Fraud Detection System

Introduction:

This document explains the credit card fraud detection system implemented in fraud_detect.ipynb. The system processes a transaction dataset, trains a Random Forest Classifier, and deploys a prediction API using FastAPI. Each step is described concisely, focusing on its purpose and significance, with minimal code.

Steps involved in Fraud Detection System:

Step 1: Loading and Inspecting the Dataset

Purpose: Load the credit card transaction dataset and examine its structure and class distribution to understand its characteristics and identify class imbalance.

Explanation: The dataset (creditcard.csv) is loaded into a pandas DataFrame. It contains 144,667 entries and 31 columns (30 features: Time, Amount, V1–V28; 1 target: Class). Most columns are floats, with Time as an integer. One row has missing values in V23 to Class. The Class column shows 99.8065% non-fraudulent (0.0) and 0.1935% fraudulent (1.0) transactions, indicating severe imbalance.

Significance: Identifying the dataset's structure and imbalance is critical, as it informs the need for techniques like resampling to handle rare fraudulent cases effectively.

Step 2: Preprocessing - Feature Scaling

Purpose: Standardize the Time and Amount columns to ensure features with different scales contribute equally to the model.

Explanation: A StandardScaler is applied to Time and Amount, transforming them to have a mean of 0 and a standard deviation of 1. The scaled values replace the originals in the DataFrame.

Significance: Scaling prevents features with larger ranges (e.g., Amount) from dominating the model, improving performance for algorithms sensitive to scale.

Step 3: Data Splitting

Purpose: Split the dataset into training and testing sets to train the model and evaluate its performance on unseen data.

Explanation: Rows with missing Class values are dropped. Features (X) and target (y) are separated, and the data is split into 80% training and 20% testing sets, with stratification to preserve the class imbalance. A random seed ensures reproducibility.

Significance: Splitting allows unbiased evaluation, and stratification ensures both sets reflect the dataset's imbalanced class distribution, critical for fraud detection.

Step 4: Handling Class Imbalance with Downsampling

Purpose: Balance the training set by downsampling the majority class (non-fraudulent) to match the minority class (fraudulent) size.

Explanation: The training data is split into majority (Class=0) and minority (Class=1) subsets. The majority is randomly downsampled without replacement to equal the minority's size. The balanced data is shuffled, and features and target are separated.

Significance: Balancing addresses the class imbalance, enabling the model to learn patterns for fraudulent transactions, which are otherwise underrepresented.

Step 5: Training the Random Forest Classifier

Purpose: Train a Random Forest Classifier on the balanced training data to predict transaction fraud.

Explanation: A Random Forest Classifier is initialized with a fixed seed and trained on the balanced training set.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train_bal, y_train_bal)
```

Significance: Random Forest is robust for high-dimensional data and effective for classification. Training on balanced data ensures sensitivity to fraudulent cases.

Step 6: Initial FastAPI Prediction Endpoint

Purpose: Create a simple FastAPI endpoint for fraud prediction based on transaction amount and type, as a prototype.

Explanation: A FastAPI app defines a /predict endpoint accepting a Transaction object with amount and type. It predicts “fraudulent” if amount > 1000, else “legit,” with static probabilities (0.85 or 0.15). Predictions are logged to a file.

Significance: This rule-based endpoint demonstrates API deployment but does not use the trained model, serving as a starting point for a more robust implementation.

Step 7: Placeholder for Model Retraining

Purpose: Define a function to retrain the model with new data to keep it updated.

Explanation: A function checks for new data, loads it, retrains the model, and saves it. The required functions (new_data_available, load_new_data, train_model) are not implemented.

Significance: This placeholder outlines a mechanism for continuous model updates, essential for adapting to evolving transaction patterns, but requires completion.

Step 8: Updated FastAPI Prediction Endpoint with Timing

Purpose: Enhance the /predict endpoint to use the trained model and measure prediction time.

Explanation: The endpoint is updated to predict using the Random Forest model and calculate response time. However, it references an undefined data variable, indicating a bug.

Significance: This step aims to integrate the model and monitor performance, but the bug (fixed in the next step) prevents proper execution.

Step 9: Final FastAPI Prediction Endpoint with Model Integration

Purpose: Deploy a complete FastAPI application using the trained model for real-time fraud predictions.

Explanation: The trained model is loaded from a file. A FastAPI app defines a /predict endpoint accepting a Transaction object with all 30 features (Time, Amount, V1–V28). The input is converted to a DataFrame, reordered to match training data, and passed to the model for prediction. The response includes the prediction (0 or 1) and fraud probability. Logging captures inputs and outputs. Ngrok provides a public URL, and Uvicorn runs the server.

Significance: This final implementation enables real-time fraud detection with the trained model, accessible via a public API. Logging ensures traceability, and Ngrok facilitates external access.

Conclusion:

The notebook builds a fraud detection system by preprocessing data, addressing imbalance, training a Random Forest model, and deploying a FastAPI endpoint. Addressing the identified issues and adding evaluation and error handling will enhance its production readiness.