaluation

Both XGBoost and CatBoost models have shown significant improvements in recall, F1 score, and AUC from their baseline to final versions. CatBoost consistently outperforms XGBoost in terms of recall, F1 score, and AUC, indicating its superiority in capturing positive cases, achieving a balance between precision and recall, and distinguishing between classes effectively.
CatBoost is a gradient boosting algorithm that excels in handling complex datasets and capturing intricate relationships between features. After hyperparameter tuning, the CatBoost model may have been optimized to strike a better balance between precision and recall, leading to a higher F1-score. Additionally, CatBoost's inherent capability to handle categorical variables and perform feature importance analysis could contribute to improved ROC-AUC.

Set	XGBoost Baseline	CatBoost Baseline	XGBoost Final	CatBoost Final
recall	0.561541	0.602212	0.703532	0.731002
f1	0.446335	0.464566	0.600945	0.609549
roc_auc	0.688591	0.705721	0.834447	0.845844

Figure 15: **Table of combination of Final Model Metrics**

Moreover, the examination of the confusion matrix reveals the improvement of prediction. In order to identify fraudulent applications, we aim to maximize both true positives (correctly identified fraudulent applications) and false negatives (fraudulent applications classified as non-fraudulent). Both XGBoost and CatBoost at their final stages have a relatively high number of true positives, with CatBoost having slightly more (2049) compared to XGBoost (1972). In terms of False Negatives, XGBoost has 831 false negatives, while CatBoost has 754 false negatives. These represent fraudulent applications incorrectly classified as non-fraudulent, which can pose a significant risk if not detected.
Considering the balance between precision (minimizing false positives) and recall (minimizing false negatives), CatBoost seems to have a slight advantage by exhibiting fewer instances of false positives and false negatives in contrast to XGBoost. Therefore, CatBoost may be preferred for its better ability to identify both fraudulent and non-fraudulent applications accurately in the context of bank account opening fraud detection.

Overall, our model demonstrates enhanced generalization to unseen data according to our evaluation metrics, aligning with our goal for improvement supported by measurable metrics.

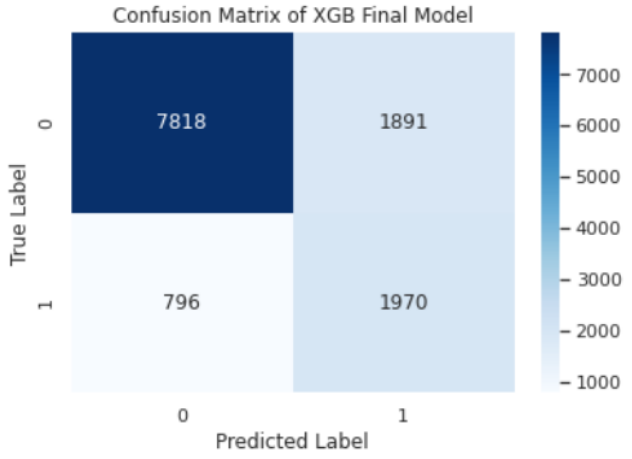

Figure 16: **Final Model Confusion Matrix of XGB**

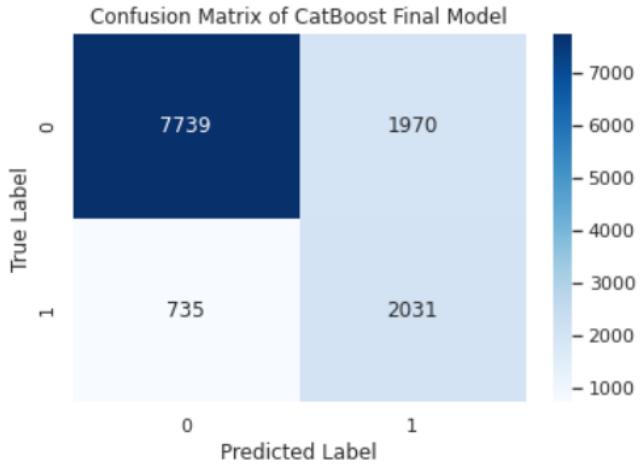

Figure 17: **Final Model Confusion Matrix of CatBoost**

4. Literature

Bank Account Fraud by Sérgio Jesus

https://github.com/feedzai/bank-account-fraud/tree/main

Our project draws inspiration from Sérgio's work on the "Bank Account Fraud" project, where we both began with an identical fraud dataset. Sérgio's methodology involved the use of LightGBM and CTGAN models to predict fraudulent transactions, marked as fraud_bool in the dataset. Through a meticulous process of evaluating 200 hyperparameter configurations via random search, he identified and utilized the thirty most impactful features determined by the LightGBM models. Following this feature selection, he proceeded to train a CTGAN model, leveraging the insights gained from the identified features.

In addition to employing LightGBM models, Sérgio integrated Logistic Regression and Decision Trees as baseline models to anchor his predictions. A notable aspect of his approach was the implementation of differential privacy techniques, specifically the addition of Laplacian noise to the dataset, enhancing data privacy and security without significantly compromising data utility.

While our project aligns with Sérgio's in terms of the foundational dataset and the overarching goal of fraud prediction, we diverge in our methodologies, particularly regarding data splitting strategies. Sérgio opted for a temporal split, leveraging the month of transaction as a criterion for dividing the dataset into training and testing sets. This strategy, coupled with his thoughtful selection of performance and fairness metrics, underscored the nuanced and contextually aware approach to model evaluation. Learning from Sérgio's project, we applied similar models for feature selection in our own analysis. His project served as a valuable blueprint, guiding us in refining our feature selection process.

Bank Fraud Detection Using GBM by Juan José

https://www.kaggle.com/code/juanjosmorenogiraldo/bank-fraud-detection-using-gbm/notebook#Table-of-Contents

Juan's project, utilizing the same dataset as ours, placed a significant emphasis on exploring the distribution of each feature and investigating their correlations with fraud_bool, the target variable we aimed to predict. Our project mirrored a similar EDA process to Juan's, shedding light on the patterns within our dataset. However, notable distinctions arose in our approaches to model selection and addressing data imbalance. Juan employed the Synthetic Minority Over-sampling Technique (SMOTE) from the imblearn.over_sampling package to generate synthetic instances, effectively countering the issue of imbalanced data without merely duplicating minority class samples, which could heighten the risk of overfitting. Conversely, we opted to streamline our dataset to 50,000 observations, aiming to mitigate the impact of imbalanced classification. Moreover, Juan selected Gradient Boosting Machine (GBM), Logistic Regression, and Decision Tree models for his analysis, achieving marginally superior performance with recall, F1 score, and ROC-AUC values approximately reaching 0.9. This disparity in outcomes can likely be attributed to the different strategies employed in handling data imbalance. Through Juan's project, we gained valuable insights into conducting and interpreting EDA within the context of fraud detection datasets.

5. Result

5.1 Comparison

Both XGBoost and Catboost outperformed the baseline models for both train and test set. Recall that we split the dataset into two subset: train and test to ensure the generalization of our model. All feature engineering and model optimization are done on the train dataset, and the performance is confirmed in the test dataset.

After discovering the real correlation between features and "fraud_bool", some features demonstrate a stronger correlation with the fraud_bool target variable. After trying adding these

features into our model, and performing hyperparameter tuning, for XGBoost, we have improved the recall by 32.87%, F1 score by 44.14%, ROC-AUC by 24.57%, and for CatBoost, we have improved the recall by 27.53%, F1 score by 36.40%, ROC-AUC by 22.26%. Notice that for both CatBoost and XGBoost, we found that “housing_status_BA” and “device_os_windows” are the most helpful features in addition to the baseline features. Our models can outperform the baseline because we have more features like “housing_status_BA” and “device_os_windows”, and we use hyperparameter tuning to prevent overfit. The gap of the difference in performance is significant, since we improved at least 27% of the recall, 36% of the F1 score, and 22% of the ROC-AUC for both datasets and models.

5.2 Effectiveness

Despite incorporating numerous features into our final model beyond those used in the baseline models, not every feature contributed positively to performance enhancement. Our meticulous approach of individually integrating features under consistent hyperparameters revealed that certain attributes, such as “velocity_6h” and “email_is_free”, did not significantly bolster the model's efficacy on their own. Nonetheless, the synergistic effect of most added feature combinations notably surpassed the baseline model's performance, underscoring the importance of strategic feature selection in optimizing our predictive capabilities.

5.3 Hyperparameter

For the XGBoost, we used n_estimators = 90 to provide a balance between model complexity and computational efficiency, and max_depth = 6 and reg_lambda = 7 to prevent overfit. For CatBoost, we used iterations = 300 to strike a balance between learning the data intricacies and computational demand, and depth = 6 and l2_leaf_reg = 0.7 to prevent overfit, and learning_rate = 0.1 to guide the model towards optimal performance efficiently. All these numbers are obtained from GridSearchCV.

5.4 Major Takeaways

1. Common features like “housing_status_BA”, “income”, and “credit_risk_score” are very helpful with predicting fraud_bool.
2. The features that improved the performance of both models are the same. This might be because all of these features have relatively high correlation with “fraud_bool”, and the combination of these features provides meaningful information towards “fraud_bool”.
3. During the model optimization phase, we meticulously conducted controlled experiments. For each iteration, we varied either the features, hyperparameters, or model types as independent variables to observe their impact on Recall, F1 score, and ROC-AUC metrics. This systematic approach provided deeper insights into the nuances of data analysis and the intricacies of model development, allowing us to refine our strategies and enhance model performance effectively.

ACKNOWLEDGMENTS

In our final report on Bank Account Fraud Detection, we convey our deepest appreciation to our professor and the esteemed faculty at the University of California, San Diego, for their expert guidance and steadfast support, which were instrumental in our research. Our gratitude extends to Kaggle and the authors of the Bank Account Fraud dataset for furnishing us with an essential resource that greatly facilitated our work. Special thanks is owed to Sérgio Jesus, whose project on Bank Account Fraud provided significant inspiration and deepened our understanding of feature analysis. This project stands as a testament to the collective knowledge and the spirit of collaboration of everyone involved, for which we are deeply thankful.

REFERENCES

[1] Bank Account Fraud by Sérgio Jesus. https://github.com/feedzai/bank-account-fraud/tree/main

[2] Bank Fraud Detection Using GBM by JUAN JOSÉ https://www.kaggle.com/code/juanjosmorenogiraldo/bank-fraud-detection-using-gbm/notebook#Table-of-Contents