Identifying Early Signs of Bank Account Fraud

ORIE 5741 - Cornell University (Spring 2024)

May 10, 2024

Anthony Coffin-Schmitt (awc93)

Jackie Lasseter (jl2638)

Elana Pocress (erp49)

1 Introduction

Fraudulent applications in the financial sector carry significant risk to business and the public. Identity theft, fraudulent credit and loan transactions, lost time and revenue are a few risks of bank fraud. As many financial institutions move to more capable online platforms, it is becoming easier for bad actors to submit fraudulent applications that can go undetected. The ease of submitting applications, combined with an increase in fluidity of bank customers between institutions, results in a higher number of applications to review and validate. It is not feasible or scalable to have manual review of submitted applications, nor does having a person review necessarily increase fraud detection rates. Even with such manual review, it is difficult to accurately determine what applications are fraudulent or safe due to the sheer amount of applications received and depth of these applications.

One solution is utilizing automated machine learning algorithms and techniques to identify possible fraudulent applications for further review or automatic rejection. With increasingly larger datasets of previous bank applications and additional external information about applicants, it is possible to develop fraud detection systems with high accuracy rates. However, developing highly accurate systems remains a challenge. Typically fraud datasets are highly imbalanced with a very small fraud class and most machine learning algorithms depend on balanced and independent data. Also, there is limited data, i.e. a single application, to make predictions from. Other fraud detection systems benefit from additional data instances, such as credit card purchase histories with several questionable transactions in a short time range.

In this project, we have several goals. First, we will utilize a large anonymized bank account application dataset, the Bank Account Fraud (BAF) suite [4], and explore the data features available. We aim to develop a machine learning model to detect fraudulent applications with relatively high confidence and analyze different meaningful metric results. Finally, we want to be able to offer financial institutions a range of prediction models that will adapt to their own priorities, e.g. rejecting the vast majority of fraudulent applications, or perhaps detecting a reasonable number of fraud applications while avoiding a high false detection rate.

2 Business Case

In 2023 alone, the Federal Trade Commission reported[3] 10 billion dollars lost from financial fraud . Fraud is prevalent in all industries, but specifically has a large impact on the financial sector. Ideally to minimize losses, the fraud should be detected as early as possible and at the source of a new account application. Waiting to catch fraudulent transactions as they occur is inefficient and carries higher risk. There is a large role that machine learning models can play in helping to identify these risks and eliminate potential fraudulent customers. Our project goals should address many of the ways that machine learning can help in the automation of fraud detection (and other prediction problems that extend beyond the use case for financial institutions). Because the issue of fraud is so

widespread and has impacts on industries even beyond banking, the output of our models are relevant and generalizable to other instances like manufacturing and health care.

3 Data Analysis

3.1 Dataset

Released in 2023 at NeurIPS Datasets and Benchmarks, BAF is a large-scale and realistic tabular dataset that maintains privacy. It contains a large sample of 2.5 million bank application instances generated from a Conditional Tabular Generative Adversarial Network (CTGAN) model trained on anonymized original banking data. This ensures that the dataset is privacy preserving in order to protect the identity of applicants in the past. Within the dataset suite, there are six instances, five of which represent a specific bias type and lastly a 'base' dataset. Each instance contains 1 million sub-samples from the large sample. We will be working with the 'base' dataset as it most closely resembles the true distribution of real life bank applications and most applicable to our project goals.

The labeled tabular datasets contain 31 features and temporal information. There is also information on the protected attributes of different groups, e.g. age, employment status, and income. The sample count includes 1 million instances, which is quite comprehensive data. The dataset is complete and does not have any missing values. Some key features include: `income`, `name_email_similarity`, `address`, `age`, `payment_type`, `zip`, `distinct_DOB`, `employment_status`, `credit_risk_score`, `email_is_free`, `housing_status`, and `previous_account`.

3.2 Exploratory Data Analysis

Out of the 31 different features, 14 features contained continuous data, 12 contained discrete data, 5 contained categorical data. Additionally, no duplicate examples were found. We looked at the distributions of the covariates in the dataset (Figure 1. Specifically, we wanted to compare these covariate distributions for the fraudulent and non-fraudulent data. The covariates that stood out were `name_email_similarity` and `current_address_months_count`. There was a more noticeable disparity between the non-fraud and fraud distributions. Additionally, the distributions were less smooth across all the non-fraud distributions. This is because there is less data in this class, which makes sense due to the law of large numbers. A summary of the results can be found in Appendix A.

The dataset was extremely imbalanced. 988,971 instances (98.9% of the dataset) were from the non-fraudulent class while 11,029 instances (1.1% of dataset) were from the fraudulent dataset. The dataset was complete, meaning there were no missing values or NaN find. Any item in the bank application form that was left empty was represented with a '-1'.

Furthermore, a correlation matrix was created to see if there were any other relationships between features (Figure 2). Unfortunately, the matrix showed very little correlation among features, requiring more extensive algorithms than a simple correlation to improve the model. A Principal Component Analysis (PCA) was performed on the dataset, but no obvious principal components were found that separated fraud from non-fraud instances (Figure 3).

3.3 Feature Engineering and Data Pre-processing

For feature engineering, in order to handle categorical variables, one-hot encoding was used to convert categorical variables into a binary format in order to provide more meaningful inputs into the machine learning models. The feature size went from 31 to 51 features, as 20 one-hot vectors were added. Also, during the data exploration, we found one feature with only one unique value, specifically '`device_fraud_count`'. As this feature made no meaningful contribution to the dataset it was dropped.

Before pre-processing, we applied a random 75% / 25% train-test split of the data, giving 750,000 instances for training and 250,000 instances for testing. The split was stratified, so there was an equal distribution of fraud and non-fraud instances in both the training and test sets. The test split is crucial to assess the model's abilities to generalize new, unseen data. Scaling and sampling techniques were not applied to the test set to avoid data leak and overly optimistic prediction confidence. Initially we performed Stratified K-Fold Cross Validation, trying 10-Fold and 5-Fold. However, computational

limitations hindered our attempts and we did not perceive a significant difference in the predictive power of cross validation compared to using the test-train split.

For data pre-processing, we knew that some models might benefit from scaling, such as Logistic Regression. So we initially explored three scaling techniques from the Scikit-learn package[7], specifically StandardScaler, MinMaxScaler, and RobustScaler. We also maintained a No Scaling set for comparison. The StandardScaler standardizes features by removing the mean and scaling to unit variance. MinMaxScaler scales the features to a given range, preserving the shape of the original distribution. Lastly, RobustScaler scales features based on statistics that are robust to outliers. Each scalar has a different effect on our machine learning algorithms, ranging from minimal to significant, so experimenting with multiple was necessary. RobustScaler and MinMaxScaler mostly did not make significant improvements in the final result. Therefore, only StandardScaler or no scaling were utilized. (See Figure).

As the dataset was highly imbalanced, some form of resampling to balance out the minority positive fraud class would likely be necessary. With the Imbalanced-learn package[6], we performed resampling to adjust the class distributions. A review of successful sampling techniques for highly imbalanced data led to the three techniques we tested. For over-sampling methods, we utilized Synthetic Minority Over-Sampling Technique (SMOTE) and increased the minority class (fraudulent data) to 10% of the total dataset from originally $\tilde{1}\%$. SMOTE works by generating new samples for the minority class by interpolating between existing minority class instances. For under-sampling we applied NearMiss which generated a 90% non-fraud to 10% fraud distribution. NearMiss selects a subset of majority class examples that are closest to minority class examples and removes those instances from the majority class. Lastly, with RandomUnderSampler, which randomly removes instances from the majority class until at the desired distribution, we generated a 50%-50% distribution. RandomUnderSampler was performed sequentially after first applying SMOTE. Similarly to the scaling techniques, we kept a 'No Resampling' data set for test comparisons.

4 Prediction Models

4.1 Model Selection

Since this project included utilizing a large set of imbalance data, we needed to try a variety of models in order to see which could handle this data best. We decided to prioritize both binary classification models that have a long proven track record, e.g. Logistic Regression, Decision Trees[8], and Random Forests[1], and models with more recent success, specifically Gradient Boosting Classifier, XGBoost (Extreme Gradient Boosting)[2], and LightGBM (Light Gradient Boosting Machine)[5]. We initially implemented kernel Support Vector Machines (SVM) as well but due to the larger number of data instances and high feature size, it did not scale well in regards to training times, we also found it was not generalizing well to the test dataset.

The diverse benefits from different models allowed comparison of different metrics and see which models excelled or faltered. Logistic Regression explored whether a generalized linear model would be sufficient, while Decision Trees offer intuitive decision-making. Random Forests may help mitigate overfitting through ensemble averaging. Lastly, Gradient Boosting Classifier, XGBoost, and LightGBM excel in capturing complex patterns and achieving superior predictive performance, with XGBoost and LightGBM known for scalability and efficiency. Choosing between these models allows for balancing interpretability, predictive power, and computational requirements in binary classification tasks.

4.2 Machine Learning Pipeline

To ensure uniformity, reproducibility, and fair comparisons, we used the same random seed throughout the model training and evaluation pipeline. We primarily worked in Python notebooks and implemented functions that iterated over each scaler, resampler, and model until convergence or until a high max iteration count. For each combination of the model and sampler, a pipeline was constructed, from either an Imbalanced-learning or Scikit-learn package. This allowed for a streamlined and automated machine learning workflow, while also substantially reducing the repetitive coding implementation.

If a sampler is provided, the pipeline includes an additional step to perform random under-sampling. The goal of under-sampling was to reduce bias in the dataset. The pipeline was then fitted on the

training data or the scaled training data. Finally, the pipeline makes predictions and probability estimates on the test data. In order to evaluate the model, the performance metrics compared the predictions to the true labels. A general overview of the pipeline can be seen in Figure 4

5 Metric Scoring

5.1 Metric Selection

The performance metrics of precision, recall, F1-score, receiver operating characteristic (ROC) curve and area under the ROC curve (ROC-AUC) were utilized to determine the efficacy of the model. Precision measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives and false positives). This highlights the accuracy of positive predictions made by the model. Precision is particularly important as a metric in this case because false positives are extremely undesirable. Incorrectly flagging legitimate applications as fraudulent can majorly inconvenience applicants and result in the business losing a new customer. However, precision does not fully tell the entirety of the best model to utilize.

Recall is the opposite of precision. It measures the proportion of correctly positive instances (true positives) out of all true positives and false negatives. Since in fraud detection failing to identify fraudulent activities can have significant financial implications, recall is a key metric in this case. For example, labeling a fraudulent account as not fraudulent has a more significant impact than vice versa. Therefore, this metric will be key in determining our classifiers performance.

ROC curve is a graphical representation of the true positive rate for different threshold values of a binary classification model. The closer the model is to the top left corner of the graph, the better the performance. This means there is a higher true positive rate and lower false positive rate. The metric ROC AUC gives a numerical assignment to an ROC curve by calculating the area under the curve (AUC). The higher the AUC value the better the model performs. However, ROC AUC by itself does not give insight into whether the model is performing specifically with precision and recall.

The confusion matrix is a table that visualizes the performance of a classification model by comparing predicted labels against actual labels. It helps give a quick overview of the model's accuracy, precision, recall, and F1 score by organizing predictions into four categories: true positives, true negatives, false positives, and false negatives.

For evaluation of the model's performance, we focused on precision, recall, ROC curve, ROC AUC, and the confusion matrix to assess. We largely ignored the overall accuracy metric due to it not being as informative in highly imbalanced data sets.

5.2 Model Performance Results

A comparison of precision, recall, F1-scores, and ROC AUC metrics are made between all models and the model's specific resampling method. There are 13 different base models, along with four resampling methods, giving 52 different metric results.

The precision metric (Figure 5) had a wide range in performance between models, with the best model being the Random Forest with No Resampling scoring over 70%. Most other models performed poorly. The precision figure also clearly shows that No Resampling outperformed other resampling techniques.

For recall (Figure 6), there is a better performance as compared to precision. Logistic Regression (scaled), Logistic Regression (scaled, balanced), LightGBM (scaled, balanced) all had very high recall scores close to 100%. Several models did not perform well, particularly the tree based models, XGBoost, and Logistic Regression with no pre-processing. It also appears that resampling with NearMiss or No Resampling results in better performance. When we compare the precision figure with the recall figure, it appears that the various models either excel in precision or recall, but very rarely both.

The F1-scores (Figure 7) illustrate how it is difficult to have equally high performance with precision and recall as all scores are below 30%, and many are below 5%. It does appear that the resampling method SMOTE with RandomUnderSampler outperforms other resampling methods in most of the models.

The metric given the most consideration in our final model selection is ROC AUC (Figure 8), as this metric balances the trade off between true positive rate and false positive rate. There were several models that performed well. We found that LightGBM (balanced) had the best performance. Typically, either No Resampling or SMOTE resampling had better ROC AUC scores compared to NearMiss. Focusing on the LightGBM (balanced) model, the ROC Curve (Figure 9). The No Resampling model score was 89.8%, SMOTE was 89.3%, and SMOTE RandomUnderSampler was also 89.3%. Both the probability threshold curve, in orange, and the curve based on prediction, in green, were included. The threshold in this case is used to classify observations into fraud or not fraud classes.

The confusion matrix for the LightGBM (balanced) - No Resampling versus LightGBM (balanced) - SMOTE resampling (Figure 10) gives clear example of the impact re-sampling can have on model. While the No Sampling model excelled at detecting fraud with 80.09% of fraud cases detected, it came at the expense of a high false positive rate of 16.19%. Whereas the SMOTE mode had a much lower fraud detection rate of 38.96% but a much more reasonable false positive detection rate of 2.36%. The confusion matrix can perhaps most clearly show a financial institution what to expect in detection rates for various models and can let them weigh the trade offs based on their needs.

6 Discussion

6.1 Weapons of Math Destruction

While predicting financial fraud has very important economic consequences, we do not envision the outcome of this model to become a weapon of math destruction. Our outcomes are clearly defined – either an application is fraudulent or not and there is little ambiguity because there are clearly defined regulations on what is and is not fraudulent. There is certainly a question regarding the ethical aspects of using certain predictors to determine whether something is fraudulent. This is particularly relevant when looking at the background of historical discrimination against certain minorities.

With this being said, it is important to not take any single model at face value and use one automated system to determine the fate of the applicants. For many financial institutions that welcome thousands of new applications every day there should be additional checks in place to ensure accuracy and fairness. One way this can be implemented is by outputting a probability score over a binary prediction. For example, the model can output an associate "likelihood" with every applicant id on the probability that such application will be fraudulent which can then be flagged for human review. This allows us to grant greater trust in our algorithm to shape the way an enterprise makes decisions because there is a quality assurance behind each decision.

6.2 Ethics and Equity

With any machine learning model, fairness considerations are needed. Algorithms can have big impacts on decision making. Since the models are trained on data available, minority groups for whom there is less data available may have less accuracy in their predictions. For example, one of the instances used is customer age. Customers that are young could be discriminated against in comparison to their older counterparts. Additionally, another feature is name and email similarity, if customers do not have an email address that resembles their name, the customer is also more likely to be discriminated against based on our model. Oftentimes, predictive algorithms can discriminate against groups that have historically been discriminated against due to the nature of the dataset, so it is important to take these ethics concerns into consideration when devising algorithms.

6.3 Future Improvements

While our final model recommendation of LightGBM (balanced) with No Resampling does detect the majority of frauds, there is still room for improvements. Performing RandomizedSearchCV or GridSearchCV from the Scikit-learn package would allow for hyperparamter fine-tuning and better overall performance. Cross validation can be computationally expensive but it does allow for iterative improvements in the model's performance.

Another possible improvement is the inclusion of the F-beta score, which is similar to F1-score but the mean is weighted. This would allow a financial institution to assess how models would perform based

on the weight assigned to high fraud detection rates versus the weight assigned to false positives. As each financial institution has vast risk management resources and are already privy to the different costs associated with those predictions, the F-beta score would give further insight into the best model. That score was not included in our metric evaluation due to being uncertain of the appropriate weight to apply.

7 Contributions

All group members were involved in analyzing and cleaning the dataset, as well as creating the presentation and report. All members also implemented different classification models of interest to gain a general idea of performance. Jackie Lasseter worked on data exploration. Elana Pocress worked on the business need and video script. Anthony Coffin-Schmitt worked on the model pipeline and metric results. All members contributed to the presentation and report.

References

[1] Leo Breiman. "Random Forests". English. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A%3A1010933404324.

[2] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[3] Federal Trade Commission. *As Nationwide Fraud Losses Top $10 Billion in 2023, FTC Steps Up Efforts to Protect the Public*. Feb. 2024. URL: https://www.ftc.gov/news-events/news/press-releases/2024/02/nationwide-fraud-losses-top-10-billion-2023-ftc-steps-efforts-protect-public.

[4] Sérgio Jesus et al. "Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets for ML Evaluation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 33563–33575. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/d9696563856bd350e4e7ac5e5812f23c-Paper-Datasets_and_Benchmarks.pdf.

[5] Guolin Ke et al. "LightGBM: a highly efficient gradient boosting decision tree". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. ISBN: 9781510860964.

[6] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-365.html.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] J. R. Quinlan. "Induction of Decision Trees". In: *Mach. Learn.* 1.1 (Mar. 1986), pp. 81–106. ISSN: 0885-6125. DOI: 10.1023/A:1022643204877. URL: https://doi.org/10.1023/A:1022643204877.

Links

ORIE 5741 Project Github	github.com/AnthonyC-S/orie_5741_project
Bank Account Fraud Kaggle Dataset	kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022
Bank Account Fraud Github	github.com/feedzai/bank-account-fraud
Bank Account Fraud Datasheet	github.com/feedzai/bank-account-fraud/blob/main/documents/datasheet.pdf

Appendix

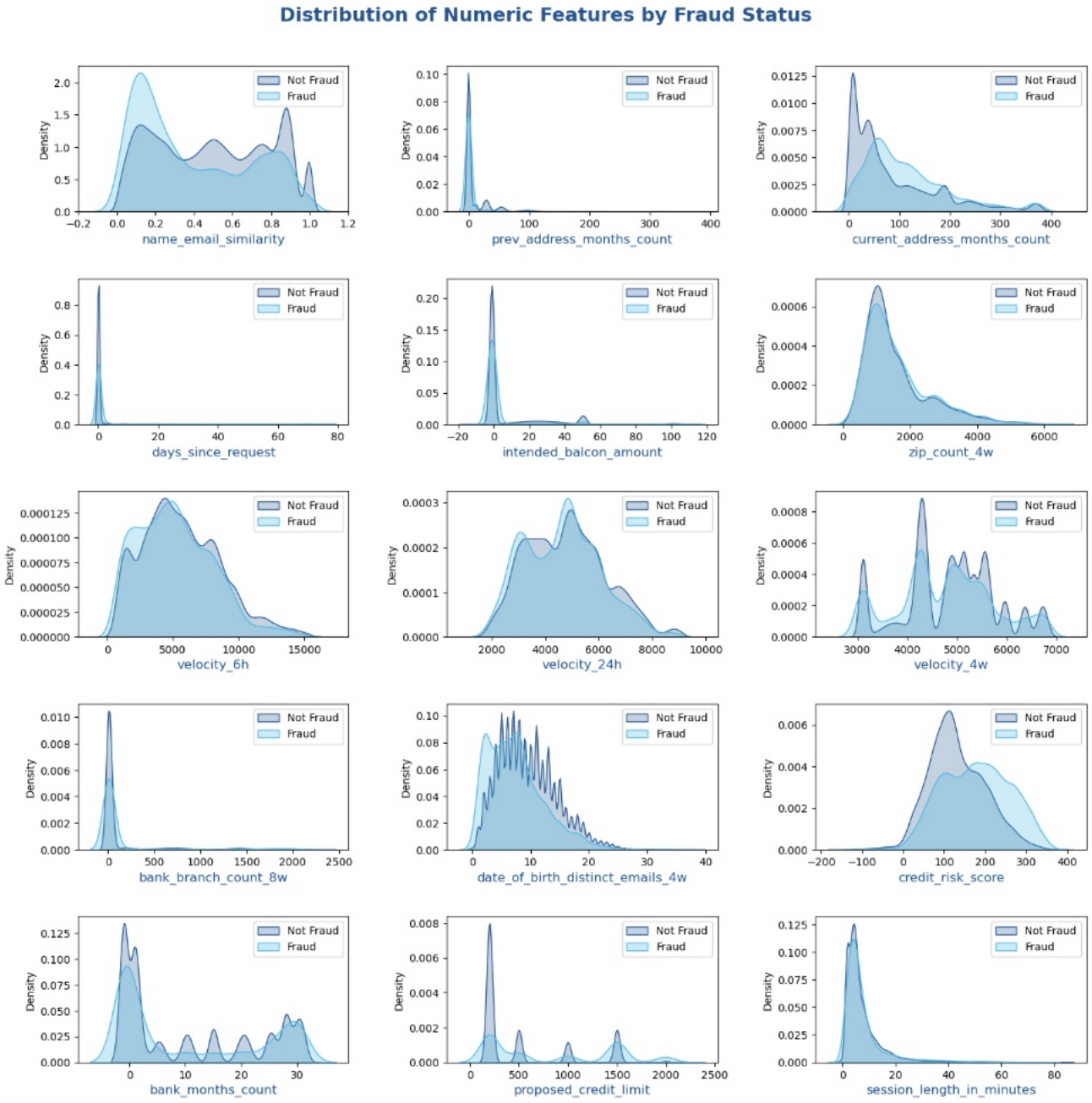

Figure 1: Dataset feature distributions, comparing the non-fraud covariates to the fraud covariates.

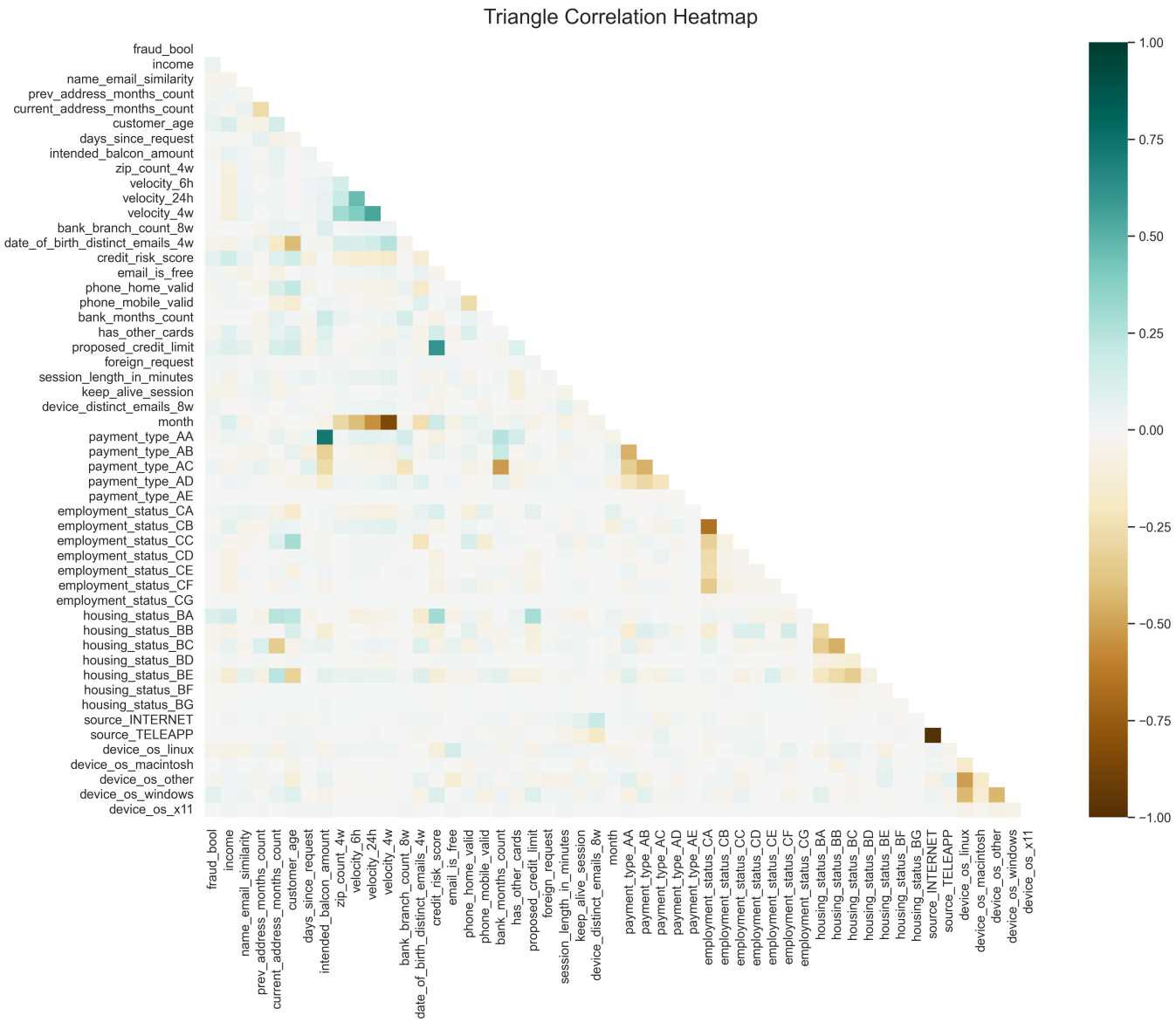

Figure 2: Triangle correlation heatmap of all features after one-hot vector encoding. Limited correlation seen with the classification class, `fraud_bool`.

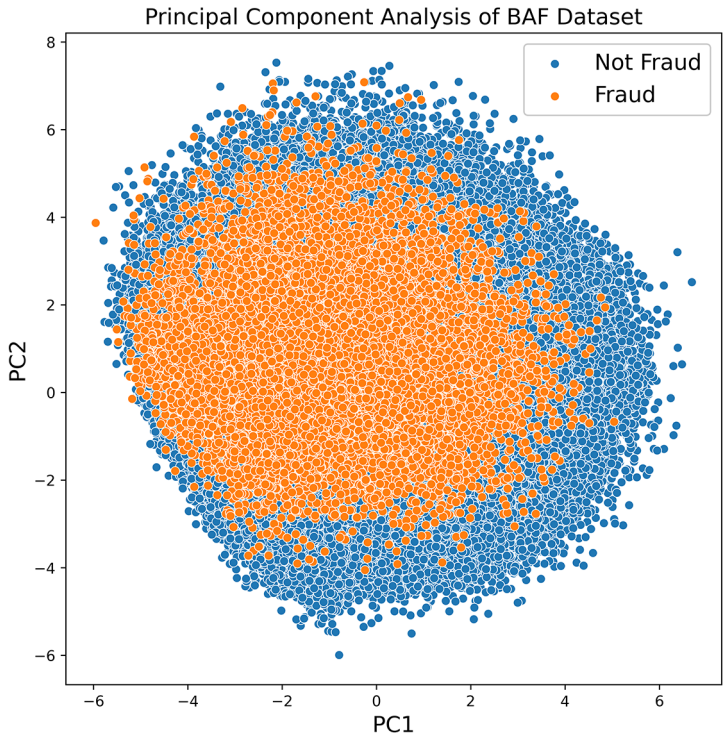

Figure 3: Principal component analysis did not show any clear separation between the classification class, `fraud_bool`.

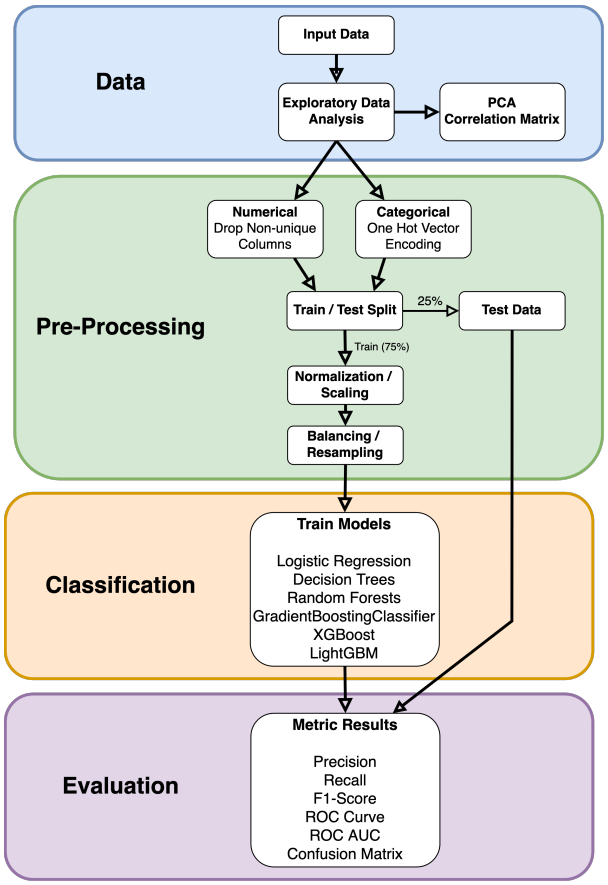

Figure 4: Model pipeline flowchart.

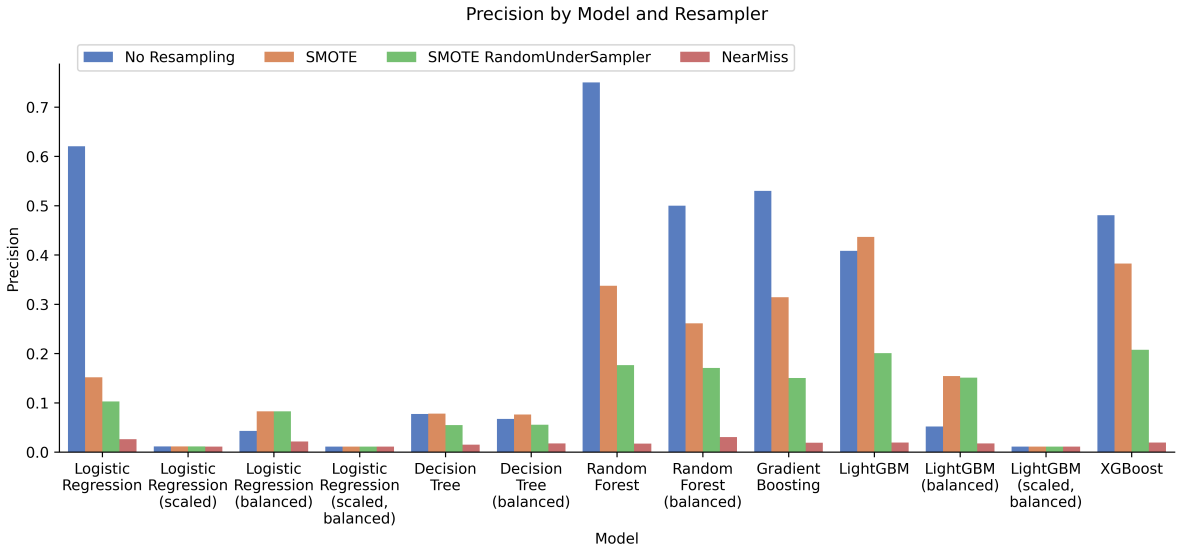

Figure 5: Model metric scores for precision.

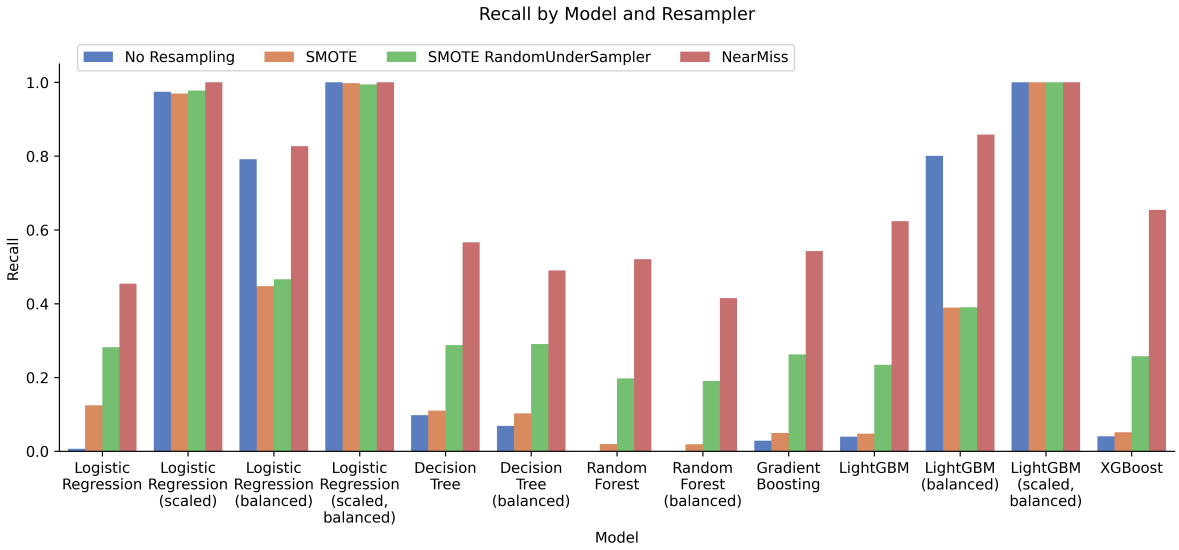

Figure 6: Model metric scores for recall.

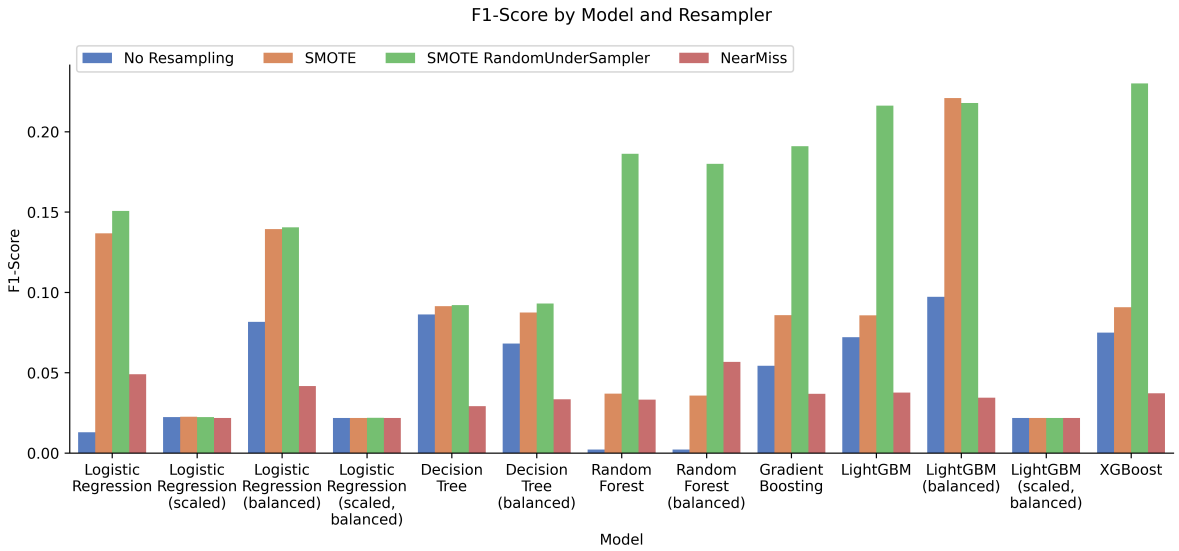

Figure 7: Model metric scores for F1-score.

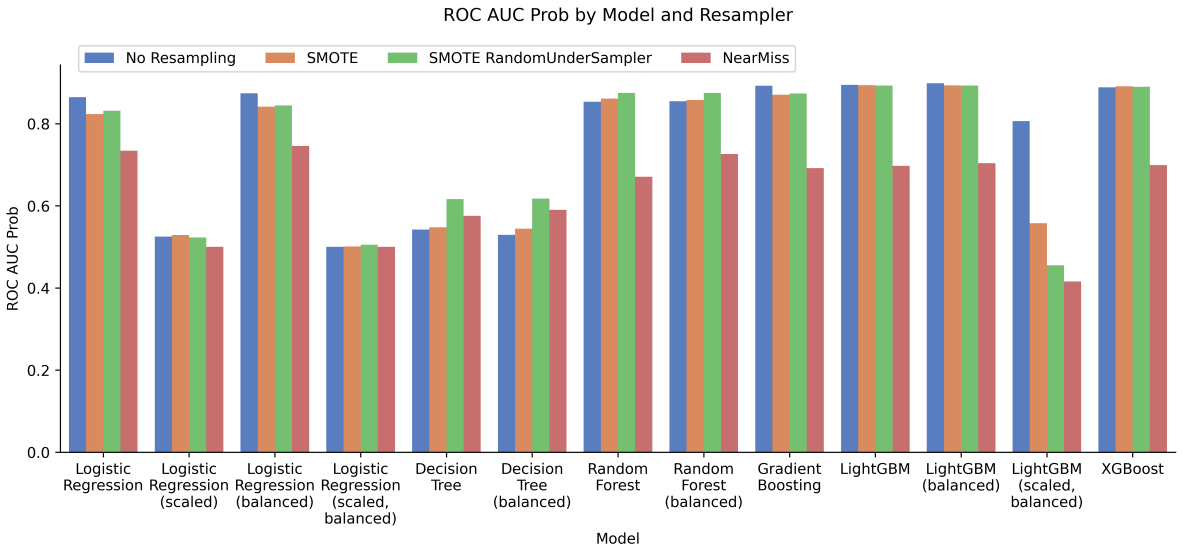

Figure 8: Model metric scores for ROC AUC based on probability thresholds.

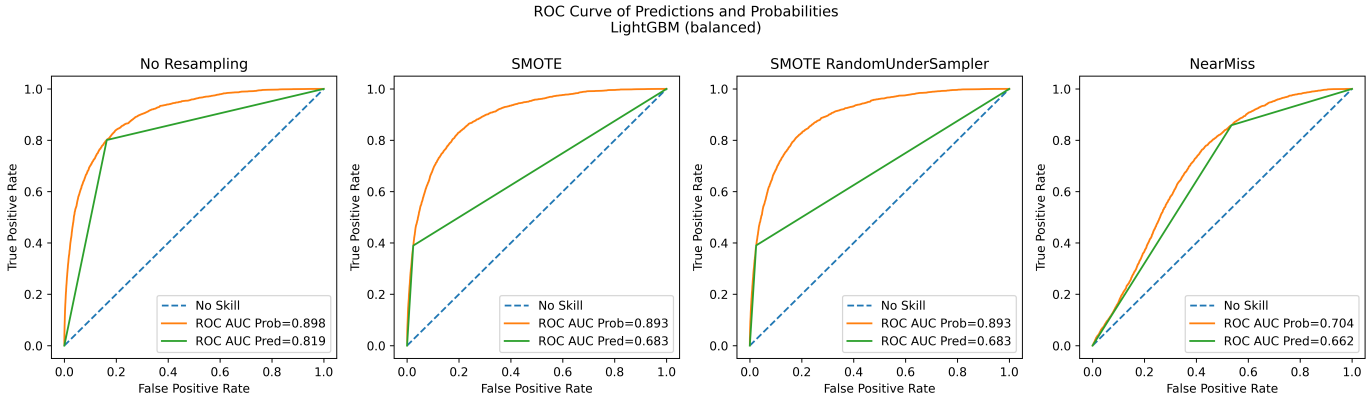

Figure 9: ROC Curves for LightGBM models. The orange line represents the probability thresholds while the green line is based on the binary classification prediction.

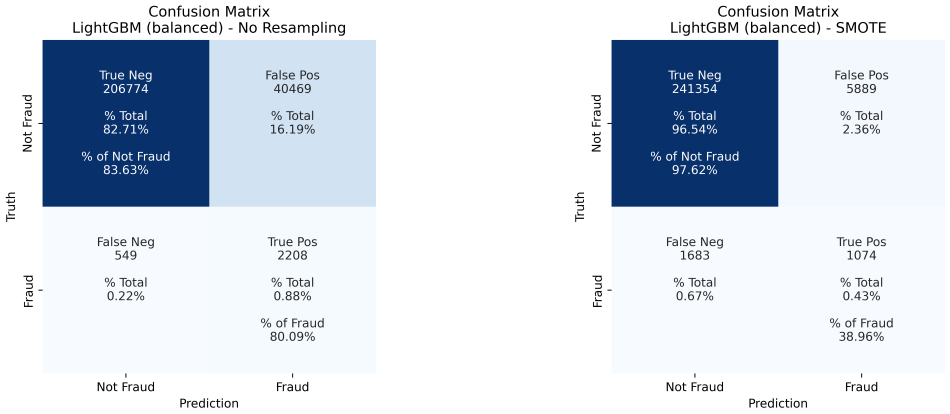

Figure 10: Confusion matrices for LightGBM model with no resampling and with SMOTE resampling. Illustrates how a model can prioritize a high fraud detection rate (80.09%) at the cost of false positives (16.19%) as seen on left, compared to a lower fraud detection rate (38.96%) with fewer false positives (2.36%) as seen on right.