Project Summary Report

Credit Card Fraud Detection in Fintech

Submitted By

Tamim Ahmed

Dept. of Electronics and Telecommunication Engineering

Chittagong University of Engineering and Technology

Credit Card Fraud Detection App

Introduction:

This report summarizes my work in the ML Developer Bootcamp (Milestone 1), developing a fraud detection system using the Kaggle Credit Card Fraud Detection dataset (284,807 transactions, 30 features: Time, Amount, V1–V28, 0.17% fraud). I have explored data, trained models, and built a Streamlit app, gaining skills in data analysis, modeling, and deployment.

Day 1: Data Exploration

I began by loading the dataset with Pandas and analyzing its structure. The class distribution revealed 284,315 non-fraud and 492 fraud transactions, confirming a 0.17% fraud rate. Using Seaborn, I visualized features like V3, V4, V10 and V14, which showed distinct patterns for fraud cases. The primary observation was the need to address class imbalance to ensure effective model training.

Day 2: Baseline Models

I preprocessed the data by scaling features with StandardScaler and splitting it into train-test sets. I trained four baseline models: Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), and Gaussian Naive Bayes. Performance was limited by class imbalance, with low recall across all models. Logistic Regression (AUC: 0.75-0.80) and Gaussian Naive Bayes (AUC: 0.70-0.75) performed poorly, while Decision Tree (AUC: 0.80-0.85) and KNN (AUC: 0.78-0.83) showed moderate results but high false negatives.

Day 3: Model Improvement

To address class imbalance, I have applied undersampling, creating a balanced dataset of 788 rows (394 fraud, 394 non-fraud). I trained three models: Random Forest, Decision Tree (re-evaluated), and Gradient Boosting. Random Forest and Gradient Boosting achieved high AUCs (0.90-0.95), with Gradient Boosting excelling in recall (0.75-0.90), critical for fraud detection. Decision Tree improved (AUC: 0.85-0.90) but was outperformed by ensembles. Feature importance analysis identified V14, V10, V3, and V4 as top predictors.

Day 4: Streamlit Web Application

I have developed a Streamlit app for fraud prediction. Users can upload CSV file of transections (Time, Amount, V1–V28) to get predictions and fraud case count. Display is limited to 100 rows for large CSVs, with full results downloadable.

Model Performance:

Model	AUC	Precision	Recall	Note
Logistic Regression	0.75–0.80	0.65–0.75	0.50–0.65	Poor recall due to imbalance.
Decision Tree (Day 2)	0.80–0.85	0.70–0.80	0.60–0.75	Moderate; high false negatives.
KNN	0.78–0.83	0.65–0.75	0.55–0.70	Scaling-dependent; intensive.
Gaussian Naive Bayes	0.70–0.75	0.60–0.70	0.50–0.65	Poor fit for PCA features
Decision Tree (Day 3)	0.85–0.90	0.75–0.85	0.65–0.80	Improved but outperformed.
Random Forest	0.90–0.95	0.80–0.95	0.70–0.85	Robust, balanced performance.
Gradient Boosting	0.90–0.95	0.75–0.90	0.75–0.90	High recall, best for fraud.

Challenges and Solutions:

The primary challenge was the severe class imbalance (0.17% fraud), which caused baseline models to predict non-fraud overwhelmingly, resulting in low recall. Undersampling balanced the dataset but reduced training data, improving recall. In the Streamlit app, a ValueError occured due to feature order mismatches was resolved by aligning inputs to the training data's order (Time, V1-V28, Amount).

Lesson Learned:

I have learned data analysis using Pandas and Seaborn to uncover patterns in imbalanced datasets as well as to train and evaluate diverse models (Logistic Regression, Decision Tree, KNN, Gaussian Naive Bayes, Random Forest, Gradient Boosting) with scikit-learn. Also, I have learned to build a Streamlit app for real-world predictions.

Conclusion:

The ML Developer Bootcamp provided a comprehensive introduction to machine learning, from data exploration to model deployment. I successfully built a fraud detection system, achieving high performance with Random Forest and Gradient Boosting (AUC: 0.90-0.95) and deploying an interactive Streamlit app.