Detecting bank account application fraud using supervised binary classification models

Michael Nguyen Jr
Team Leader
mjrnguyen@ucdavis.edu

Albin Franzén
alfranzen@ucdavis.edu

Nidhi
nnoname@ucdavis.edu

Isabel Shic
email@ucdavis.edu

Ruibin Zhu
email@ucdavis.edu

***Abstract*—Fraudulent bank accounts are a problem for many banks since they lose money from needing to manually detect and remove the fraudulent accounts before they apply for fraudulent loans or allow for government regulation to intervene with penalties. In this paper we propose using supervised machine learning for automatic detection of fraudulent bank fraud applications and achieve an AUCROC score of 0.89 using an XGBoost model. We present the exploratory data analysis and preprocessing required to achieve such results, leading to the development of a user interface for testing new observations.**

***Index Terms*—bank fraud detection, binary classification,**

I. Introduction and background

Every year many bank account applications are processed which is a problem for banks because they lose money on taking care of the bank accounts and need to adhere to financial and reputational losses [1]. Fraudulent bank accounts cause banks to lose economic profits from unrecoverable loans, penalties and regulatory fines so it is important for the reduction of such accounts [2]. The banks also lose trust and reputation customers and regulators which could cause loss of consumers [2].

There are various methods to detect bank fraud and originally the classical methods were to manually detect fraud often after it has occurred which meant that the banks had to incur a loss [3]. Employees had to be specially trained to sift through data which was inefficient and cumbersome to the banking institutions[3]. With the advent of more data and computational power it could be shown that fraudulent bank account applications contain certain characteristics which can be used to distinguish them from real applications [4]. This led to algorithms being developed and especially in the modern era machine learning has been applied for bank fraud detection [5].

There are various methods to use machine learning for fraud detection and they can be summarized as supervised methods, unsupervised methods and semi-supervised methods [6]. Supervised methods create a dataset of application data with labels of whether the account is fraudulent or not and a model is trained on this data. Semi-supervised models have some labeled data which is often only the non-anomalous class and then the anomalous class or uncertain values aren't labeled. Finally there exists unsupervised learning which rely on techniques for anomaly detection to detect what values adhere from the norm and attempts to create classes from these. Another notable mention that can be used for bank fraud detection are graph based methods that use network analysis to detect fraud and has historically been used to find connections in crime networks.

The difficulty with modeling bank fraud data is that the data is highly unbalanced with very few fraudulent cases and very many normal cases. This means that the models become very sensitive to the distribution of the training data because a single user could create many fraudulent applications that would make the training data very inaccurate [7]. Another consideration to be taken into account is what features are be required to model the application information which requires extensive exploratory data analysis and feature engineering.

In this work we use the NeurIPS 2022 bank fraud detection dataset which is a famous dataset from the NeurIPS conference to compare different machine learning models and their capabilities [8]. We chose this dataset because it contains a large variety of 32 features with many different properties and with a lot of preexisting work it would be simple to compare our designed models with the current state of the art. Our plan is to do a more extensive data exploration than anything seen before with an individual analysis of all features and in depth comparison between them. We use various visualisation techniques and unlike previous papers we also use hypothesis testing on our potential feature transformations to determine which ones to keep.

After the data preprocessing is complete we demonstrate how sklearn pipelines can effectively be used to comprise the data preprocessing into distinct steps and combine them as a single entity. After the preprocessing is complete we test a plethora of different machine learning models using a random search and then we fine-tune them with a grid search using automl tools. Their accuracy is measured with the area under curve for the receiver operating characteristic (AUCROC) and can be compared to decide on the best one. Due to the preprocessing being made with sklearn pipelines it gives the possibility of the preprocessing to also be random/grid searched if necessary and allow a seamless integration into the user interface developed in flask to present the final model. The final model is displayed with a user interface that allows users to input gathered data and the program outputs whether that instance is a fraudulent bank account application or not.

II. Literature Review

Bank fraud detection has been a critical area of research due to the financial and reputational risks associated with fraudulent activities. A comprehensive understanding of the detection mechanisms for bank account application fraud is essential to establish the study's contribution to this domain. Existing research demonstrates a variety of algorithmic solutions, particularly supervised binary classification models, anomaly detection, and hybrid techniques, each offering unique strengths in combating fraudulent activities.

Supervised learning techniques rely on labeled datasets to identify fraudulent transactions or applications. Common algorithms include logistic regression (LR), decision trees (DT), support vector machines (SVM), and ensemble models like XGBoost. Neural networks, particularly convolutional neural networks (CNNs) and long short-term memory networks (LSTMs), have proven effective in capturing sequential patterns in temporal data despite their higher computational demands. For example, studies demonstrate the effectiveness of SVM combined with recursive feature elimination in enhancing classification accuracy by selecting the most relevant features **"cite** . However, the dependency on labeled datasets makes supervised models less adaptable to novel fraud patterns.

Unsupervised techniques, such as k-means clustering, isolation forests, and autoencoders, are effective for detecting anomalies in unlabeled datasets. Semi-supervised approaches, such as hidden Markov models (HMMs) and generative models, can augment training by synthesizing data, addressing challenges related to data imbalance and scarcity. Recent studies highlight the potential of quantum machine learning (QML) and time series models in capturing complex patterns in specific data structures [3][4]. However, these methods often struggle with interpretability and require careful tuning to avoid false positives.

Hybrid approaches combine supervised and unsupervised techniques to leverage the strengths of both methodologies. For instance, integrating autoencoders with supervised classifiers enhances the model's ability to detect both known and novel fraud patterns. Similarly, graph-based methods, which analyze relationships within financial networks, provide additional context for identifying anomalies. The PLADS algorithm, for example, processes graph streams in batches, using graph pattern deviations to adapt to real-time data [5]. This integration of techniques mitigates the limitations of relying solely on one approach.

Traditional statistical methods, such as ordinary least squares (OLS), autoregression (AR), and logistic regression, serve as foundational techniques for fraud detection. While these methods are computationally efficient and interpretable, they often underperform in high-dimensional or dynamic environments. Risk-based approaches like value-at-risk (VaR) and expected shortfall provide a financial risk perspective but are limited in detecting nuanced fraud patterns [6], [7]. These methods are typically used as supplementary tools alongside advanced machine learning models.

Key challenges in fraud detection include imbalanced datasets, real-time detection, and minimizing the False Positives. Fraudulent instances are significantly outnumbered by legitimate ones, leading to biased model training. Reducing false alarms avoids customer dissatisfaction and operational inefficiency. Evaluation metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC) are commonly used to assess model performance. Recent advancements, such as the 3S Testing framework, address distributional shifts and improve evaluation by generating synthetic test sets that better reflect real-world scenarios [8].

III. Dataset Description and Exploratory Data Analysis

The dataset that we are using is the base dataset of the kaggle NeurIPS bank fraud detection dataset, which includes 1 million instances and 31 features [8]. Each instance is labeled with a binary target variable "fraud_bool", where 1 represents fraudulent applications and 0 represents legitimate applications. We used an 80/20 split for training and testing data so the EDA is done on the training data to avoid data leakage. We start the EDA by testing the frequency of fraud and discover that the frequency is 1.1% which means we have highly unbalanced data which will need to be taken into consideration. Next we discover that we do not have any missing values in the dataset however the dataset description on kaggle says that values that are -1 should be marked as missing. Further exploration of the data reveals that the there exists both numerical and categorical features so we do the EDA separately on both categories.

There are 7 numerical features which are "name_email_similarity", "days_since_request", "intended_balcon_amount", "velocity_6h", "velocity_24h", "velocity_4w" and "session_length_in_minutes". The "name_email_similarity" is a value between 0 and 1 that represents how similar the applicant name is to the email address, "days_since_request" is the number of days passed since the application was done, "intended_balcon_amount" is the initial transferred amount, the velocity features are the average number of applications in the time frame, and "session_length_in_minutes" is the length of the user session on the bank website. After separating these features we look at their statistics and plot their distributions and the distribution of the output variable.

When looking at figures 1 and 2 we see that the feature "name_email_similarity" has a clear distinction between fraudulent and non-fraudulent observations. We can also see that there is a group of perfect similarity, so we decided to convert the feature into a categorical feature with 4 categories of width 0.24 and the one with width 0.04 to capture the high similarity emails. The "days_since_request" feature appear to be highly skewed it would be recommended to apply a log transform and remove outliers and the same goes for the "intended_balcon_amount". One difference is that the feature "intended_balcon_amount" has many missing values which

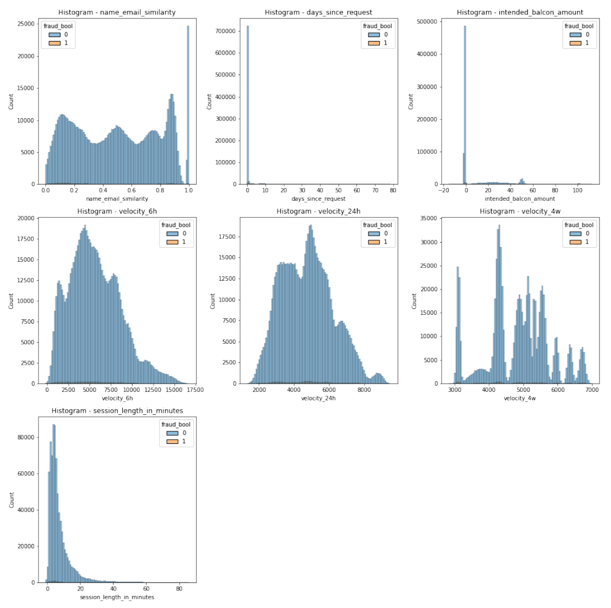


Fig. 1. Histograms of numerical features

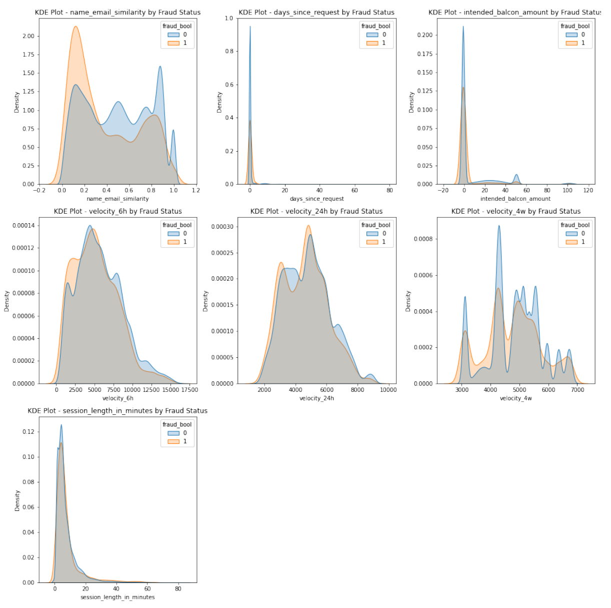


Fig. 2. Kernel density estimates of fraud distribution for numerical features

means that a user hasn't placed money, and when doing a p-value test with the output we discover that users that have placed money into the initial account have a 0.5% chance of being fraudulent rather than the baseline 1.1% so we decided to add a flag feature for missing value and imputing the rest to the mean. The velocity metrics are quite similar to a normal distribution so we standard scale them to the mean and "velocity_6h" has a couple of negative missing values which are imputed to 0. Finally we remove "velocity_4w" because the distribution appears random with no connection to the output in figure 2. The feature "session_length_in_minutes" also doesn't appear to have any have any correlation with the output so we decide to remove the feature. The final step that has to determine possible feature transformations is that we plotted the correlations matrix between the numerical features and there we discover that the velocity features appear correlated but that the rest are very independent which motivates removing one velocity feature.

The next step of the EDA is to explore the categorical features. When comparing the number of categories of the features there appear to be 3 distinct types, high-cardinality features with many categories, low-cardinality features with a few (¡12) categories and boolean categorical features with only 2 categories.

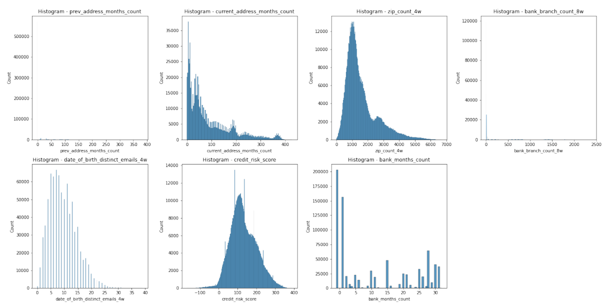


Fig. 3. Histograms of high-cardinality categorical features

In figure 3 there are 7 features of high-cardinality categorical features which are "prev_address_months_count", "current_address_months_count", "zip_count_4w", "bank_branch_count_8w", "date_of_birth_distinct_emails_4w", "credit_risk_score" and "bank_months_count". When doing the same statistics analysis and distribution analysis as for the high-cardinality categorical features we discover that "prev_address_months_count", "current_address_months_count" and "bank_months_count" have statistically significant missing features so we create a flag variable for it and impute the missing values with the means of the columns. The feature "current_address_months_count" is also log transformed due to skewness and for all the features we use standard scaler and remove outliers. Correlation analysis shows that the features are not correlated with each other apart from the address months count and that credit score has the largest influence on the output.

The low-cardinality categorical features in figure 4 are features with less than or equal to 12 categories. The features are "customer_age", "payment_type", "employment_status", "housing_status", "device_os", "device_distinct_emails_8w", "month", "income", "proposed_credit_limit" where the features "customer_age", "income" and "proposed_credit_limt" are ordinal encoded and the rest will be one hot encoded due to the relation between categories . In the EDA we also explore the relationship to the output variable for each category and make inferences that for example a higher proposed credit limit or a higher age gives a higher chance of fraud in figure 5.

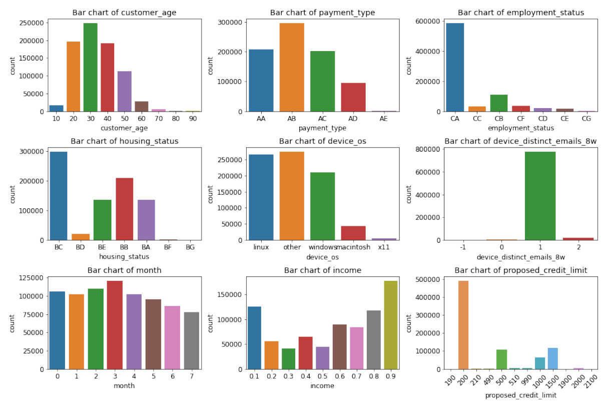


Fig. 4. Bar graphs of low-cardinality categorical features

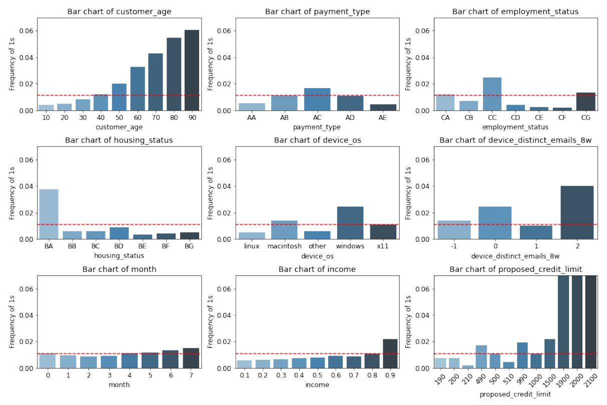


Fig. 5. Bar graphs of fraud distribution low-cardinality categorical features

The final category of categorical features are boolean features which have 2 distinct categories. There are different distributions of the features as seen in figure 6, and in the code for the EDA we also do a similar fraud distribution analysis as figure 5 and a correlation analysis. The takeaway is that all the features appear to be important with significant correlation to the output, and the features "phone_home_valid" and "phone_mobile_valid" are highly correlated, meaning that if one of the numbers is valid, the other one probably also is. This means we can convert it to a single feature of how many valid phone numbers exist. We also combine "foreign_request" and "keep_alive_session" to detect long foreign sessions. Interestingly, the feature "device_fraud_count" appears to have only a single value detected, meaning no fraud has been detected from the same device count. We finish by summarizing all the transformations in the EDA.

IV. METHODOLOGY

In this section, we present the methodology adopted to develop a robust model for detecting fraudulent bank applications. Our approach involves data preprocessing, feature engineering, handling class imbalance, model selection, hyperparameter tuning through manual means and further tuning using Optuna, model training, evaluation, and deployment.

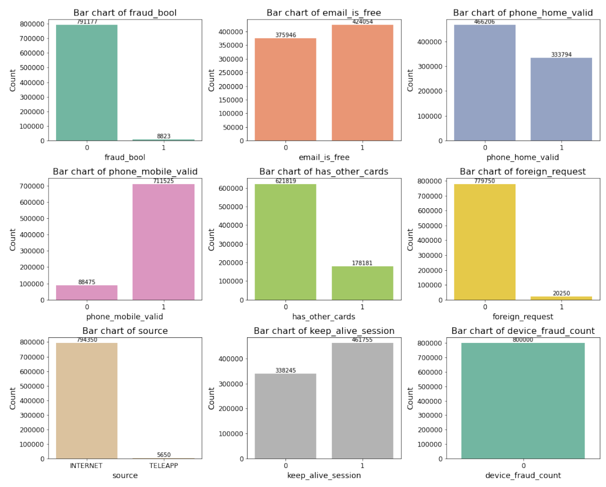


Fig. 6. Bar graphs of boolean features

A. Data Preprocessing and Feature Engineering

The dataset comprises a mix of numerical features, categorical features with varying cardinalities, and boolean variables. To effectively preprocess and engineer features, we categorized them into four distinct groups: numerical features, high-cardinality categorical features, low-cardinality categorical features, and boolean features. This categorization allowed us to customize preprocessing techniques to the specific characteristics of each feature type.

Numerical features were analyzed for missing values, skewness, outliers, and correlation with the target variable. Some key preprocessing steps include missing value imputation, transformation and scaling, feature binning and feature creation. We imputed missing values using the mean or median, and introduced binary indicator variables (missing flags) to preserve information about missingness. We applied log transformations to skewed features (e.g., `days_since_request`) to reduce skewness and approximate normal distributions. We also standardized numerical features to zero mean and unit variance to improve model performance and discretized `name_email_similarity` into categorical bins based on observed thresholds correlating with fraud incidence. Finally we created new features by converting `intended_balcon_amount` into a binary indicator denoting the presence of an initial deposit because of its significant association with fraud likelihood.

For high-cardinality categorical features, such as `credit_risk_score` and `bank_months_count`, we handle missing values, use encoding and select high correlation features. We impute missing categories with a new category (`Unknown`) and add missing flags. We also employ target encoding to replace categories with the mean target value for that category, retaining the base fraud bias while reducing dimensionality. We implemented smoothing to mitigate the impact of rare categories and retained features with strong correlation to the target variable; we considered excluding features with low correlation or high missingness.

For features with fewer categories, such as `employment_status` and `device_os`, we used one-hot encoding, category grouping and ordinal encoding. For one-hot encoding we transformed categorical variables into binary indicator variables to capture non-linear relationships without assuming ordinal relationships. We combined rare categories into an `Other` category to prevent overfitting and reduce model complexity. Another preprocessing step we used was ordinal encoding which was applied to features with inherent order, such as `customer_age`, preserving the ordinal information significant for predicting fraud risk.

Boolean features were processed by combining features and creating aggregated risk indicators. We merged related features (e.g., `phone_home_valid` and `phone_mobile_valid`) into a single feature indicating the total number of valid phone numbers and created (`total_risk_flags`) by summing boolean risk indicators to quantify the overall risk profile.

B. Handling class imbalance

The dataset exhibited significant class imbalance, with fraudulent applications comprising approximately 1% of the data. To address this we adapted our evaluation metric, used models that can handle imbalances but opted out to using resampling techniques like SMOTE due to them not giving higher accuracy. We used Recall @5% FPR alongside the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) to evaluate model performance. While AUC-ROC provides an overall measure of the trade-off between true positives and false positives, Recall @5% FPR focuses on the model's ability to detect fraud while keeping the False Positive Rate at a manageable level. This is done instant of using a simple accuracy metric because a constant negative class classifier with this dataset could achieve over 98% accuracy but fail to identify any fraud cases. We used models that could handle class imbalance intrinsically, such as XGBoost and LightGBM, with parameters like `scale_pos_weight` adjusted to emphasize minority class learning. We considered but ultimately opted against oversampling or undersampling due to potential overfitting and information loss.

C. Model Selection and Training

We did exploratory training using on several supervised machine learning models by using random grid search on a few of their parameters (found through trial and error) to identify the most effective for continued training:

- **Baseline Models**:
  - **Logistic Regression**: Evaluated as a benchmark due to its simplicity and interpretability.
  - **Support Vector Machines (SVM)**: Evaluated with linear and RBF kernels to capture potential non-linear relationships.
  - **Random Forests**: Evaluated for their ability to handle a large number of features and interactions.
  - **K-Nearest Neighbors (KNN)**: Evaluated for its simplicity, even though it has limitations in high-dimensional spaces.
  - **Gradient Boosting Machines (GBM)**: Evaluated for their robustness and ability to model complex patterns.
- **Advanced Models**:
  - **XGBoost**: Evaluated for its efficiency, scalability, and performance in handling tabular data with imbalanced classes.
  - **LightGBM**: Evaluated for its speed and capability to handle large datasets with lower memory usage.

After initial experimentation, XGBoost and LightGBM demonstrated superior and equal performance at a 0.8903 AUC-ROC score, which is higher than the other models' AUC-ROC scores. Consequently, we focused on these models for further optimization.

D. Hyperparameter Tuning with Optuna

To enhance the performance of XGBoost and LightGBM, we employed Optuna, an automatic hyperparameter optimization framework. We utilized Optuna for its efficient sampling algorithms (TPE sampler) and pruning strategies (Hyperband pruner), which accelerate the hyperparameter search process. We Defined extensive hyperparameter search spaces for both models, including parameters like learning rate, number of estimators, max depth, and regularization terms. We Created custom objective functions that included cross-validation within each trial to provide reliable performance estimates. The objective was to maximize the mean AUC-ROC across folds. One advantage of Optuna is that it could be configured to leverage GPU acceleration (using `tree_method='gpu_hist'` for XGBoost and `device='gpu'` for LightGBM) to speed up computations. The searching was conducted with an extensive hyperparameter searches of 200 trials for each model. Intermediate models were saved after each trial to prevent data loss and allow for resumption in case of interruptions. We then selected the best hyperparameters based on the highest mean AUC-ROC achieved during the trials. The models were retrained on the full training set using these optimal parameters. For XGBoost, the hyperparameters tuned included:

- `n_estimators`
- `learning_rate`
- `max_depth`
- `min_child_weight`
- `gamma`
- `subsample`
- `colsample_bytree`
- `reg_alpha`, `reg_lambda`
- `scale_pos_weight`
- `tree_method` set to `'gpu_hist'` for GPU acceleration

For **LightGBM**, the hyperparameters tuned included:

- `n_estimators`
- `learning_rate`
- `max_depth`
- `num_leaves`
- `min_child_weight`

- `subsample`
- `colsample_bytree`
- `reg_alpha`, `reg_lambda`
- `scale_pos_weight`
- `device` set to `'gpu'` for GPU acceleration

E. Model Evaluation

Evaluation of model performance was conducted systematically to ensure the selection of the most effective model. We implemented stratified 5-fold cross-validation within the Optuna objective functions to maintain class proportions in each fold and obtain reliable performance estimates. The primary performance metric was AUC-ROC due to its suitability for imbalanced datasets. Additionally, Precision-Recall curves were analyzed, and metrics such as Precision, Recall, and F1-Score for the minority class were reported. We also analyzed confusion matrices to understand the types of errors and to adjust classification thresholds appropriately. For the final models, feature importance was examined using built-in methods and SHAP (SHapley Additive exPlanations) values to interpret model predictions and ensure transparency.

F. Implementation and Deployment

Throughout the entire process, we've gradually integrated the model into a deployable application. We employed scikit-learn's `Pipeline` and `ColumnTransformer` to encapsulate preprocessing steps and the model, ensuring consistent application of transformations during training and inference. We serialized the trained models and preprocessing pipelines using `joblib` for efficient loading during deployment. We also developed a user-friendly interface using Flask, allowing users to input application data and receive real-time fraud risk assessments. Input validation and informative prompts were implemented to facilitate accurate data entry. The model was deployed using Gradio, an interactive platform that simplifies the sharing of machine learning models with end-users, enabling them to interact with the model through a web interface without the need for local installations.

Throughout the methodology, we adhered to best practices in machine learning, including rigorous testing and validation, careful handling of data imbalance, and attention to reproducibility and interpretability. Our approach ensures that the developed model is not only accurate but also reliable and applicable in real-world banking environments.

V. Experimental Results and Evaluation

A. Model Selection

In Bank Fraud Detection, a high rate of detecting fraudulent transactions and finding fraudulent bank accounts is obviously ideal but we need to make sure not to miss any potential fraudulent cases such as our model classifying a fraudulent case as legitimate. We will look into 4 models in depth which can be considered our best models: "LightBGM", "XGBoost", "Logistic Regression", and "Random Forest". These four models come with pros and cons which may be suitable for different situations that we will discuss in more depth.

B. Initial Analysis

After training our models we initially looked at LightBGM and XGBoost which yielded the best AUCROC scores of 0.8903 which indicates excellence in distinguishing between fraudulent and non-fraudulent transactions. The two other models we considered : "Logistic Regression", and "Random Forest" also have fairly high AUCROC scores of 0.8729 and 0.8691 respectively which is not a substantial amount lower than LightBGM and XGBoost. We visualize ROC curves for each model (Fig.7) comparing our key models and how they perform against each other.

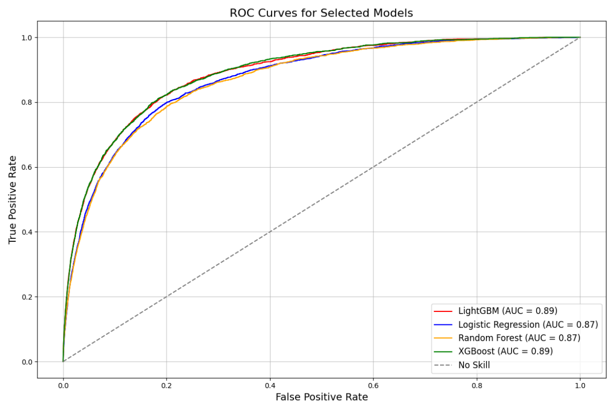


Fig. 7. ROC Curve

C. Into Classification Metrics

We may think that the models with higher scores would automatically be our winners given their high scores but we need to consider other metrics (Fig.9) such as Recall and False Positive Rates (FPR) because of how important it is to not miss any potential fraudulent cases. Looking at the classification metrics (Fig.8), we see XGBoost and LightGBM do very well in identifying legitimate cases that are actually legitimate but compared to Random Forest and Logistic Regression, it does not detect as many cases of fraud. With Random Forest and Logistic Regression the models will pick up on 74% and 80% of all fraud within the data set respectively which are really good, compared to about 1% for both XGBoost and LightGBM. Though the cases where a legitimate customer would be flagged as fraudulent is apparent given higher false positive rates in Logistic Regression and Random Forest, failing to detect potential cases of fraud could be extremely problematic given the sensitive nature of fraud detection. We will need to select the best model given the best balance of actual fraud cases, actual legitimate cases, missed fraud cases, and wrongfully flagged fraud cases.

D. Confusion Matrix Analysis

Looking at the confusion matrices can provide more insight into which model may be the most appropriate given specific situation (Fig.8, Fig.9). With a total of about 2200 fraudulent cases, XGBoost and LightBGM only correctly predict a mere 37 and 32 cases of fraud respectively which is an incredibly

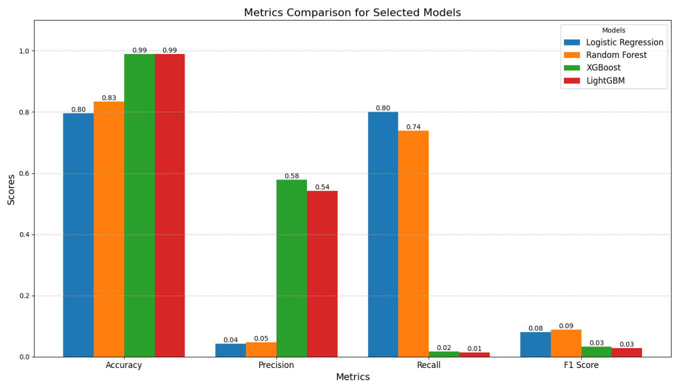
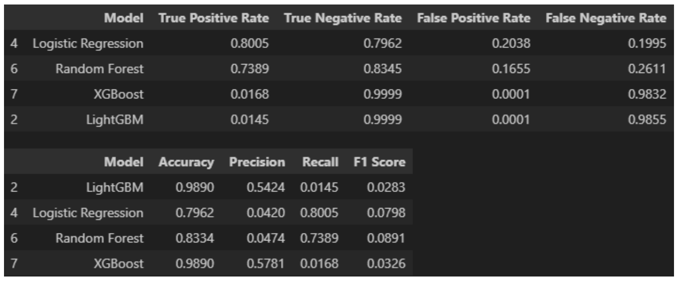


Fig. 8. Metric Comparison



	Model	True Positive Rate	True Negative Rate	False Positive Rate	False Negative Rate
4	Logistic Regression	0.8005	0.7962	0.2038	0.1995
6	Random Forest	0.7389	0.8345	0.1655	0.2611
7	XGBoost	0.0168	0.9999	0.0001	0.9832
2	LightGBM	0.0145	0.9999	0.0001	0.9855

	Model	Accuracy	Precision	Recall	F1 Score
2	LightGBM	0.9890	0.5424	0.0145	0.0283
4	Logistic Regression	0.7962	0.0420	0.8005	0.0798
6	Random Forest	0.8334	0.0474	0.7389	0.0891
7	XGBoost	0.9890	0.5781	0.0168	0.0326

Fig. 9. Metric Table

low number given the total number of fraudulent cases. As stated before, though about 20% of the total legitimate cases are wrongly classified to be fraudulent by Logistic Regression and Random Forest, while only 27 are wrongly classified by XGBoost and LightBGM, which correctly classifies the legitimate cases almost perfectly.

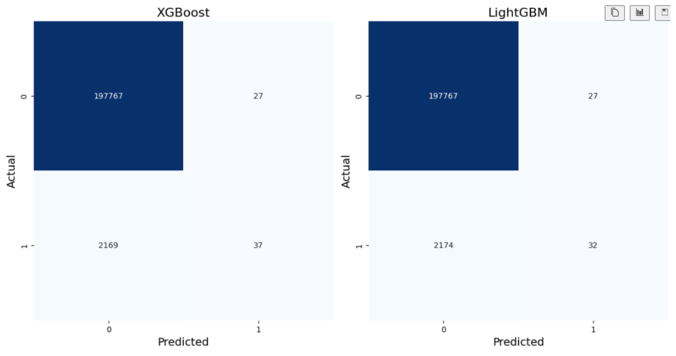


Fig. 10. XGBoost + LightGBM Matrices

E. Balancing Trade offs

Because of our imbalance dataset where the cases of fraud are about 1% of our entire data set, the numbers that clearly show an advantage in favor of Logistic Regression and Random Forest who identify a much greater proportion of fraud comparatively. The actual counts from analyzing the confusion matrices tell another story. From weighing the disadvantages and advantages of the Recall and Precision values we can decide which model could be superior. In a real world scenario given a similar distribution of data as our test data where the cases of fraud are so rare you could easily argue for the performance of XGBoost and LightBGM to be far superior. While not accurately predicting cases of actual fraud as fraud, it accurately predicts almost all actual legitimate cases as being legitimate, which just leaves about 1% of wrong predictions (2000 cases given our data) which could easily be manually checked. If we were to compare Logistic Regression and Random Forest which incorrectly predicts about 15%-20% (33000-40000 cases given our data) of the data, manually checking that many different data points could be quite challenging and inefficient.

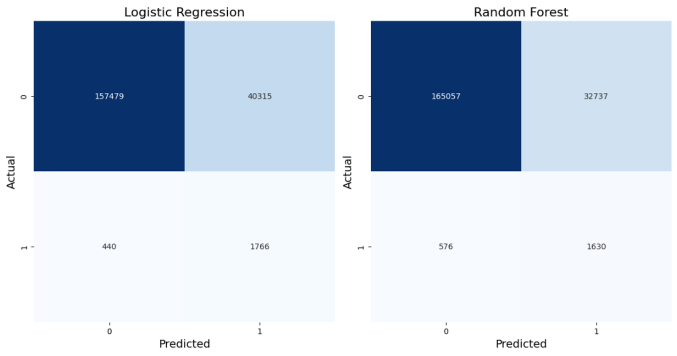


Fig. 11. Logistic Regression + Random Forest Matrices

VI. Conclusion and Discussion

In this study, we have demonstrated the effectiveness of supervised machine learning approaches to detect fraudulent bank account applications. Our XGBoost and LightGBM models achieved an AUCROC score of 0.8903, establishing a strong baseline for automated fraud detection in banking applications. A few key findings emerge from our research. By categorizing features into numerical, high-cardinality categorical, low-cardinality, and boolean types, and preprocessing each category, our exploratory data analysis and feature engineering approach positively contributed to our model's performance. The study successfully addressed the inherent class imbalance challenge (1.1The performance of XGBoost and LightGBM compared to traditional algorithms indicates that it is important to use advanced ensemble methods for complex fraud detection tasks such as these. The equivalent performance of both models indicates that both algorithms could be deployed in production environments. The use of Optuna for automated hyperparameter tuning was effective and enabled an efficient exploration of the parameter space, leading to optimized model configurations. Finally, the GPU acceleration reduced computational time, making the optimization process more practical.

A. Limitations and Future Work

While our model shows strong performance on the current dataset, fraud patterns evolve over time. Future work should focus on developing adaptive learning mechanisms to automatically update the model as new patterns emerge. Future studies could investigate additional features, particularly those related to digital footprints and behavioral patterns, which could further improve detection accuracy. While we focused on AUCROC as our primary metric, future work could incorporate actual costs associated with false positives and false negatives to optimize the model for business impact rather than purely statistical measures.

B. Practical Implications

Our testing suggests this system could be helpful for banking operations. We used common tools like Flask and Gradio to build a simple interface that bank staff could potentially use, while maintaining consistent data processing behind the scenes. The accuracy we achieved in our tests suggests this approach might help reduce manual review time while still identifying potential fraud. The methods we explored for detecting fraudulent bank accounts could potentially be useful for other types of financial fraud detection, such as credit card or loan applications. Our process for preparing and analyzing the data might provide helpful insights for similar problems in banking. In closing, we hope this project contributes something useful to the field of automated fraud detection in banking. Our experiments suggest that machine learning could help address this challenge, though more work would be needed for real-world implementation. While our results are promising in a test environment, we recognize that actual banking applications would require significant additional development and validation.

VII. Contributions

Albin Franzén: Wrote the initial EDA and started analysis of numerical features and high-cardinality categorical features. Wrote the code for the preprocessing and started model development for the standard sci-kit learn models. Wrote the introduction and EDA of the final report.

Isabel Shic: Focused on analyzing low-cardinality categorical features. Wrote code for low-cardinality pre-processing. Wrote conclusion for final report.

References

[1] "Financial fraud and credit risk: Illicit practices and their impact on banking stability," *Journal of Risk and Financial Management*, vol. 16, no. 9, p. 386, 2023.

[2] M. Mamonov, "Measuring fraud in banking and its impact on the economy: A quasi-natural experiment," *SSRN Electronic Journal*, 2022.

[3] M. Levi and M. Sherer, *The Evolution of Fraud: Case Studies and Best Practices*. Routledge, 1999.

[4] P. Sirawongphatsara, P. Pornpongtechavanich, P. Sriamorntrakul, and T. Daengsi, "Analyzing bank account information of nominees and scammers," *arXiv preprint arXiv:2308.01586*, 2023.

[5] A. Ali, S. Abd Razak, S. H. Othman, *et al.*, "Financial fraud detection based on machine learning: A systematic literature review," *Applied Sciences*, vol. 12, no. 19, 2022, ISSN: 2076-3417. [Online]. Available: https://www.mdpi.com/2076-3417/12/19/9637.

[6] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," *Expert Systems with Applications*, vol. 193, p. 116 429, 2022, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.116429. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421017164.

[7] A. Ahmadzadeh and R. A. Angryk, "Measuring class-imbalance sensitivity of deterministic performance evaluation metrics," *arXiv preprint arXiv:2206.09981*, 2022.

[8] S. Jesus, J. Pombal, D. Alves, *et al.*, "Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets for ML Evaluation," *Advances in Neural Information Processing Systems*, 2022.