Credit Card Fraud Detection Using a Hypertuned Random Forest Classifier

S.M. Shahriar
sayeem26s@gmail.com

***Abstract*—In this project, I developed and deployed a machine learning-based fraud detection system using a hypertuned Random Forest classifier. The objective was to identify fraudulent transactions from a highly imbalanced credit card dataset consisting of 284,807 transactions, with only 492 confirmed frauds. Through thorough preprocessing, feature scaling, and the implementation of oversampling using SMOTE, I achieved highly accurate results. The final model demonstrated an accuracy of 0.9995, precision of 0.8571, recall of 0.8293, F1-score of 0.8430, and an AUC of 0.9834. The trained model was then deployed in a Streamlit web application, allowing real-time CSV input and fraud prediction visualization. This work demonstrates the feasibility and effectiveness of scalable fraud detection pipelines in financial sectors using interpretable ensemble models.**

***Index Terms*—Credit Card Fraud, Random Forest, Machine Learning, Oversampling, Streamlit, AUC, F1 Score**

I. Introduction

Credit card fraud poses significant financial and reputational threats to individuals and institutions. Given the rarity of fraudulent activity in real-world financial datasets, traditional classification models often perform poorly due to severe class imbalance. My work focuses on tackling this challenge using a robust and interpretable machine learning approach—Random Forest.

I began by exploring the publicly available Kaggle credit card fraud dataset, followed by feature engineering, hyperparameter tuning, and full-stack application deployment. The high AUC (0.9834) and other strong metrics demonstrate the capability of this pipeline to accurately detect rare fraudulent events in a practical and scalable manner.

II. Dataset and Exploratory Analysis

The dataset used was the Credit Card Fraud Detection dataset from Kaggle, consisting of anonymized features (V1–V28), transaction amount, and time.

A. Data Inspection and Preprocessing

The dataset contains 284,807 transactions, of which only 492 are labeled as frauds, representing a fraud rate of just 0.172%. I observed that the `Time` column had little to no correlation with the fraud label and removed it to reduce noise. Fig. 1 illustrates the class imbalance between fraudulent and non-fraudulent transactions.

III. Methodology

To implement the end-to-end solution, I followed a structured methodology.

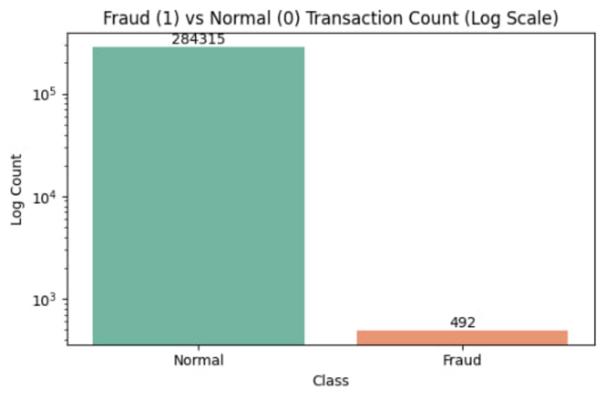


Fig. 1: Distribution of fraudulent vs. non-fraudulent transactions.

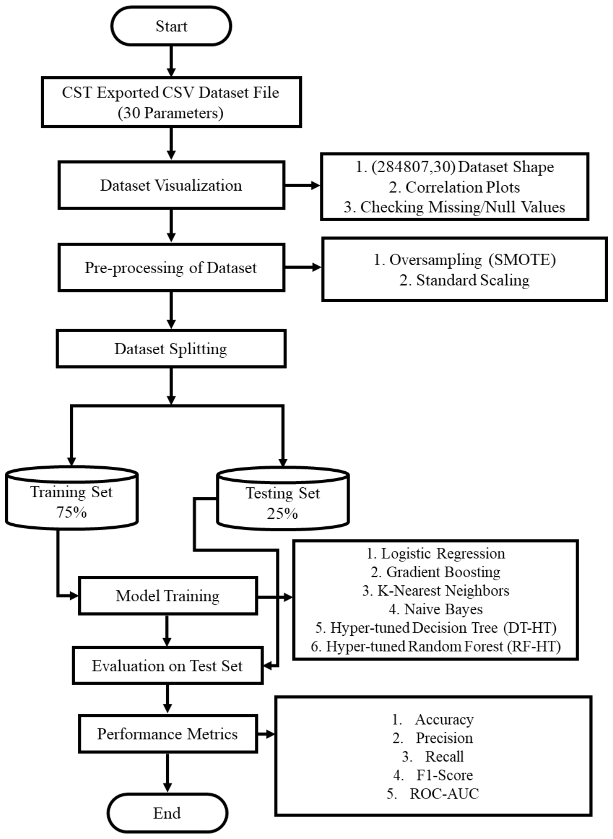


Fig. 2: Proposed methodology flowchart for the fraud detection system.

A. *Feature Scaling and Oversampling*

I standardized the `Amount` column using StandardScaler. Due to the extreme class imbalance, I employed SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic examples of fraudulent transactions. This significantly boosted recall and F1-score, which are critical for real-world fraud detection systems where false negatives are highly undesirable.

B. *Model Selection and Hyperparameter Tuning*

I evaluated Logistic Regression, Decision Trees, and Gradient Boosting, but Random Forest offered the best balance of performance and interpretability. I performed hyperparameter tuning with GridSearchCV to optimize `n_estimators`, `max_depth`, and `class_weight`, ensuring high generalization and low overfitting.

IV. Results and Evaluation

The hypertuned Random Forest model achieved strong performance, as shown in Table I. Fig. 3 illustrates the ROC curve, confirming excellent discrimination between classes.

TABLE I: Performance of Hypertuned Random Forest Classifier

Metric	Value
Accuracy	0.9995
Precision	0.8571
Recall	0.8293
F1 Score	0.8430
AUC	0.9834

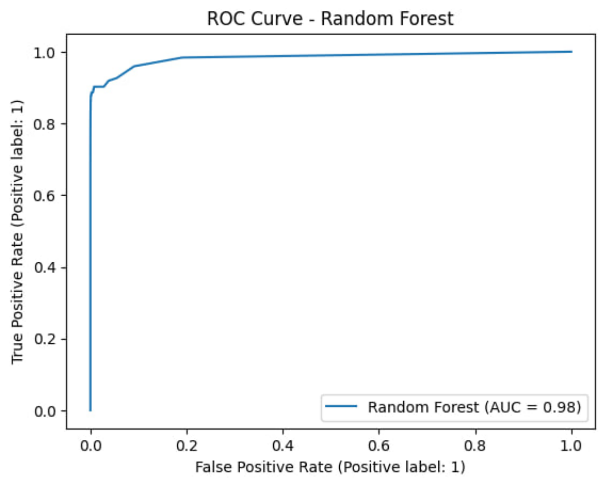
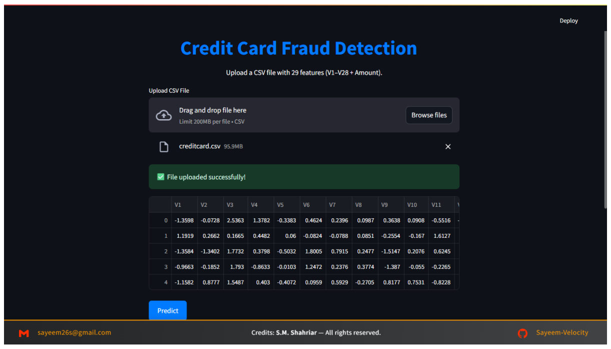


Fig. 3: ROC curve for the final Random Forest model.

V. Deployment and Interface

I developed a user-friendly web application using Streamlit, allowing users to upload CSV files containing transaction data and receive instant fraud predictions. The interface handles data preprocessing, prediction, and result visualization seamlessly. It provides a clean and accessible experience for non-technical stakeholders. Fig. 4 shows a screenshot of the deployed interface in action.



Upload a CSV file with 29 features (V1–V28 + Amount).

Upload CSV File

Drag and drop file here
Limit 200MB per file • CSV

creditcard.csv 95.9MB

File uploaded successfully!

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
0	-1.3598	-0.0728	2.5363	1.3782	-0.3383	0.4624	0.2396	0.0987	0.3638	0.0908	-0.5516
1	1.1919	0.2662	0.1665	0.4482	0.06	-0.0824	-0.0788	0.0851	-0.2554	-0.167	1.6127
2	-1.3584	-1.3402	1.7732	0.3798	-0.5032	1.8005	0.7915	0.2477	-1.5147	0.2076	0.6245
3	-0.9663	-0.1852	1.793	-0.8633	-0.0103	1.2472	0.2376	0.3774	-1.387	-0.055	-0.2265
4	-1.1582	0.8777	1.5487	0.403	-0.4072	0.0959	0.5929	-0.2705	0.8177	0.7531	-0.8228

sayeem26s@gmail.com   Credits: **S.M. Shahriar** — All rights reserved.   Sayeem-Velocity

Fig. 4: Streamlit App UI for fraud prediction.

VI. Challenges

The most significant challenge was the severe class imbalance in the dataset, with legitimate transactions vastly outnumbering fraudulent ones. This skew could bias model learning, so I used SMOTE to synthetically oversample the minority class. While effective, SMOTE occasionally introduced synthetic points that risked overfitting, which I countered through cross-validation and regularization.

Hyperparameter tuning with GridSearchCV was computationally intensive. Although time-consuming, it was necessary to ensure that the model achieved optimal performance without becoming overly complex or unstable.

Deployment posed additional technical challenges, especially when handling large CSV files in the Streamlit interface. To maintain responsiveness and low latency, I implemented caching and batched prediction strategies in the backend.

VII. Conclusion

This work demonstrates the effectiveness of a hypertuned ensemble model in detecting rare financial frauds. I achieved a high-performance solution through preprocessing, SMOTE oversampling, and Random Forest optimization. I further deployed the system as an interactive Streamlit web app, proving its real-world applicability. Future work could explore SHAP explanations for interpretability and cloud-based deployment for real-time monitoring.