University of Central Florida

Department of Electrical & Computer Engineering

EEL 5825
Machine Learning and Pattern Recognition

Final Project

Bank Fraud Detection Models Analysis

Author:
Austin Berg

Instructor:
Dr. Truong Nghiem

December 4, 2024

Austin Berg
Dr. Truong Nghiem
EEL5825

Table of Contents:

Austin Berg
Dr. Truong Nghiem
EEL5825

Introduction:

Project Scope

Study and compare various classification models for the purpose of detecting bank fraud. The models considered are: 1) a linear model covered in class (for example the logistic regression model with/without nonlinear transformation); 2) the classification tree XGboost model; 3) the Naive Bayes model (with/without nonlinear transformation); Given a set of training data, these models will be tasked with learning a pattern to detect bank fraud on accounts from transaction data. Transaction data may include date and time, amount withdrawn or deposited, type of transaction (cash, check or online transfer), and possibly others. The models will be coded in Python using the SciKit-Learn library. Then give the models test data with known outcomes to check for accuracy.

Dataset

The reference dataset used is from Kaggle, a website hosting machine learning datasets, models and code. The exact dataset is "Bank Account Fraud Dataset Suite (NeurIPS 2022)".

This dataset was chosen for a multitude of reasons, including but not limited to:

- Realism: Dataset was generated from a real fraud detection dataset.
- Bais: There is a distinct bias between fraud and non-fraud accounts.
- Large Size: 1 million total accounts, to allow for accurate training and to alleviate overfitting.

Approach

To solve a bank fraud detection problem, the process typically begins with acquiring and preparing the data, ensuring it is structured for analysis. The next step involves selecting relevant features that may help distinguish fraudulent transactions from legitimate ones. The data is then split into different subsets for training and testing. A model is trained on the training data to learn patterns and relationships within the features that correlate with fraud. The model is tested on unseen data to assess its generalization and real-world performance. The model's performance is measured using appropriate metrics, ensuring it can reliably detect fraudulent transactions. This approach, adaptable to various models, focuses on building, validating, and evaluating a predictive system to identify fraud.

Background:

Smote

Because this dataset has a large imbalance between fraudulent and non-fraudulent transactions, a technique called Synthetic Minority Over-sampling Technique (SMOTE) can be used to address this issue. SMOTE works by creating synthetic examples for the minority class (fraudulent transactions) instead of simply duplicating existing examples. It selects a data point from the minority class and identifies its k-nearest neighbors using the k-nearest neighbors (KNN) algorithm. Then, new instances are generated by interpolating between the selected point and its neighbors, creating synthetic samples that are similar but not identical to the originals. This helps balance the class distribution, allowing the model to better learn from the minority class without overfitting, leading to more robust and accurate predictions.

GridSearchCV

GridSearchCV is a technique used to optimize the hyperparameters of machine learning models by systematically searching through a predefined set of parameters to find the best combination. It performs an exhaustive search over a specified parameter grid, testing all possible combinations of parameters to identify the one that results in the best model performance, usually based on a specified evaluation metric. GridSearchCV typically uses cross-validation to assess the model's performance for each combination of hyperparameters, ensuring that the selected model generalizes well to unseen data. This technique can significantly improve model performance by fine-tuning hyperparameters such as learning rate, regularization strength, and tree depth, which are crucial for achieving optimal results. By automating the search for the best hyperparameters, GridSearchCV saves time and helps avoid manual tuning, leading to more accurate and efficient models.

Threshold Tuning

Threshold tuning is a technique used to adjust the decision threshold for classifying predictions, particularly in imbalanced datasets or when optimizing for specific performance metrics like precision and recall. In binary classification, models typically output probabilities for each class, and a threshold is applied to these probabilities to make a final classification decision. By default, a threshold of 0.5 is used, meaning that if the probability of the positive class (e.g., fraud) is greater than or equal to 0.5, the model classifies the instance as positive, otherwise as negative. However, this threshold can be adjusted to optimize the model's performance for specific goals. For instance, in fraud detection, it might be more important to capture as many fraudulent transactions as possible, even at the cost of increasing false positives. By experimenting with different thresholds, one can maximize metrics like the F1-score, which balances precision and recall, or plot a Precision-Recall curve to visualize the trade-offs between the two. Threshold tuning allows the model to be more fine-tuned to the problem's specific needs, improving overall prediction accuracy and effectiveness.

Austin Berg
Dr. Truong Nghiem
EEL5825

Logistic regression:

Logistic regression is a statistical model for binary classification, predicting one of two outcomes (e.g., fraud or no fraud). It models the relationship between features and the probability of a specific class using a sigmoid function, which outputs values between 0 and 1. The model learns feature weights by minimizing log loss, a function that penalizes incorrect predictions based on their deviation from true probabilities. These weights show each feature's contribution to the prediction, enabling decisions through a linear combination of inputs. Logistic regression excels with linearly separable data and provides interpretable, probabilistic predictions.

Here is a description of the steps taken for the functional logistic regression model:
(All words in ***bolded italics*** are function libraries)

First, the code loads the data using ***pandas***, a tool that organizes data into a structured format, making it easier to manipulate (in this case, it imports the .csv file and its data). It separates the target column, fraud_bool, which indicates whether a transaction is fraudulent, so the model knows what to predict. For preprocessing, it scales numerical features (like transaction amounts) with ***MinMaxScaler*** to ensure all values are on a similar range (0 to 1), and it uses ***OneHotEncoder*** to convert text-based categories (like city or transaction type) into numerical formats that the model can interpret.

To improve model performance, it generates new features with ***PolynomialFeatures***, which capture interactions between the original features, helping the model find more intricate patterns. Since fraudulent transactions are rare (the data is "imbalanced"), it uses ***SMOTE*** (Synthetic Minority Over-sampling Technique) to create synthetic examples of fraud, balancing the dataset and giving the model a fair chance to learn both fraud and non-fraud cases effectively.

After preprocessing, it splits the data into three parts using ***train_test_split***: training, validation, and test sets. The training set is used to teach the model, while the validation set is carefully balanced (with an equal number of fraud and non-fraud cases) to fine-tune decision thresholds later. The test set, kept separate, is used to evaluate final model performance.

The ***SciKit-Learn*** logistic regression model (***LogisticRegression***) is set up to handle imbalanced data, with specific parameters like class weights to adjust its sensitivity, a regularization value C=1 (found with ***GridSearchCV***) to prevent overfitting, and a maximum number of iterations to ensure convergence. It learns patterns in the training data to distinguish fraud from legitimate transactions.

Next, the model's threshold for predicting fraud is fine-tuned. Instead of the default threshold of 0.5, it tests a range of values to find the one that maximizes the F1-score; a metric that balances precision (avoiding false alarms) and recall (catching all fraud cases). A ***Precision-Recall curve*** is plotted to visualize how these metrics change with the threshold, marking the best one.

Finally, the model is evaluated on the test set using the selected threshold. ***Accuracy***, a ***confusion matrix*** (showing the breakdown of predictions), and a ***classification report*** (precision, recall, F1-score for each class), are printed to assess effectiveness.

Austin Berg
Dr. Truong Nghiem
EEL5825

Naïve-Bayes:

Naïve Bayes is a probabilistic classification model based on Bayes' Theorem, assuming strong independence among features. It predicts the likelihood of a class given input features by combining prior probabilities of classes with the likelihood of the observed data under each class. Despite the simplifying assumption of feature independence, which rarely holds in real-world data, Naïve Bayes often performs well in practice, particularly for text classification and spam detection tasks. The model is computationally efficient, as it calculates probabilities for each feature independently, making it scalable to large datasets. Its simplicity, robustness, and ability to handle categorical data make Naïve Bayes a versatile tool for various classification problems.

Here is a description of the steps taken for the functional Naïve-Bayes model:

(All words in ***bolded italics*** are function libraries)

First, the code loads the data using ***pandas***, a tool that organizes data into a structured format, making it easier to manipulate (in this case, it imports the .csv file and its data). It separates the target column, fraud_bool, which indicates whether a transaction is fraudulent, so the model knows what to predict. For preprocessing, it scales numerical features (like transaction amounts) with ***MinMaxScaler*** to ensure all values are on a similar range (0 to 1), and it uses ***OneHotEncoder*** to convert text-based categories (like city or transaction type) into numerical formats that the model can interpret.

To improve model performance, it generates new features with ***PolynomialFeatures***, which capture interactions between the original features, helping the model find more intricate patterns. Since fraudulent transactions are rare (the data is "imbalanced"), it uses ***SMOTE*** (Synthetic Minority Over-sampling Technique) to create synthetic examples of fraud, balancing the dataset and giving the model a fair chance to learn both fraud and non-fraud cases effectively.

After preprocessing, it splits the data into three parts using ***train_test_split***: training, validation, and test sets. The training set is used to teach the model, while the validation set is carefully balanced (with an equal number of fraud and non-fraud cases) to fine-tune decision thresholds later. The test set, kept separate, is used to evaluate final model performance.

The ***SciKit-Learn*** Complement Naive Bayes model (***ComplementNB***) is configured with an alpha parameter of 1 (Found with ***GridSearchCV***) to smooth probabilities and prevent overfitting on sparse data. The model is trained using the fit method, learning patterns in the training data to differentiate between fraud and legitimate transactions effectively.

Next, the model's threshold for predicting fraud is fine-tuned. Instead of the default threshold of 0.5, it tests a range of values to find the one that maximizes the F1-score—a metric that balances precision (avoiding false alarms) and recall (catching all fraud cases). A ***Precision-Recall curve*** is plotted to visualize how these metrics change with the threshold, marking the best one.

Finally, the model is evaluated on the test set using the selected threshold. ***Accuracy***, a ***confusion matrix*** (showing the breakdown of predictions), and a ***classification report*** (precision, recall, F1-score for each class), are printed to assess effectiveness.

Austin Berg
Dr. Truong Nghiem
EEL5825

XGBoost:

XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable machine learning algorithm for both regression and classification tasks. It is built on decision trees and uses gradient boosting to minimize errors iteratively. Gradient boosting combines weak learners (simple decision trees) into a single, strong learner. Each subsequent tree corrects the errors of its predecessors by focusing more on poorly predicted instances. XGBoost introduces innovations like regularization (to prevent overfitting), parallel computation, and robust handling of missing data, making it fast and highly performant.

Here is a description of the steps taken for the functional XGBoost model:
(All words in ***bolded italics*** are function libraries)

First, the code loads the data using ***pandas***, a tool that organizes data into a structured format, making it easier to manipulate (in this case, it imports the .csv file and its data). It separates the target column, fraud_bool, which indicates whether a transaction is fraudulent, so the model knows what to predict. For preprocessing, it scales numerical features (like transaction amounts) with ***MinMaxScaler*** to ensure all values are on a similar range (0 to 1), and it uses ***OneHotEncoder*** to convert text-based categories (like city or transaction type) into numerical formats that the model can interpret.

Since fraudulent transactions are rare and lead to class imbalance, the ***SMOTE*** (Synthetic Minority Over-sampling Technique) algorithm is applied to create synthetic examples of fraud, balancing the dataset and enabling the model to learn patterns for both fraud and non-fraud cases effectively. After preprocessing, it splits the data into three parts using ***train_test_split***: training, validation, and test sets. The training set is used to teach the model, while the validation set is carefully balanced (with an equal number of fraud and non-fraud cases) to fine-tune decision thresholds later. The test set, kept separate, is used to evaluate final model performance.

The ***XGBoostClassifier*** is then configured with important parameters such as scale_pos_weight to adjust for class imbalance, early_stopping_rounds to prevent overfitting by halting training if no improvement is observed, and other regularization parameters like tree depth and learning rate (found with ***GridSearchCV***) to optimize the model's performance. The model is trained on the SMOTE-processed training set and validated using the balanced validation set.

Next, the model's threshold for predicting fraud is fine-tuned. Instead of the default threshold of 0.5, it tests a range of values to find the one that maximizes the F1-score—a metric that balances precision (avoiding false alarms) and recall (catching all fraud cases). A ***Precision-Recall curve*** is plotted to visualize how these metrics change with the threshold, marking the best one.

Finally, the model is evaluated on the test set using the selected threshold. ***Accuracy***, a ***confusion matrix*** (showing the breakdown of predictions), and a ***classification report*** (precision, recall, F1-score for each class), are printed to assess effectiveness.

Austin Berg
Dr. Truong Nghiem
EEL5825

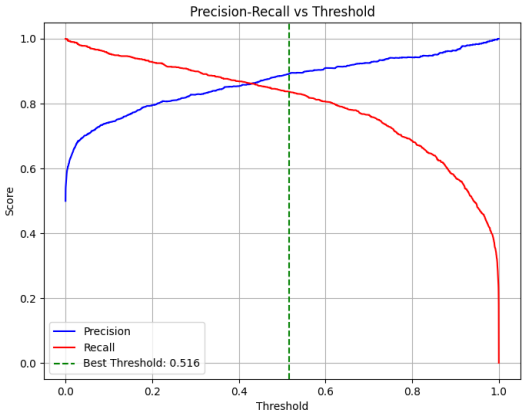
Results:

Scoring Metrics

- Recall: The proportion of actual positive cases (e.g., fraud) correctly identified by the model, focusing on minimizing false negatives.
- Accuracy: The proportion of total predictions (both positive and negative) that the model classified correctly.
- F1-Score: The harmonic means of precision and recall, providing a balanced measure of a model's performance, especially when handling imbalanced datasets.

Models' Results

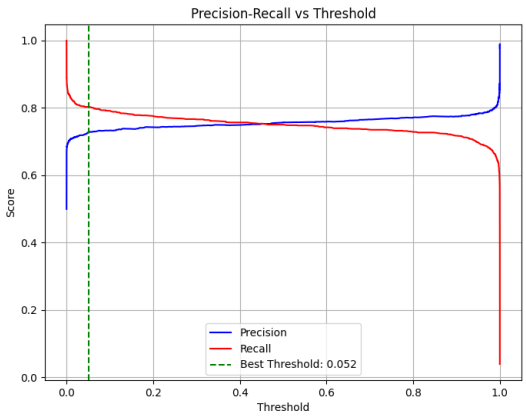
Logistic Regression:



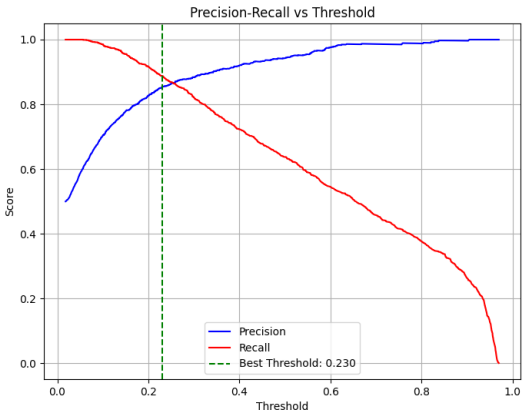
With an optimal threshold of 0.516, the Logistic Regression model achieved an F1-score of 0.86 and the highest accuracy among the models at 90.03%. It delivered strong recall for fraud cases, successfully identifying 83% of fraudulent transactions. Its high accuracy suggests that it performed well in balancing the detection of fraud while maintaining correct predictions for legitimate transactions.

Naïve Bayes:

With an optimal threshold of 0.052, Naïve Bayes achieved an F1-score of 0.76 and an accuracy of 69.92%, the lowest among the models. While it achieved a recall of 82% for fraud cases, it struggled with the highly imbalanced dataset, leading to reduced overall performance. Its inability to account for the correlations between features likely contributed to its weaker results.'



XGBoost:



With an optimal threshold of 0.230, the XGBoost model achieved an F1-score of 0.87 and an accuracy of 84.09%. It demonstrated excellent recall for fraud cases (class 1), identifying 89% of them correctly. While its performance at capturing fraudulent activity was strong, its overall accuracy reflected challenges in handling the distribution of legitimate and fraudulent transactions in the dataset.

Model Comparison

Best Performance

Logistic Regression achieved the best accuracy (90.03%) and an excellent trade-off between precision and recall for fraud cases (class 1). Its slightly more balanced metrics make it more reliable in practice. It reduced false positives compared to XGBoost while maintaining strong recall. Its robust performance highlights its suitability for handling the challenges of class imbalance in the dataset.

Intermediate Performance

XGBoost performed nearly as well as Logistic Regression in recall for class 1 (0.89), making it slightly better at identifying fraud. However, its excellent recall was at the cost of a slightly lower overall accuracy than Logistic Regression. This indicates that while XGBoost is excellent at capturing fraud, it may face challenges in maintaining performance for legitimate transactions. Despite this, its high recall makes it a strong contender for scenarios where maximizing fraud detection is a priority.

Worst Performance

Naïve Bayes performed the worst, with the lowest accuracy (69.92%) and F1-score among the models. While it achieved a respectable recall of 82% for fraud cases, its overall performance was hindered by its simplistic assumptions and inability to handle the dataset's class imbalance effectively. This model struggled to account for feature dependencies, which are often critical in fraud detection. As a result, Naïve Bayes was unable to provide reliable predictions, making it the least suitable model for this task.

Conclusion

In conclusion, Logistic Regression emerged as the best-performing model for this fraud detection task, achieving the highest accuracy (90.03%) and demonstrating strong recall for fraudulent cases. It was able to strike a balance between correctly identifying fraudulent transactions and maintaining accurate predictions for legitimate ones. This balance is essential in fraud detection, where both catching fraud and avoiding unnecessary disruptions to legitimate users are critical. Logistic Regression's ability to calibrate thresholds effectively and adapt to the class imbalance contributed significantly to its robust performance.

XGBoost also delivered commendable results, achieving an impressive recall of 89% for fraud cases and an overall accuracy of 84.09%. Its strength lies in its advanced algorithm, which uses decision tree ensembles to capture complex patterns in the data. This makes it particularly adept at identifying rare events, such as fraud. However, its slightly lower overall accuracy compared to Logistic Regression suggests it may struggle with the broader classification task, especially when dealing with the large class imbalance inherent in the dataset. Despite this limitation, XGBoost's high recall makes it an excellent option when maximizing fraud detection is a priority.

On the other hand, Naïve Bayes struggled significantly, achieving the lowest accuracy (69.92%) and F1-score among the models. While it achieved a reasonable recall of 82% for fraud cases, its overall performance was hindered by its reliance on the assumption of feature independence. This assumption does not hold in fraud detection, where features often have complex relationships. Additionally, Naïve Bayes was unable to effectively handle the extreme class imbalance in the dataset, leading to inaccurate probability estimates for the majority class (legitimate transactions). This weakness likely caused the model to misclassify a large number of transactions, reducing its overall effectiveness.

The strong performance of Logistic Regression and XGBoost highlights their adaptability and robustness in handling imbalanced datasets. Logistic Regression's simplicity and ability to model linear relationships effectively make it a solid choice for fraud detection tasks. XGBoost, with its ability to capture non-linear relationships and optimize performance through boosting, also proved highly capable. Naïve Bayes, in contrast, struggled due to its overly simplistic assumptions and inability to capture the nuanced relationships between features. These results stress the importance of selecting models that align with the complexity and specific challenges of the data.

References, Software and Libraries:

Dataset Used:

Bank Account Fraud Dataset Suite (NeurIPS 2022)
https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022

Software Used:

Python: Python Software Foundation. Python Language Reference, version 3.12.

https://www.python.org

Libraries Used:

pandas:

https://pandas.pydata.org

numpy:

https://numpy.org

scikit-learn:

https://scikit-learn.org

imblearn:

https://imbalanced-learn.org

matplotlib:

https://matplotlib.org

xgboost:

https://xgboost.readthedocs.io