Fraud Detection Final Report

Introduction and Motivation:
The dataset I am using is a financial fraud detection dataset. I chose this dataset because I am interested in utilizing machine learning in a cybersecurity setting to identify bad actors. I believe the intersection of these two specialties has potential in the future, as companies are already starting to employ machine learning models to assist their cybersecurity teams. Additionally, I am interested in exploring this subject more in graduate school.

Dataset Overview:
This dataset was obtained form kaggle.com specifically here:
https://www.kaggle.com/datasets/sriharshaeedala/financial-fraud-detection-dataset

This dataset contains 6,362,620 rows and 11 columns, meaning that there are a total of 6,362,620 training examples and 11 features. The target variable of this dataset is (isFraud). The number of training examples that I used to train my models was 5,090,096, and I had 1,272,524 testing examples. Meaning that I split the data set into 80% training and 20% testing examples.
The dataset has both categorical and numerical features as described below:

Attribute	Description (inferred)
step	A unit of time (hours 1-744) since the start of the simulation (Numerical)
type	The type of transaction. (Categorical)
amount	The amount of money involved in the transaction. (Numerical)
nameOrig	The customer is initiating the transaction. (Their ID number)
oldbalanceOrg	Balance of the origin account before the transaction. (Numerical)
newbalanceOrig	Balance of the origin account after the transaction. (Numerical)
nameDest	Destination account ID
oldbalanceDest	Initial recipient's balance before the transaction. (Numerical)
newbalanceDest	New recipient's balance after the transaction. (Numerical)
isFraud	Indicates whether a transaction was actually fraudulent. (Categorical)
isFlaggedFraud	Flags large-scale, unauthorized transfers between accounts, with any single transaction exceeding 200,000 being (Categorical)
amount_oldbalanceOrg_ratio	equation: amount / (oldbalanceOrg + 1) added 1 to avoid dividing by zero. If the balance was

(Feature Engineered)	empty. tells you what percentage of the sender's money was moved. (Numerical)
isCompleteTransfer (Feature Engineered)	If the newbalance = 0 and the old balance is > 0 then flag as fraud, since usually fraudulent transactions may zero out the account. (Categorical)
errorBalance or avg_error (Feature Engineered) equation: Oldbalance - amount transacted - abs(new balance)	Measures balance discrepancies; values close to 0 suggest normal transactions, while higher values indicate a higher likelihood of fraud.
Transaction_ count (Feature Engineered)	Calculates how many transactions each account has. Why it helps: Fraudulent accounts typically perform fewer but more targeted transactions. A low or unusually high transaction count can be a red flag.
Net_balance_change (Feature Engineered)	Calculates whether the account gained or lost money in the transaction. This feature tells us if this type of transaction is usually fraudulent or not by calculating all of the isFraud transactions with this type of transaction.
Amount_std () (Feature Engineered)	Calculates the standard deviation of amounts for each account. Why it helps: Fraudsters often send irregular amounts. High variability (or lack thereof) in how much someone sends can help the model detect abnormal patterns.
type_fraud_rate	Calculates the Average fraud rate for that transaction type. Why it helps: Some transaction types are much more likely to be fraudulent. Giving the model this statistical fraud tendency by type provides valuable prior knowledge.

Predictive Questions: The two questions I am trying to predict are:

1. Can a model predict whether a transaction is fraudulent?

   Why?: I am asking this question because it focuses on identifying fraud as it happens. Creating machine learning models based on this question can help financial institutions identify and stop individuals' suspicious transactions in real time, mitigating financial loss.
2. Can a model detect recipient accounts that are commonly linked to fraudulent activity? (This question has been altered from my slides).

   Why?: I am asking this question because it targets long-term risk profiling of accounts. By detecting these fraudulent recipients I can help financial institutions monitor which destination accounts could display potentially fraudulent behavior across multiple transactions. With this information, the institutions can then launch an investigation into a particular flagged account.

Explanation of question 2:

[18]:
```
# Count the number of transactions per account
transactions_per_account = df.groupby('nameDest').size()

# Compute the average number of transactions per account
average_transactions = transactions_per_account.mean()

print(f"Average number of transactions per account: {average_transactions:.2f}")
```

```
Average number of transactions per account: 2.34
```

In question two, what I am doing is for the target variable “isFraud”, I assign a value of one to a recipient account nameDest, and if any transaction involving that account was fraudulent, and if it is no,t then I assign zero. Then I compute the average of each of the features to start off with, so I am taking the mean of each feature by grouping the dataset by destination account. This method works because each destination account has on average, around 2.34 transactions per destination account. Although more transactions per account in the future could make this model better, 2.34 destination accounts can still give us insight into fraudulent behavior from these receivers.

Modeling Process and Results:

Question 1:
For question one, at first, I tried various models. First, I started with logistic regression as a baseline model, as the problem I was trying to solve is a binary classification problem. Although the results of this model were not meaningful, some other models should be applied. The second model that I utilized was the random forest classifier because it handles high-dimensional data as well and handles overfitting well. Additionally, I had the model manage imbalance classes using the parameter class_weight='balanced'. Lastly, I used XGBoost, which, like random forest, is a decision tree model, and the model performed similarly to the random forest model, as seen below.

<u>The parameters I hypertuned for each model were:</u>
Logistic Regression:

- class_weight='balanced': which automatically adjusts weights inversely proportional to class frequencies. I chose this because the dataset is highly imbalanced as there are less fraudulent transactions than non-froud and this gives fraudulent transactions more influence.
- solver='liblinear': Is a linear classifier that supports data with millions of parameters which is why I chose it.
- max_iter=1000: meaning that it ran for 1000 iterations. I chose this because this is around when logistic regression converged.

Random Forest:

- n_estimators=100: Number of decision trees in the forest. I chose 100 because the model performs well with around 100 and to reduce training time I kept it at this value.
- class_weight='balanced': I chose this because it manages class imbalance by weighting the minority class “Fraud” higher.

XGBoost Classifier:

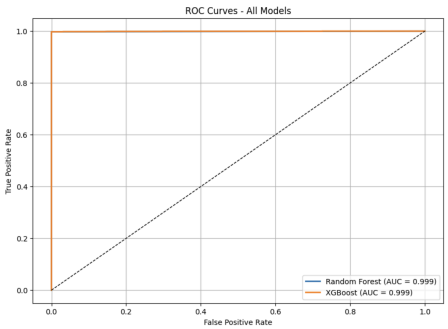
- eval_metric='logloss': Sets log loss and the loss function or evaluation metric. I chose this because log loss usually works well for binary classification and it seemed to work well for this model. Also, it tracks how well probabilities match.

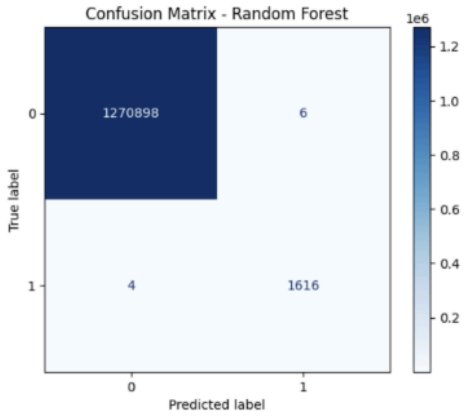
- scale_pos_weight=50: Increases the importance of correctly classifying fraud cases by weighting them 50 times more than normal transactions during training. I chose this because this helps XGBoost learn from the fraud cases that do not appear often by increasing the weight of their gradient.

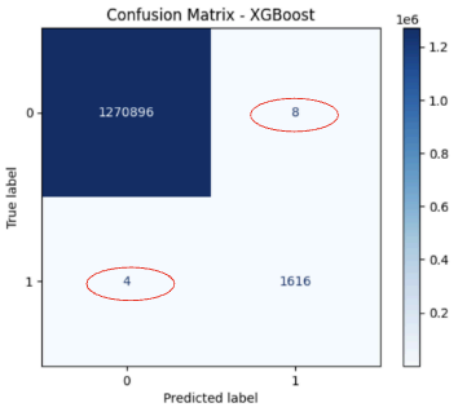
Results for Question 1:

Model Performance Metrics

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression (Base Model)	0.9648	0.0307	0.8716	0.0590
1	Random Forest	1.0000	0.9963	0.9975	0.9970
2	XGBoost	1.0000	0.9951	0.9975	0.9960







Description of Results (Q1):
Overall, the results show that Random Forest and XGBoost significantly beat the baseline Logistic Regression model by far in detecting fraudulent transactions. While Logistic Regression achieves a decent recall of 87.16%, it has a bad precision score of just 3.07%, meaning the model produces many false alarms, as revealed by its F1-score of 0.059. Both Random Forest and XGBoost, however, achieve a flawless accuracy of 1.000, and all their precision, recall, and F1-scores are higher than 0.99. ROC curves confirm their superior performance with AUC equal to 0.999 in both models, with almost perfect classification. These results are backed up by confusion matrices: Random Forest misclassified only 10

cases (6 false positives and 4 false negatives), whereas XGBoost misclassified 12 (8 false positives and 4 false negatives). Overall, the differences between the two are statistically insignificant. Finally, these results show that Random Forest and XGBoost outperform the base model of Logistic regression. In conclusion, I would probably choose the Random Forest model for this problem as it produces accurate results with a shorter run time than XGBoost.

Takeaways and Reflection:

What worked overall was feature engineering helped the models improve in both questions, although there is a large noticeable difference in question one. Also, playing around with the hyperparameters changed the result of what worked for every model as expected. What was surprising is how random forest did for both questions, although it makes sense since they are both similar questions. I was also surprised by how well Random Forest and XGBoost did for question one. What I would do differently next time is at the start, I would take more time understanding the data before jumping into making the models, so I truly understand what questions to ask and how to answer them with models. I learned how to apply machine learning models to unique problems like fraud detection, which shows that machine learning has many use cases. Overall, this project taught me how data framing and feature design impact model performance and real-world applicability.