Financial Transactions Fraud Detection

Dataset:
https://www.kaggle.com/code/shreyassatre/financial-transactions-fraud-detection-eda/input?select=Synthetic_Financial_datasets_log.csv

Data Preparation and Model Training Python Notebook:
https://www.kaggle.com/code/shreyassatre/financial-transactions-fraud-detection-eda

Methodology: Model Training

- **Data Understanding:** I gained insights into the data structure, distribution of features, and identified potential imbalances.
- **Data Preprocessing:** I cleaned, transformed, and prepared the data for effective model training, including creating a new feature called "amountCategory". This feature was designed to capture the significance of transaction amounts in fraud detection.
- **Model Selection and Training:** I considered various machine learning models suitable for handling imbalanced data for fraud detection, addressed class imbalance, and trained models on the preprocessed data.
- **Model Evaluation:** I evaluated the performance of trained models using appropriate metrics tailored to our specific use case.

Challenges Faced

1. **Data Imbalance:** Fraudulent transactions are typically a small minority compared to legitimate ones. This imbalance can skew model training and lead to poor performance in detecting fraudulent activity.
2. **Model Selection:** Choosing the right model for imbalanced data and real-time fraud detection requires considering factors like accuracy, interpretability, and computational efficiency.

Overcoming Data Imbalance

I employed several techniques to mitigate the impact of data imbalance:

- **Oversampling:** Duplicating minority class (fraudulent) transactions to create a more balanced dataset.
- **Undersampling:** Randomly removing data points from the majority class (legitimate) to match the size of the minority class.
- **SMOTE (Synthetic Minority Oversampling Technique):** Creating synthetic minority class samples based on existing ones.
- **Cost-Sensitive Learning:** Assigning higher weights to misclassifications of fraudulent transactions during model training.

Model Selection for Real-Time Fraud Detection

I considered the following models for real-time fraud detection. The selected models can handle imbalanced data through various techniques:

- **Logistic Regression:** Can be mitigated using techniques like oversampling, undersampling, or class weighting.
- **Random Forest:** Ensemble nature reduces the impact of individual tree biases.

- **Gradient Boosting Models:** Can be configured to assign higher weights to the minority class during training.

Procedure: Model Training and Evaluation

1. **Split the data** into training and testing sets.
2. **Applied data preprocessing** techniques, including the creation of the `amountCategory` feature, to address missing values, outliers, and handle categorical variables if applicable.
3. **Implemented chosen data balancing methods** (e.g., SMOTE or cost-sensitive learning).
4. **Trained models** using the balanced dataset.
5. **Evaluated model performance** on the testing set. I considered metrics like precision, recall, F1-score, and AUC-ROC (Area Under the ROC Curve) that are well-suited for imbalanced datasets.
6. **Compared model performance** and selected the model that best balances accuracy, interpretability, and efficiency for your specific real-time fraud detection requirements.

Procedure: Model Training and Evaluation

XGBoost Performs well with **95% Accuracy** and **99% AUC-ROC**

Technical Implementation:

Backend:

- ✓ **Model Development:** A Python-based machine learning model, specifically XGBoost, was trained on historical transaction data. The model was optimized to handle imbalanced datasets and effectively identify fraudulent patterns.
- ✓ **API Development:** FastAPI was employed to create a RESTful API that exposes endpoints for real-time fraud prediction. The API seamlessly integrates with the frontend and backend components.
- ✓ **Database:** SQLite, managed by SQLAlchemy, was used to store transaction data and customer information. This database is essential for tracking transactions, updating account balances.

Frontend:

- ✓ **User Interface:** An Angular-based frontend was developed to provide a user-friendly interface for monitoring transactions and viewing fraud alerts.
- ✓ **Visualization:** Chart.js was utilized to create visualizations of transaction data, helping to identify trends and patterns that may indicate fraudulent behavior.

Deployment:

- ✓ The backend was deployed on Render: https://transaction-fraud-detection-backend.onrender.com/docs (Note: As free instance spin down with inactivity, which can delay requests by 50s or more.)
- ✓ The frontend was deployed on Netlify: https://transaction-fraud-detection.netlify.app

Code Repository:

https://github.com/Shreyassatre/Transaction-Fraud-detection-backend

(Note: Although repository is named backend but it also has Frontend code)